Provable Data Possession in Single Cloud Server: A Survey, Classification, and Comparative Study

Ayad Barsoum
Computer Science Department
St.Mary's University
San Antonio, Texas, USA
abarsoum@stmarytx.edu

ABSTRACT

Storage-as-a-Service (SaaS) offered by cloud service providers is a *paid* facility that enables organizations to outsource their data to be stored on remote servers. Thus, SaaS reduces the maintenance cost and mitigates the burden of large local data storage at the organization's end. However, the fact that data owners no longer physically possess their sensitive data raises new challenges to the tasks of data confidentiality and integrity in cloud computing systems. Many researchers have focused on the problem of provable data possession (PDP), and proposed different schemes to audit data on remote storage sites.

In this paper, we investigate the concept of PDP and provide an extensive survey for different PDP schemes on a single cloud server. Moreover, the paper discusses the design principles for various PDP constructions, highlights some limitations, and present a comparative analysis for numerous PDP models. We classify PDP schemes into protocols for *static* data, and models that support outsourcing of *dynamic* data.

Keywords

Cloud computing, outsourcing data storage, cryptographic protocols, data integrity, verification techniques

1. INTRODUCTION

Cloud computing is a distributed computational model over a large pool of shared-virtualized computing resources (*e.g.*, storage, processing power, memory, applications, services, and network bandwidth), where customers are provisioned and de-provisioned recourses as they need. The architecture of cloud computing can be split in two: front-end and back-end. The front-end represents cloud customers, organizations, or applications (*e.g.*, web browsers) that use the cloud services. The back-end is a huge network of data centers with many different applications, system programs, and data storage systems. It is metaphorically believed that, cloud service providers (CSPs) have almost *infinite* computation power and storage capacity.
Cloud computing services can be categorized into: Application-as-a-Service (AaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [1]. The widely used model of cloud computing services is the AaaS model, in which the customers have access to the applications running on the cloud provider's infrastructure. Google Docs, Google Calendar, and Zoho Writer are known examples of this model. In the PaaS model, the customers can deploy their applications on the provider's infrastructure under condition that these applications are created using tools supported by the provider. The cloud service provider (CSP) hosts a set of software and development tools on its servers to be used by the developers to create their own applications. Google Apps is one of the best known PaaS models. IaaS model enables customers to rent and use the provider's resources (storage, processing, and network). Hence, the customers can deploy any applications including operating systems.
The cloud computing architecture can be deployed under different models [1]:

- *Public cloud*. The infrastructure of the CSP is publicly accessible by general customers and organizations in exchange for pre-specified fees according to the usage of the CSP's services.
- *Private cloud*. The cloud infrastructure is dedicated to an organization which may manage the infrastructure or leave this management to a third party.
- *Hybrid cloud*. The cloud infrastructure is composed of two or more clouds (private or public). The organizations provide and handle some internal and external resources. For example, an organization can use a public cloud service as Amazon Elastic Compute Cloud (Amazon EC2) to perform the general computation, while the data files are stored within the organization's local data center in a private cloud.
- *Community cloud*. The cloud infrastructure is available for shared use of several organizations supporting a specific community and having shared concerns (e.g., mission, security requirements, and compliance considerations)

The considerable attention of cloud computing paradigm is due to a number of key advantages: cost effectiveness, low management overhead, immediate access to a wide range of applications, flexibility to scale up/down information technology (IT) capacity, and mobility where customers can access information wherever they are, rather than having to remain at their desks.

1.1 Outsourcing Data Storage

In our current digital world, various organizations produce a large amount of *sensitive* data including personal information, electronic health records, and financial data. The amount of digital data is

increasing at a staggering rate; doubling almost every year and a half [2], and outpacing the storage ability of many organizations. This data often needs to be stored at multiple locations for a long time due to operational purposes and regulatory compliance. The local management of such huge amount of data is problematic and costly due to the requirements of high storage capacity and qualified personnel. While there is a steady drop in the cost of storage hardware, the management of storage has become more complex and represents approximately 75% of the total ownership cost [2]. Storage-as-a-Service (sort of IaaS) offered by CSPs is an emerging solution to mitigate the burden of large local data storage and reduce the maintenance cost by means of outsourcing data storage. Through outsourcing data storage scenario, organizations delegate the storage and management of their data to a CSP in exchange for pre-specified fees metered in GB/month. Such outsourcing of data storage enables organizations to store more data on remote servers than on private computer systems. In addition, some organizations may create large data files that must be archived for many years but are rarely accessed, and thus there is no need to store such files on the local storage of the organizations. More importantly, the CSP often provides better disaster recovery by replicating the data on multiple servers across multiple data centers achieving a higher level of availability. Therefore, many authorized users are allowed to access the remotely stored data from different geographic locations making it more convenient for them. A relatively recent survey indicates that IT outsourcing has grown by a staggering 79% as organizations seek to focus more on their core competencies and reduce costs [3].

1.2 Remote Data Storage Challenges

The fact that data owners no longer physically possess their sensitive data raises new challenges to the tasks of data confidentiality and integrity in cloud computing systems. Unauthorized access and misuse of customers' confidential data are serious concerns regarding data outsourcing; hence, it is of significant importance to be aware of data administrators (CSPs) and their extend of data access right. In some practical applications, data confidentiality is not only a privacy concern, but also a juristic issue. For example, in e-Health applications inside the USA the usage and exposure of protected health information should meet the policies admitted by Health Insurance Portability and Accountability Act (HIPAA) [4], and thus keeping the data private on the remote storage servers is not just an option, but a demand. The confidentiality feature can be guaranteed by the owner via encrypting the data before outsourcing to remote servers. As such, it is a crucial demand of customers to have a strong evidence that the cloud servers still possess their data and it is not being tampered with or partially deleted over time, especially because the internal operation details of the CSP may not be known to cloud customers.

The completeness and correctness of customers' data in the cloud may be at risk due to the following reasons. First, the CSP – whose goal is to make a profit and maintain a reputation – has an incentive to hide data loss (due to hardware failure, management errors, various attacks) or reclaim storage by discarding data that has not been or is rarely accessed. Second, a dishonest CSP might delete some of the data or might not store all data in a high performance storage required by the contract with certain customers, *i.e.*, place it on low cost (and hence slow) media. Third, the cloud infrastructures are subject to a wide range of internal and external security threats. Incidences of security breaches of cloud services surface from time to time [5, 6]. In short, although outsourcing data to the cloud is attractive from the view point of cost and complexity of long-term large-scale data storage, it does not offer sufficient guarantee on data integrity. This problem, if not properly handled, may hinder the successful deployment and wide acceptance of the cloud paradigm.

Once customers' data has been outsourced to remote servers, efficient verification of the completeness and correctness of the outsourced data becomes a formidable challenge. Traditional cryptographic primitives for data integrity and availability based on hashing and signature schemes are not applicable to outsourced data without having a local copy. It is impractical for the owners to download all stored data to validate its integrity; this would require an expensive I/O operations and immense communication overheads across the network. Therefore, efficient techniques are needed to verify the integrity of outsourced data with reduced communication, computation, and storage overheads. Consequently, many researchers have focused on the problem of provable data possession (PDP), and proposed different schemes to audit the data on remote storage sites.

This paper investigates the concept of PDP over a single remote server, discusses the design principles of different PDP protocols, highlights some limitations, and present a comparative study for numerous PDP schemes. We classify the PDP schemes according to the nature of the outsourced data: *static* data and *dynamic* data.

Paper organization. The remainder of the paper is organized as follow. The PDP concept is presented in Section 2. Section 3 provides different PDP schemes for single data copy. The comparative analysis is shown in Section 4. Concluding remarks are given in Section 5.

2. PROVABLE DATA POSSESSION

In this section, we describe the concept of provable data possession (PDP) and provide the dimensions of our classification of PDP schemes

2.1 Concept

PDP is a technique that allows an entity to prove that the data is in its possession for validating data integrity over remote servers. In a typical PDP model, the data owner generates some metadata/information for a data file to be used later for verification purposes through a *challenge-response* protocol with the remote/cloud server. The owner sends the file to be stored on a remote server which may be untrusted, and deletes the local copy of the file. As a proof that the server is still possessing the data file in its original form, it needs to correctly compute a response to a challenge vector sent from a verifier – who can be the original data owner or a trusted entity that shares some information with the owner. Shortly, PDP schemes allow a verifier to efficiently, periodically, and securely validate that a remote server – which supposedly stores the owner's potentially very large amount of data – is actually storing the data intact.

The problem of data integrity over remote servers has been addressed for many years and there is a simple solution to tackle this problem as follows. The data owner computes a message authentication code (MAC) of the whole file before outsourcing to a remote server. The owner keeps only the computed MAC on his local storage, sends the file to the remote server, and deletes the local copy of the file. Later, whenever a verifier needs to check the data integrity, he sends a request to retrieve the file from the archive service provider, re-computes the MAC of the whole file, and compares the re-computed MAC with the previously stored value. Alternatively, instead of computing and storing the MAC of the whole

file, the data owner divides the file F into blocks $\{b_1, b_2, \ldots, b_m\}$, computes a MAC σ_j for each block b_j: $\sigma_j = \text{MAC}_{sk}(j||b_j)_{1\leq j\leq m}$, sends both the data file F and the MACs $\{\sigma_j\}_{1\leq j\leq m}$ to the remote/cloud server, deletes the local copy of the file, and stores only the secret key sk. During the verification process, the verifier requests for a set of randomly selected blocks and their corresponding MACs, re-computes the MAC of each retrieved block using sk, and compares the re-computed MACs with the received values from the remote server [7]. The rationale behind the second approach is that checking part of the file is much easier than the whole of it. However both approaches suffer from a severe drawback; the communication complexity is linear with the queried data size which is impractical especially when the available bandwidth is limited.

2.2 Our Classification

In this sub-section we classify PDP schemes according to the nature of outsourced data. Some PDP models focus on archived and warehoused data, which is essential in many applications such as digital libraries and astronomical/medical /scientific/legal repositories. Such data are subject to infrequent change, so they are treated as *static*. Other PDP protocols handle *dynamic* behavior of data. Each PDP scheme has its own design principles: some schemes are based on the RSA model, some are based on bilinear pairing, some are based on homomorphic verifiable tags, some are based on hashing techniques, and others are based on authenticated data structures. Figure 1 outlines our classification for different PDP models.

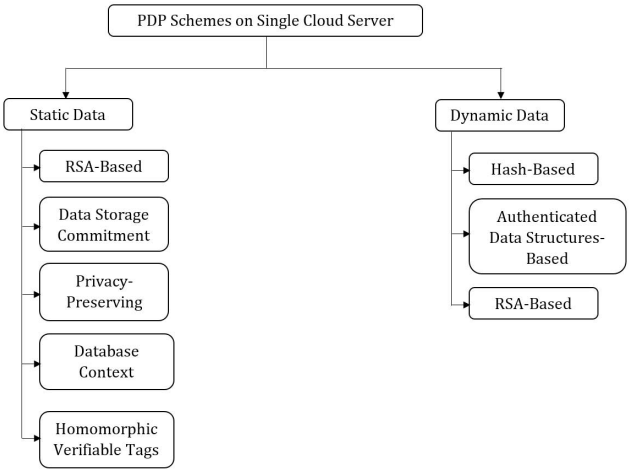

Fig. 1: Classification of PDP schemes.

3. PROVABLE SINGLE-COPY DATA POSSESSION

In this section, we review different PDP schemes for single data copy. We provide the rationale behind these schemes, their features and limitations. We start with PDP schemes for static data, then we direct our survey to models that deal with dynamic data.

3.1 Provable Static Data Possession

3.1.1 RSA-Based PDP Schemes. MAC-based approaches for remote data integrity are associated with high communication overhead. Deswarte *et al.* [8] thought of a technique to reduce the communication cost by using two functions f and H', where H' is a one-way function and f is another function. The relation between H' and f is that $f(C, H'(File)) = h(C||File)$, where h is any secure hash function and C is a random challenge number sent from the verifier to the remote server. Thus, the data owner has to compute $H'(File)$ and store it on his local storage. To audit the file, the verifier generates a random challenge C, computes $V = f(C, H'(File))$, and sends C to the remote server. Upon receiving the challenge C, the server computes $S = h(C||File)$ and sends the response S to the verifier. To validate the file integrity, the verifier checks $V \stackrel{?}{=} S$. At least one of the two functions f and H' must be kept secret because if both were public, it would be easy for a malicious server to compute and store only $H'(File)$ that is not the entire file, and then dynamically responds with a valid value $f(C, H'(File))$ that is not the expected one $h(C||File)$.
Unfortunately, Deswarte *et al.* [8] have not found such functions f, H', and h satisfying the desired verification rule. To workaround this problem, a finite number $\widetilde{N}$ of random challenges are generated offline for the file to be checked, and the corresponding responses $h(C_i||File)_{1\leq i\leq \widetilde{N}}$ are pre-computed and stored on the verifier local storage. To audit the file, one of the $\widetilde{N}$ challenges is sent to the remote server and the received response is compared with the pre-computed one (previously stored on the verifier side). However, this solution limits the number of times a particular data file can be checked by the number of random challenges $\widetilde{N}$. Once all random challenges $\{C_i\}_{1\leq i\leq \widetilde{N}}$ are consumed, the verifier has to retrieve the data file from the storage server to compute new responses, but this is unworkable.
Deswarte *et al.* [8] provided another protocol to overcome the problem of limited number of audits per file. In this protocol the data file is represented as an integer d. Figure 2 illustrates the scheme presented in [8].

Data owner:

- Represents the data file as an integer d
- Generates RSA modulus $N = pq$ (p & q are prime numbers)
- Pre-computes and stores $M = a^d \bmod N$ ($a \in_R \mathbb{Z}_N$)
- Sends the file value d to the remote server

Challenge Response

Verifier		Remote Server
1. Picks $r \in_R \mathbb{Z}_N$		
2. Computes a challenge $A = a^r \bmod N$	$\xrightarrow{A}$	
		3. Computes a response: $S = A^d \bmod N$
	$\xleftarrow{S}$	
4. Computes $V = M^r \bmod N$		
5. Checks $V \stackrel{?}{=} S$		

Fig. 2: The PDP protocol by Deswarte *et al.* [8].

Remark. The main limitation in the protocol of Deswarte *et al.* [8] is the computation overhead on the server side. In each verification, the remote server has to do the exponentiation over the entire file. Thus, if we are dealing with huge files, *e.g.*, in order of Terabytes (as most practical applications require) this exponentiation will be heavy. The data owner can reduce the exponent part in the computation $M = a^d \bmod N$ by utilizing the Fermat-Euler theorem [9],

where $a^d \equiv a^{d \bmod \phi(N)} \bmod N$ and $\phi(N) = (p-1)(q-1)$ is the Euler's totient function. The remote server cannot use this trick because $\phi(N)$ is not known in public.

Filho *et al.* [10] proposed a scheme to verify data integrity using the RSA-based homomorphic hash function. A function $\widehat{H}$ is homomorphic if, given two operations + and $\times$, we have $\widehat{H}(d + d')$ = $\widehat{H}(d) \times \widehat{H}(d')$. The protocol in [10] is illustrated in Figure 3.

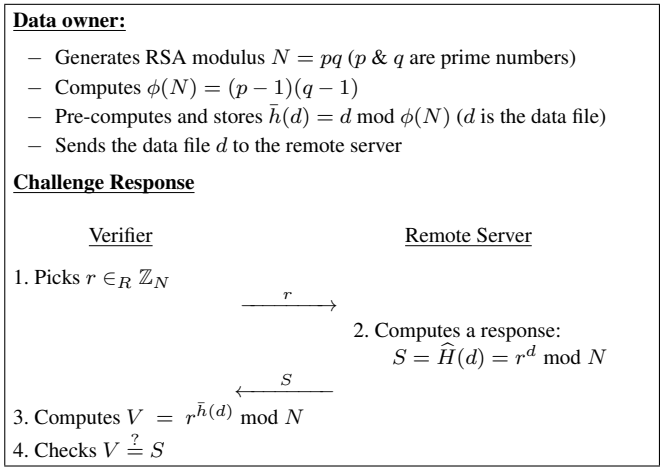

Fig. 3: The PDP protocol by Filho *et al.* [10].

Remark. The server's response $S = \widehat{H}(d)$ is a homomorphic function; $\widehat{H}(d+d') \equiv r^{d+d'} \equiv r^d r^{d'} \equiv \widehat{H}(d)\widehat{H}(d') \bmod N$. To find a collision for this hash function, one has to find two messages $d,\ d'$ such that $r^d \equiv r^{d'}$, *i.e.*, $r^{d-d'} \equiv 1 \bmod N$. Thus, $d - d'$ must be multiple of $\phi(N)$. Finding such two messages $d,\ d'$ is believed to be difficult since the factorization of N is unknown. The limitation of the protocol presented in [10] is similar to that of the protocol in [8]: the archive service provider has to exponentiate the entire data file, which is a heavy computation overhead especially for large files.

To circumvent the problem of exponentiating the entire file, Sebé *et al.* [11] presented a scheme to remotely verify data integrity by first fragmenting the file into blocks, fingerprinting each file block, and then using an RSA-based hash function on the blocks. Thus, the data file F is divided into a set of m blocks: $F = \{b_1, b_2, \ldots, b_m\}$, where m fingerprints $\{M_j\}_{1 \le j \le m}$ are generated for the file and stored on the verifier local storage. Their scheme does not require the exponentiation of the entire file. Figure 4 demonstrates the protocol of Sebé *et al.* [11].

Remark. Although the protocol presented by Sebé *et al.* [11] does not require exponentiation of the entire file, a local copy of the fingerprints – whose size is linear in the number of file blocks – must be stored on the verifier side. The verifier has to store $\{M_j\}_{1 \le j \le m}$, each of size $|N|$ bits consuming $m|N|$ bits from the verifier local storage, which may impede the verification process when using small devices like PDAs or cell phones. Moreover, this protocol supports only private verifiability, *i.e.*, only the data owner can challenge the remote server and validate the data possession. If there is

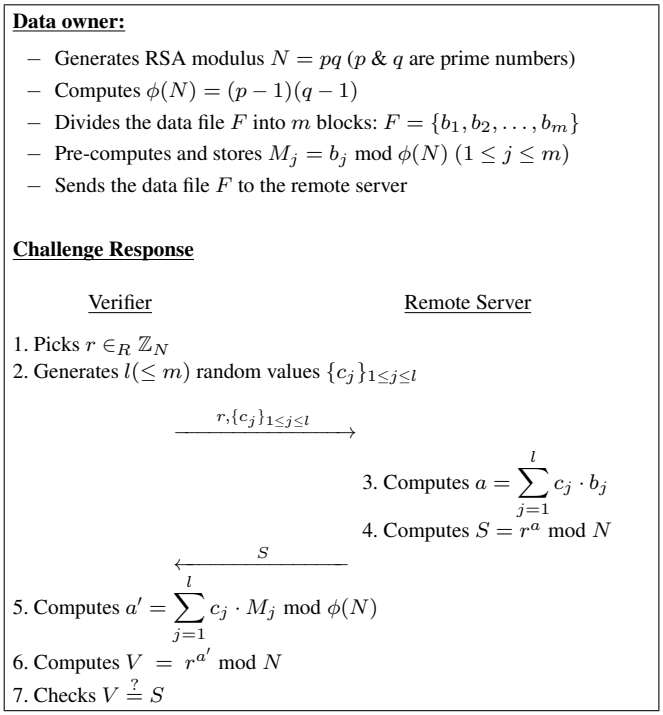

Fig. 4: The PDP protocol by Sebé *et al.* [11].

a dispute regarding data integrity, we cannot resort to a trusted third party auditor to resolve such a dispute.

3.1.2 Data Storage Commitment Schemes. Golle *et al.* [12] provided a scheme to verify data storage commitment, a concept that is weaker than integrity. They investigated "storage-enforcing commitment scheme". Through their scheme a storage server demonstrates that it is making use of storage space as large as the client's data, but not necessarily the same exact data. The storage server does not directly prove that it is storing a file F, but proves that it has committed sufficient resources to do so. Their scheme is based on n-Power Computational Diffie-Hellman (n-PCDH) assumption: for a group $\mathbb{Z}_p$ (p is a prime number) with a generator g, there is no known probabilistic polynomial time algorithm A that can compute g^{x^n} given $g^x, g^{x^2}, \ldots, g^{x^{n-1}}$ with non-negligible probability. Figure 5 illustrates the scheme of Golle *et al.* [12].

Remark. In [12] each file block $b_j \in \mathbb{Z}_p$ can be represented by $\lceil \log_2\ p \rceil$ bits, and thus the total number of bits to store the file $F = m\lceil \log_2\ p \rceil$ bits. For the storage server to cheat by storing all the possible values of f_k (*i.e.*, $m+1$ values), it needs $(m+1)\lceil \log_2\ p \rceil$ bits which is slightly larger than the size of the original file. The guarantee provided by the protocol in [12] is weaker than data integrity since it only ensures that the server is storing something at least as large as the original data file but not necessarily the file itself. In addition, the verifier's public key is about twice as large as the file.

3.1.3 Privacy-Preserving PDP Schemes. Shah *et al.* [13, 14] presented privacy-preserving PDP protocols. Using their schemes, an external third party auditor (TPA) can verify the integrity of files stored by a remote server without knowing any of the file contents. The data owner first encrypts the file, then sends both

Setup

- File $F = \{b_1, b_2, \ldots, b_m\}, b_j \in \mathbb{Z}_p$
- Let $n = 2m + 1$
- Secret key $sk = x \in_R \mathbb{Z}_p$
- Public key $pk = (g^x, g^{x^2}, \ldots, g^{x^n}) = (g_1, g_2, \ldots, g_n)$
- Data owner computes and stores $f_0 = \prod_{j=1}^{m} g_j^{b_j} \bmod p$

Challenge Response

Verifier		Remote Server
1. Picks a random $k \in [0, m]$		
	$\xrightarrow{k}$	
		2. Computes $f_k = \prod_{j=1}^{m} g_{j+k}^{b_j}$
	$\xleftarrow{f_k}$	
3. Checks $f_0^{x^k} \stackrel{?}{=} f_k$		

Fig. 5: The PDP protocol by Golle *et al.* [12].

the encrypted file along with the encryption key to the remote server. Moreover, the data owner sends the encrypted file along with a key-commitment that fixes a value for the key without revealing the key to the TPA. The primary purposes of the schemes presented in [13,14] are to ensure that the remote server is correctly possessing the client's data along with the encryption key, and to prevent any information leakage to the TPA which is responsible for the auditing task. Thus, clients – especially with constrained computing resources and capabilities – can resort to external audit party to check the integrity of outsourced data, and this third party auditing process should bring in no new vulnerabilities towards the privacy of client's data. In addition to the auditing task of the TPA, it has another primary task which is extraction of digital contents. For the auditing task, the TPA interacts with the remote server to check that the stored data is intact. For the extraction task, the TPA interacts with both the remote server and the data owner to first check that the data is intact then delivers it to the owner. The protocols presented by Shah *et al.* [13, 14] are illustrated in Figure 6.

Setup

- Data owner sends a key K and the encrypted file $E_K(F)$ to the remote server
- Data owner sends a key-commitment value g^K and the encrypted file $E_K(F)$ to the TPA (g is a generator for $\mathbb{Z}_p$)
- The TPA generates a list L of random values and HMACs: $L = \{(R_i, \widetilde{H}_i)\}_{1 \le i \le \widetilde{N}}$, $\widetilde{H}_i$ = HMAC$(R_i, E_K(F))$, and R_i is a random number.
- TPA keeps $\{L, h(E_K(F)), g^K\}$ and can discard $E_K(F)$ (h is a secure hash function)

TPA		**Remote Server**
Checking Data Integrity		
1. Picks any $(R_i, \widetilde{H}_i)$ from L and updates $L = L \backslash \{(R_i, \widetilde{H}_i)\}$		
	$\xrightarrow{R_i}$	
		2. Computes $\widetilde{H}_s$=HMAC$(R_i, E_K(F))$
	$\xleftarrow{\widetilde{H}_s}$	
3. Checks $\widetilde{H}_i \stackrel{?}{=} \widetilde{H}_s$		
Checking Key Integrit		
1. Generates $\beta \in_R \mathbb{Z}_p$		
	$\xrightarrow{g^\beta}$	
		2. Computes $W_s = (g^\beta)^K$
	$\xleftarrow{W_s}$	
3. Checks $(g^K)^\beta \stackrel{?}{=} (W_s)$		
Data Extraction		
	$\xleftarrow{D_s = E_K(F)}$	
• Checks the hash of its local cached copy: $h(E_K(F)) \stackrel{?}{=} h(D_s)$. If valid, sends $E_K(F)$ to the owner		
Key Extraction		
• Assume that the owner and the server agree on a shared random secret X		
	$\xleftarrow{K+X,\ g^X}$	
• Checks $g^{K+X} \stackrel{?}{=} g^K \cdot g^X$. If valid, sends $K + X$ to the owner		
• Owner gets $K = (K + X) - X$		

Fig. 6: The PDP protocols by Shah *et al.* [13, 14].

Remark. The protocols presented in [13, 14] achieve privacy-preserving towards third party auditing process and extract digital contents from remote servers, but have some limitations:

- Limited number of verifications for a particular data item (must be fixed beforehand).
- Storage overhead on the TPA; it has to store $\widetilde{N}$ hash values for each file to be audited.
- Lack of support for *stateless* verification; the TPA has to update its state (the list L) between audits to prevent using the same random number or the same HMAC twice.
- High communication complexity to retrieve $E_K(F)$ if the TPA wants to regenerate a new list of hash values to achieve unlimited number of audits.

Wang *et al.* [15] designed a lightweight TPA scheme using homomorphic linear authenticators (HLA). The main idea of their scheme is to integrate HLA with random masking technique to audit the outsourced data while keeping their privacy. In the scheme of [15], the TPA sends a challenge to check data integrity and the CSP computes a response μ to that challenge. Before sending the response μ to the TPA, the CSP blinds μ using a random mask, *i.e.*, $\mu' = \mu + r.h((u^x)^r)$, where μ' is the blinded blocks aggregation, r is a random mask, u is a public key, x is a private key, and h is a hash function.

3.1.4 PDP in Database Context. In the database outsourcing scenario, the database owner stores data at a storage service provider and the database users send queries to the service provider to retrieve some tuples/records that match the issued query. Data integrity is an imperative concern in the database outsourcing paradigm; when a user receives a query result from the service provider, it is crucial to verify that the received tuples are not being tampered with by a malicious service provider. Mykletun *et al.* [16] investigated the notion of signature aggregation to validate the integrity of the query result. Signature aggregation enables bandwidth- and computation-efficient integrity verification of query replies. In the scheme presented in [16], each database

record is signed before outsourcing the database to a remote service provider.
Mykletun *et al.* [16] provided two aggregation mechanisms: one is based on RSA [17] and the other is based on BLS (Boneh-Lynn-Shacham) signature [18]. For the scheme based on the RSA signature, each record in the database is signed as: $\sigma_j = h(b_j)^d \bmod N$, where h is a one-way hash function, b_j is the data record, d is the RSA private key, and N is the RSA modulus. A user issues a query to be executed over the outsourced database, the server processes the query and computes an aggregated signature $\sigma = \sum_{j=1}^{t} \sigma_j \bmod N$, where t is the number of records in the query result. The server sends the query result along with the aggregated signature to the user. To verify the integrity of the received records, the user checks $\sigma^e \stackrel{?}{=} \prod_{j=1}^{t} \sigma_j \bmod N$, where e is the RSA public key.
The second scheme presented by Mykletun *et al.* [16], which is based on the BLS signature [18] is similar to the first scheme but the record signature $\sigma_j = h(b_j)^x$, where $x \in_R \mathbb{Z}_p$ is a secret key. To verify the integrity of the received records, the user checks $\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}(\prod_{j=1}^{t} h(b_j), y)$, where g is a generator of the group $\mathbb{Z}_p$, $y = g^x$(public key), and $\hat{e}$ is a computable bilinear map.
Correctness and Completeness are imperative concerns in the database outsourcing paradigm. Completeness means that the service provider should send *all* records that satisfy the query criteria not just subset of them. The completeness requirement was not considered by the schemes presented in [16], and it has been addressed by other researchers (see for example [3, 19]).

Remark. The schemes provided in [16] depend on the retrieved records of the query result to verify the integrity of the outsourced database. On the other hand, efficient PDP schemes require *blockless* verification, *i.e.*, the verifier has to have the ability to validate data integrity even though he neither possesses nor retrieves any of the file blocks. Blockless verification is a main concern to minimize the required communication cost over the network.

3.1.5 PDP Schemes Based on Homomorphic Verifiable Tags. Ateniese *et al.* [20] presented a model to overcome some of the limitations of other PDP protocols: limited number of audits per file determined by fixed challenges that must be specified in advance, expensive server computation by doing the exponentiation over the entire file, storage overhead on the verifier side by keeping some metadata to be used later in the auditing task, high communication complexity, and lack of support for blockless verification. Ateniese *et al.* [20] provided a PDP model in which the data owner fragments the file F into blocks $\{b_1, b_2, \ldots, b_m\}$ and generates metadata (a tag) for each block to be used for verification. The file is then sent to be stored on a remote/cloud server, which may be untrusted and the data owner may delete the local copy of the file. The remote server provides a proof that the data has not been tampered with or partially deleted by responding to challenges sent from the verifier. The scheme presented in [20] provides *probabilistic* guarantee of data possession, where the verifier checks a random subset of stored file blocks with each challenge (spot checking).
Homomorphic verifiable tags (HVTs)/homomorphic linear authenticators (HLAs) are the basic building blocks of the PDP scheme presented in [20]. Briefly, the HVTs/HLAs are unforgeable verification metadata constructed from the file blocks in such a way that the verifier can be convinced that a linear combination of the file blocks is accurately computed by verifying only the aggregated tag/authenticator. In the work of [20], the authors differentiate between the concept of *public verifiability* and *private verifiability*. In public verifiability anyone – not necessarily the data owner – who knows the owner's public key can challenge the remote server and verify that the server is still possessing the owner's files. On the other side, private verifiability allows only the original owner to perform the auditing task. Two main PDP schemes are presented in [20]: sampling PDP (S-PDP) and efficient PDP (E-PDP) schemes. In fact, there is a slight difference between these two models, but the E-PDP scheme provides a weaker guarantee of data possession. The E-PDP protocol guarantees only the possession of the *sum* of file blocks and not necessarily the possession of each one of the blocks being challenged. Both protocols presented in [20] are illustrated in Figure 7.

I. S-PDP scheme

Setup

- $N = pq$ is the RSA modulus (p & q are prime numbers)
- g is a generator of QR_N (the set of quadratic residues modulo N)
- Public key $pk = (N, g, e)$, secret key $sk = (d, v)$, $v \in_R \mathbb{Z}_N$, and $ed \equiv 1 \bmod (p-1)(q-1)$
- π is a pseudo-random permutation, f is a pseudo-random function, H is a hash-and-encode function ($H : \{0,1\}^* \rightarrow QR_N$), and h is a cryptographic hash function.
- File $F = \{b_1, b_2, \ldots, b_m\}$
- Owner generates a tag T_j for each block b_j: $T_j = (H(v||j) \cdot g^{b_j})^d \bmod N$
- Owner sends $F = \{b_j\}_{1 \le j \le m}$ and $\{T_j\}_{1 \le j \le m}$ to the remote server

Challenge Response

Verifier		Remote Server
1. Picks two keys k_1(key for π), k_2(key for f), c(# of blocks to be challenged), and $g_s = g^s \bmod N (s \in_R \mathbb{Z}_N)$	$\xrightarrow{c,\ k_1,\ k_2,\ g_s}$	
		2. Computes challenged block indices: $\{j_i\} = \pi_{k_1}(i)_{1 \le i \le c}$
		3. Computes random values: $\{a_i\} = f_{k_2}(i)_{1 \le i \le c}$
		4. Computes $T = \prod_{i=1}^{c} T_{j_i}^{a_i} \bmod N$
		5. Computes $\rho = h(g_s^{\sum_{i=1}^{c} b_{j_i} \cdot a_i} \bmod N)$
	$\xleftarrow{T,\ \rho}$	
6. Computes $\{j_i\} = \pi_{k_1}(i)_{1 \le i \le c}$ and $\{a_i\} = f_{k_2}(i)_{1 \le i \le c}$		
7. Computes $\tau = \frac{T^e}{\prod_{i=1}^{c} H(v\|j_i)^{a_i}}$		
8. Checks $h(\tau^s \bmod N) \stackrel{?}{=} \rho$		

II. E-PDP scheme

The only difference between the E-PDP and the S-PDP is that : $\{a_i\}_{1 \le i \le c} = 1$, and thus

- Step 4 : $T = \prod_{i=1}^{c} T_{j_i} \bmod N$
- Step 5 : $\rho = H(g_s^{\sum_{i=1}^{c} b_{j_i}} \bmod N)$
- Step 7 : $\tau = \frac{T^e}{\prod_{i=1}^{c} H(v||j_i)}$

Fig. 7: The S-PDP and E-PDP protocols by Ateniese *et al.* [20].

Ateniese *et al.* [21] showed that the HLAs can be constructed from homomorphic identification protocols. They provided a "compiler-like" transformation to build HLAs from homomorphic identifica-

tion protocols and showed how to turn the HLA into a PDP scheme. As a concrete example, they applied their transformation to a variant of an identification protocol proposed by Shoup [22] yielding a factoring-based PDP scheme.

Remark.The schemes of Ateniese *et al.* [20] have resolved many constraints of other PDP protocols. However, their schemes are based on RSA, which make the HVTs relatively long; each file block has an HVT in order of $|N|$ bits. Thus, to achieve 128-bit security level, the generated tag should be of size 3072 bits. Shacham and Waters [23] presented an attack against the E-PDP scheme, which enables a malicious server to cheat with non-negligible probability requiring no more storage than an honest server to store the file.

3.2 Provable Dynamic Data Possession

One of the core design principles of outsourcing data is to provide dynamic scalability of data for various applications. In this section we describe different provable single-copy dynamic data possession (PSDDP) schemes, where the data owner can issue requests to update and scale the outsourced data.

3.2.1 Hash-Based PSDDP Schemes. Ateniese *et al.* [24] proposed a dynamic version of the PDP scheme based on cryptographic hash function and symmetric key encryption. Their scheme is efficient but allows only a fixed number of challenges due to the fact that through the scheme setup they come up with all future challenges and store pre-computed responses as tokens. These tokens can be stored either at the verifier side in a plain form or at the server side in an encrypted form. Block insertion in [24] cannot explicitly be supported (append operation is supported). Figure 8 summarizes the scheme presented in [24].

3.2.2 PSDDP Schemes Based on Authenticated Data Structures. Erway *et al.* [25] constructed a PSDDP scheme based on the PDP model of [20] to support provable updates of stored data files using rank-based authenticated skip lists. Their protocol supports block insertion by eliminating the index information in the tag computation of [20]. The purpose of using the rank-based authenticated skip list in [25] is to authenticate the tag information of the blocks to be updated or challenged.
In the PSDDP scheme of [25], the File F is fragmented into m blocks $\{b_1, b_2, \ldots, b_m\}$. A tag $\mathcal{T}(b_j)$ of block b_j is computed as $\mathcal{T}(b_j) = g^{b_j} \bmod N$ (N is the RSA modulus and g is an element of high order in $\mathbb{Z}_N^*$). The block representation $\mathcal{T}(b_j)$ is stored at the j^{th} bottom-level node of the authenticated skip list and the block itself is stored elsewhere by the server. The tags protect the integrity of file blocks, while the authenticated list ensures the security and integrity of tags.
During the challenge phase, the client requests the server to prove the integrity of randomly selected c blocks $\{b_{j_i}\}_{1\le j_1,\ldots,j_c\le m}$. The server sends the tags $\{\mathcal{T}(b_{j_i})\}_{1\le j_1,\ldots,j_c\le m}$ along with their search/verification paths. The server also sends a combined block $M = \sum_{i=1}^{c} a_i.b_{j_i}$, where $\{a_i\}_{1\le i\le c}$ are random values sent by the client as part of the challenge. The owner verifies the search/verification paths of the block tags using metadata $\mathcal{M}_c$, which is the label of the start node. Besides, the owner computes $T = \prod_{i=1}^{c} \mathcal{T}(b_{j_i})^{a_i} \bmod N$. Data integrity is valid only if the search paths are verified and $T = g^M \bmod N$.
The authenticated skip list is used to modify, insert, and delete the block tags achieving the dynamic behavior of the data file. Nodes of skip list along the search/verification path – from the start node to

Setup

- Data file F is a set of blocks $\{b_1, b_2, \ldots, b_m\}$
- g is a pseudo-random permutation, f is a pseudo-random function, and h is a cryptographic hash function. f is used to generate keys for g and to generate random numbers.
- E_K and E_K^{-1} are encryption and decryption algorithms under a key K
- Two master keys W and Z
- Data owner generates t random challenges and their corresponding responses/tokens $\{\nu_i\}_{1\le i\le t}$ as follows.
  for i = 1 to t ***do***
  1. Generate $k_i = f_W(i)$ and $c_i = f_Z(i)$
     / r is # of blocks per token */*
  2. $\nu_i = h(c_i, 1, b_{g_{k_i}(1)}) \oplus \cdots \oplus h(c_i, r, b_{g_{k_i}(r)})$
  3. $\nu'_i = E_k(ctr, i, \nu_i)$ */* ctr is an integer counter */*
- Owner sends the file $F = \{b_j\}_{1\le j\le m}$ and $\{\nu'_i\}_{1\le i\le t}$ to the server.

Challenge Response

Data owner — Remote Server

Begin challenge i

1. Generates $k_i = f_W(i)$ and $c_i = f_Z(i)$
   $\xrightarrow{k_i, c_i}$
2. $z = h(c_i, 1, b_{g_{k_i}(1)}) \oplus \cdots \oplus h(c_i, r, b_{g_{k_i}(r)})$
   $\xleftarrow{z, \nu'_i}$
3. Computes $\nu = E_K^{-1}(\nu'_i)$
4. Checks $\nu \stackrel{?}{=} (ctr, i, z)$

End challenge i

Dynamic Operations

Modify */* Assume that block b_j is to be updated to b'_j */*

Data owner — Remote Server

$\xleftarrow{1.\ \{\nu'_i\}_{1\le i\le t}}$

2. $ctr = ctr + 1$
3. ***for*** i = 1 to t ***do***
   3.1 $z'_i = E_K^{-1}(\nu'_i)$
   / if decryption fails, exit*/*
   / if z'_i is not prefixed by (ctr-1) and i, exit */*
   3.2 extracts ν_i from z'_i
   3.3 computes $k_i = f_W(i)$ and $c_i = f_Z(i)$
   / update all tokens even if they do not include the block to be updated */*
   3.4 ***for*** l = 1 to r ***do***
   if $(g_{k_i}(l) == j)$ ***then***
   $\nu_i = \nu_i \oplus h(c_i, l, b_j) \oplus h(c_i, l, b'_j)$
   3.5 $\nu'_i = E_k(ctr, i, \nu_i)$
   $\xrightarrow{j,\ b'_j,\ \{\nu'_i\}_{1\le i\le t}}$

Delete */* Assume that block b_j is to be deleted */*

The logic of Delete is similar to Modify but replaces the block to be modified with a special block "DBlock". So, the inner ***for*** loop (step 3.4) will be:

3.4 ***for*** l = 1 to r ***do***
if $(g_{k_i}(l) == j)$ ***then***
$\nu_i = \nu_i \oplus h(c_i, l, b_j) \oplus h(c_i, l, \text{DBlock})$

Insert

Physical insert is not supported. Append is allowed by viewing the data file as a matrix, and appending the new block in a round robin fashion.

Fig. 8: The PSDDP protocol by Ateniese *et al.* [24].

the node associated with the block to be updated – are only affected by the dynamic operations of file blocks.
In their work, Erway *et al.* [25] presented a variant of the PSDDP scheme using RSA trees instead of rank-based authenticated lists. Wang *et al.* [26] used Merkle hash trees [27] (instead of skip lists) and homomorphic linear authenticators (HLAs) built from BLS signatures [18] to construct a PSDDP scheme.
Zhang *et al.* [28] designed a PSDDP scheme utilizing a new data structure, which is called block update tree. The tree needs to be stored at both the owner and the remote server sides to mitigate the requirement of verification for each dynamic operation. The update tree has two main features: it is always balanced and its size is independent of the size of the outsourced data. The block update tree is a binary tree in which each node contains (i) node type (to indicate the type of dynamic operation), (ii) data block range (L, U), where the left child of a node has lower indices than L, and the right child has higher indices than U, (iii) offset R, which is used to specify the indices of blocks after insert and delete requests; and (iv) version number V represents the current version of the block.
Zhu *et al.* [29] proposed a cooperative provable data possession (C-PDP) based on using hash index hierarchy (HIH) and HLAs. The HIH is a hierarchical structure that is used to present the relationships among the data blocks of various storage service providers. The HLAs aggregates the responses generated from different CSPs into one response based on the sum of the challenges. This aggregation process reduces the communication overhead and achieves privacy preserving by hiding outsourced data location in the distributed remote servers.
In the architecture of the C-PDP scheme, one of the CSPs (or an independent server) works as an organizer. The role of the organizer includes managing other CSPs, initializing the auditing process, communicating with cloud customers, and aggregating all the responses received from from the CSPs into one response using the properties of the HLAs. The data owners are allowed to dynamically access and update their data for various applications, and the verification process is performed for the owners in hybrid clouds.

3.2.3 RSA-Based PSDDP Schemes. Hao *et al.* [30] adapted the protocol presented in [11] to support both data dynamic and public verifiability. The latter allows that anyone who knows the owner's public key can challenge the remote server and verify that the server is still possessing the owner's files. If a dispute regarding data integrity occurs between the owner and the CSP, a third party auditor can determine whether the data integrity is maintained or not. This third party auditing process should bring in no new vulnerabilities towards the privacy of owner's data. The protocol presented in [30] ensures that the data is kept private during the third party verification, where no private information contained in the data is leaked. Figure 9 summarizes the protocol presented in [30].

Setup

- $N = pq$ is the RSA modulus (p and q are prime numbers)
- g is a generator of Q_{RN} (Q_{RN} is the set of quadratic residues modulo N)
- Public key $pk = (N, g)$ and secret key $sk = (p, q)$.
- f is a pesudo-random function
- File $F = \{b_1, b_2, \ldots, b_m\}$.
- Data owner generates a tag D_j for each block b_j, where $D_j = g^{b_j} \bmod N$
- The tags are stored on the owner side and the file is sent to the server.

Challenge Response

Verifier		Remote Server
1. Generates a random key r		
2. Computes $g_s = g^s \bmod N$ ($s \in_R \mathbb{Z}_N$)	$\xrightarrow{r, g_s}$	
		3. Generates random coefficients $\{a_j = f_r(j)\}_{1 \le j \le m}$
		4. Computes $P = (g_s)^{\sum_{j=1}^{m} a_j . b_j} \bmod N$
	$\xleftarrow{R}$	
5. Generates a set of random coefficients $\{a_j = f_r(j)\}_{1 \le j \le m}$		
6. Computes $\acute{P} = \prod_{j=1}^{m} (D_j^{a_j} \bmod N) \bmod N$		
7. Computes $V = \acute{P}^s \bmod N$		
8. Checks $V \stackrel{?}{=} P$		

Dynamic Operations

Modify: */* Assume that block b_j is to be updated to b'_j */*

- Server updates b_j to b'_j
- Owner computes a new block tag $D'_j = g^{b'_j} \bmod N$.
  So, the new block tags are $\{D_1, D_2, \ldots, D'_j, \ldots, D_m\}$

Insert: */* A block $\hat{b}$ is to be inserted at index j (or appended at the end) */*

- The server updates its file to be $\{b_1, b_2, \ldots, b_j, \hat{b}, \ldots, b_{m+1}\}$ (insert: $b_{j+1} = \hat{b}$) or $\{b_1, b_2, \ldots, b_m, \hat{b}\}$ (append).
- The owner computes a new block tag $\widehat{D} = g^{\hat{b}} \bmod N$, and changes the block tags to $\{D_1, D_2, \ldots, D_j, \widehat{D}, \ldots, D_{m+1}\}$ (insert: $D_{j+1} = \widehat{D}$) or $\{D_1, D_2, \ldots, D_m, \widehat{D}\}$ (append)

Delete : */* Assume a block at position j is to be deleted*/*

- Server deletes the block b_j
- Owner deletes the corresponding tag D_j

Fig. 9: The PSDDP protocol by Hao *et al.* [30].

4. COMPARATIVE ANALYSIS

In this sub-section we provide a comparison of PDP schemes for single data copy. The comparison is held from different perspectives:

- Owner pre-computation: the operations performed by the data owner to process the file before being outsourced to a remote server.
- Verifier storage overhead: the extra storage required to store some metadata on the verifier side to be used later during the verification process.
- Server storage overhead: the extra storage on the server side required to store some metadata – not including the original file – sent from the owner.
- Server computation: the operations performed by the server to provide the data possession guarantee.
- Verifier computation: the operations performed by the verifier to validate the server's response.
- Communication cost: bandwidth required during the challenge response phase.
- Unbounded challenges: whether the scheme allows unlimited number of auditing the data file, or a fixed number of challenges.

Table 1. : Comparison of PDP schemes for a file containing m blocks, c is the number of blocks to be challenged, t is the number of tags, and $\widetilde{N}$ is a finite number of random challenges.

Scheme	[8]	[10]	[11]	[12]	[13, 14]	[20]	[15]	[24]	[25]	[26]	[30]	[29]
Owner pre-computation	EXF	$O(1)$	$O(m)$	$O(m)$	$O(1)$	$O(m)$	$O(m)$	$O(t)$	$O(m)$	$O(m)$	$O(m)$	$O(m)$
Verifier storage overhead	$O(1)$	$O(1)$	$O(m)$	$O(1)$	$O(\widetilde{N})$	—	—	—	—	—	$O(m)$	—
Server storage overhead	—	—	—	—	—	$O(m)$	$O(m)$	$O(t)$	$O(m)$	$O(m)$	—	$O(m)$
Server computation	EXF	EXF	$O(c)$	$O(m)$	$O(1)$	$O(c)$	$O(c)$	$O(c)$	$O(c \log m)$	$O(c \log m)$	$O(m)$	$O(c)$
Verifier computation	$O(1)$	$O(1)$	$O(c)$	$O(1)$	$O(1)^{\dagger}$	$O(c)$	$O(c)$	$O(1)$	$O(c \log m)$	$O(c \log m)$	$O(m)$	$O(c)$
Communication cost	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(c \log m)$	$O(c \log m)$	$O(1)$	$O(c)$
Unbounded challenges	✓	✓	✓	✓	×	✓	✓	×	✓	✓	✓	✓
Fragmentation	×	×	✓	✓	×	✓	✓	✓	✓	✓	✓	✓
Type of guarantee	DET	DET	PRO‡	DET	DET	PRO‡	PRO	PRO	PRO	PRO	DET	PRO
Dynamic support	×	×	×	×	×	×	×	✓*	✓	✓	✓	✓

† Verifier pre-computation is $O(\widetilde{N})$ to generate a list L of HMACs.
‡ This scheme can be easily modified to support deterministic guarantee.
* Block insertion cannot explicitly be supported.

- Fragmentation: whether the file is treated as one chunk, or divided into smaller blocks.
- Type of guarantee: whether the guarantee provided from the remote server is deterministic guarantee, which requires to access all file blocks, or probabilistic guarantee that depends on spot checking.
- Dynamic support: whether the scheme supports outsourcing dynamic data.

We use the notations EXF to indicate the EXponentiation of the entire File, DET to indicate deterministic guarantee, and PRO to indicate probabilistic guarantee. Table 1 summarizes the comparison from the aforementioned perspectives. In Table 1, m represents the number of file blocks, c is the number of blocks to be challenged, t is the number of tags generated by the owner, and $\widetilde{N}$ is a finite number of random challenges (used for schemes that allow only for fixed number of challenges).

5. SUMMARY AND CONCLUDING REMARKS

In this paper, we have investigated the concept of PDP as a technique to verify the integrity of data stored on remote sites. We have provided an extensive survey and a comparative analysis for numerous PDP schemes on a single cloud server. The paper also discusses the design principles for different PDP models and highlights some limitations. In our study, we have addressed PDP protocols for static data, and PDP schemes that handle dynamic behavior of outsourced data over cloud servers. Through PDP models for dynamic data, the data owner is able to send requests to the remote server for updating/scaling the stored data. The verifier is enabled to make sure that the outsourced data is consistent with the most recent modifications issued by the owner.
We have provided a comparative analysis for different PDP schemes. The comparative analysis was done from different perspectives: the storage overhead on both the verifier and server sides, the computation overhead on the server side to prove data possession, the verifier's computations complexity to check server responses, the communication cost to send a challenge vector and receive a response; and the permission for unlimited number of data audits.

6. REFERENCES

[1] P. Mell and T. Grance, "Draft NIST working definition of cloud computing," Online at http://csrc.nist.gov/groups/SNS/cloud-computing/index.html, 2009.

[2] A. Singh and L. Liu, "Sharoes: A data sharing platform for outsourced enterprise storage environments," in *Proceedings of the 24th International Conference on Data Engineering, ICDE*. IEEE, 2008, pp. 993–1002.

[3] M. Xie, H. Wang, J. Yin, and X. Meng, "Integrity auditing of outsourced data," in *VLDB '07: Proceedings of the 33rd International Conference on Very Large Databases*, 2007, pp. 782–793.

[4] 104th United States Congress, "Health Insurance Portability and Accountability Act of 1996 (HIPAA)," Online at http://aspe.hhs.gov/admnsimp/pl104191.htm, 1996.

[5] N. Gohring, "Amazon's S3 down for several hours," Online at http://www.pcworld.com/businesscenter/article/142549/amazons_s3_down_for_severalhours.html, 2008.

[6] B. Krebs, "Payment processor breach may be largest ever."

[7] K. Zeng, "Publicly verifiable remote data integrity," in *Proceedings of the 10th International Conference on Information and Communications Security*, ser. ICICS '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 419–434.

[8] Y. Deswarte, J.-J. Quisquater, and A. Saïdane, "Remote integrity checking," in *6th Working Conference on Integrity and Internal Control in Information Systems (IICIS)*, S. J. L. Strous, Ed., 2003, pp. 1–11.

[9] M. Lassak and S. Porubsky, "Fermat-Euler theorem in algebraic number fields," *Journal of Number Theory*, vol. 60, no. 2, pp. 254–290, 1996.

[10] D. L. G. Filho and P. S. L. M. Barreto, "Demonstrating data possession and uncheatable data transfer," Cryptology ePrint Archive, Report 2006/150, 2006.

[11] F. Sebé, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, and J.-J. Quisquater, "Efficient remote data possession checking in critical information infrastructures," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 8, 2008.

[12] P. Golle, S. Jarecki, and I. Mironov, "Cryptographic primitives enforcing communication and storage complexity," in *FC'02: Proceedings of the 6th International Conference on Financial Cryptography*, Berlin, Heidelberg, 2003, pp. 120–135.

[13] M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan, "Auditing to keep online storage services honest," in *HOTOS'07: Proceedings of the 11th USENIX workshop on Hot topics in operating systems*, Berkeley, CA, USA, 2007, pp. 1–6.

[14] M. A. Shah, R. Swaminathan, and M. Baker, "Privacy-preserving audit and extraction of digital contents," Cryptology ePrint Archive, Report 2008/186, 2008.

[15] C. Wang, S. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *Computers, IEEE Transactions on*, vol. 62, no. 2, pp. 362–375, Feb 2013.

[16] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced databases," *Trans. Storage*, vol. 2, no. 2, 2006.

[17] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 26, no. 1, 1983.

[18] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, London, UK, 2001, pp. 514–532.

[19] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2006, pp. 121–132.

[20] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2007, pp. 598–609.

[21] G. Ateniese, S. Kamara, and J. Katz, "Proofs of storage from homomorphic identification protocols," in *ASIACRYPT '09: Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security*, Berlin, Heidelberg, 2009, pp. 319–333.

[22] V. Shoup, "On the security of a practical identification scheme," in *EUROCRYPT'96: Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques*, Berlin, Heidelberg, 1996, pp. 344–353.

[23] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ser. ASIACRYPT '08. Springer-Verlag, 2008, pp. 90–107.

[24] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *SecureComm '08: Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, New York, NY, USA, 2008, pp. 1–10.

[25] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *CCS '09: Proceedings of the 16th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2009, pp. 213–222.

[26] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *ESORICS'09: Proceedings of the 14th European Conference on Research in Computer Security*, Berlin, Heidelberg, 2009, pp. 355–370.

[27] R. C. Merkle, "Protocols for public key cryptosystems," *IEEE Symposium on Security and Privacy*, vol. 0, p. 122, 1980.

[28] Y. Zhang and M. Blanton, "Efficient dynamic provable possession of remote data via balanced update trees," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '13. New York, NY, USA: ACM, 2013, pp. 183–194.

[29] Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 12, pp. 2231–2244, Dec 2012.

[30] Z. Hao, S. Zhong, and N. Yu, "A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability," *IEEE Transactions on Knowledge and Data Engineering*, vol. 99, no. PrePrints, 2011.