Verifying Remote Data Integrity in Peer-to-Peer Data Storage: a Comprehensive Survey of Protocols

Nouha Oualha[*], Jean Leneutre[†] and Yves Roudier[‡]

[*]*CEA, LIST*
Communicating Systems Laboratory
Bât. 451 – PC 94, F91191 Gif sur Yvette Cedex, France
nouha.oualha@cea.fr

[†]*Institut Telecom - TELECOM ParisTech*
Dep. Informatique et Réseaux, CNRS LTCI-UMR 5141, 46 rue Barrault, 75634 PARIS cedex 13, France
leneutre@telecom-paristech.fr

[‡]*EURECOM*
Dep. Réseaux et Sécurité, 2225, Route des Crêtes, 06650 Valbonne Sophia Antipolis, France
roudier@eurecom.fr

Abstract This paper surveys protocols that verify remote data possession. These protocols have been proposed as a primitive for ensuring the long-term integrity and availability of data stored at remote untrusted hosts.

Externalizing data storage to multiple network hosts is becoming widely used in several distributed storage and P2P systems, which urges the need for new solutions that provide security properties for the remote data. Replication techniques cannot ensure on their own data integrity and availability, since they only offer probabilistic guarantees. Moreover, peer dynamics (i.e., peers join and leave at any time) and their potential misbehavior (e.g., free-riding) exacerbate the difficult challenge of securing remote data. To this end, remote data integrity verification protocols have been proposed with the aim to detect faulty and misbehaving storage hosts, in a dynamic and open setting as P2P networks.

In this survey, we analyze several of these protocols, compare them with respect to expected security guarantees and discuss their limitations.

Index Terms *Peer-to-peer, distributed data storage, cryptographic protocols, data integrity.*

1 Introduction

Peer-to-peer (P2P) applications are built using the techniques and algorithms that consider a server-less communication paradigm. Becoming very popular with services like P2P file sharing including audio/video streaming, these applications have spurred a lot of interest in the research community because of the stimulating challenge of establishing security guarantees in a dynamic, self-organizing and open system. For instance, a P2P data storage application (like Wuala[*], AllMyData Tahoe[†], UbiStorage[‡], or Cucku[§]) which offers peers the possibility to store their data in the network, should provide the availability and integrity of the stored data. Such guarantees cannot be solely ensured by replication techniques, in particular, if data damage can be caused by the dynamic or selfish behavior of storage peers, not only by their accidental faults or failures. Additionally, peers may be reluctant to freely offer a large amount of storage, in order to sustain high data replication rates. Data replication techniques should be, therefore, supported by other low-resource primitives. These primitives aim to detect corrupted data and trigger re-replication of data if it is necessary. This type of primitives is provided with remote data possession verification protocols that allow to periodically check data integrity at remote storage hosts.

Compared to grid storage or cloud storage, P2P storage relies on holders being totally untrusted. The verification protocol aims then to detect data corruption or destruction caused not only by accidental faults or crashes at these holders, but more importantly by their voluntary misconduct.

Besides data security issues, the verification protocols may also assist cooperation incentive mechanisms for P2P storage, as this type of protocols allows evaluating peer behavior. Mechanisms using reputation-based or payment-based incentives (e.g., [22], [21]) stimulate peer contribution with storage resources. The reaction of these mechanisms (i.e., reward or punish) toward a peer that has agreed to store other peers' data will rely on such peer behavior evaluation provided by the verification protocol.

Designing secure and efficient verification protocols has

[*] http://wua.la/en/home.html
[†] http://allmydata.org/
[‡] http://www.ubistorage.com/
[§] http://www.cucku.com/

attracted quite a lot of researches from multiple fields: works have varied from designing secure local memory (e.g., [3], [4], and [17]) and secure storage at remote servers (e.g., [6], [7], [8], [10], [27], [28], and [32]) to securing cloud storage applications (e.g., [5], [30], and [31]). The needs in each field being specific, result in diverse kinds of such protocols. This survey has a three-fold objective: it analyses the different protocols that have been proposed so far, categorize them in different groups, and finally compare them with respect to several security and performance requirements, given beforehand. A focus is given to those protocols that can be deployed in a P2P data storage setting.

The remainder of this paper is organized as follows. Section II defines protocols verifying remote data integrity with the requirements that should be provided, and identifies taxonomy for their classification. Section III surveys proposed verification protocols and sorts them following the taxonomy given in section II. Section IV analyzes and compares these classes of protocols in terms of security, efficiency, and scalability considerations, and eventually reveals tradeoffs between these considerations. Section V outlines interesting extensions that have been proposed for these protocols. Finally, concluding remarks are presented in section VI.

2 Principles of remote data integrity verification protocols

P2P storage applications offer peers the opportunity to store, backup or archive their data in the P2P network. Such applications should ensure data integrity and availability on a long term basis. This objective requires developing appropriate primitives for detecting dishonest peers taking an unfair advantage of the self-organizing storage infrastructure. Assessing such a behavior is an objective of remote data possession verification protocols.

2.1 Overview

We consider a P2P storage application in which a peer, called the data *owner*, replicates some data and stores the replicas at several peers, called data *holders*. The latter entities have agreed to keep these replicas for a predefined time period negotiated with the owner. Their commitment is periodically checked by *verifiers*. The verifiers are appointed by the data owner who may play also this role.

Holders and verifiers may not correctly fulfill their roles because of their failure or their misbehavior (e.g., free-riding). Though, verifier misbehavior is generally less common because of the lightweight operations at their side and the potential participation of the owner to the verification process.

The retained practice of verification protocols takes the form of an interactive *proof of knowledge* protocol where the holder tries to convince the verifier that he holds the data. The protocol consists of challenge-response messages. In a typical interaction (as illustrated in Fig. 1) between a verifier and one holder (or multiple holders), the holder is periodically prompted to respond to a time-variant challenge as a proof that it is holding its commitment, the very data.

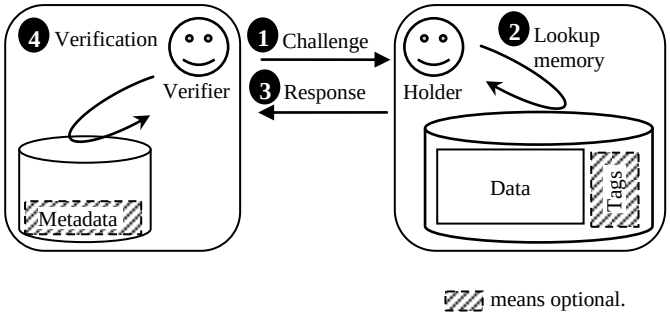

Fig. 1 Verifier-holder proof-of-knowledge interaction: 1) verifier sends a challenge message to the holder, 2) the holder computes the response based on data and tags and 3) sends the response back to the verifier, and finally 4) the verifier validates the response using some metadata. Tags and metadata are optional information i.e., one of these information or both may be used by certain protocols.

To bring the verification process to work, verifiers and/or holders generally hold some additional information, which may consist of data digests or keys.

2.2 Challenges and goals

P2P data storage may take the shape of P2P data backup where data is simply stored in multiple copies at the storage hosts; whereas the original version is still stored at the owner. Additionally, distributed storage and distributed file system applications where the original version does not necessarily exist at the owner, should be also considered.

The verification task cannot be limited to the actual owner's operations; it can be delegated to other network peers that are not necessarily trusted, in order to off-load owners' work and to mitigate their intermittent connection and potential failure.

Simple integrity checks, which make sense only with respect to a potentially defective yet trusted server, are not sufficient in this context. Holders in P2P systems are autonomous and may then misbehave by, for instance, destroying the data that they have promised to keep. By verifying data possession remotely, it is possible to detect voluntary data destructions by holders. The verification has to be efficient: in particular, verifying remotely the presence of data should not require transferring it back in its entirety; it should neither make it necessary to store the entire data at the verifier.

Enforcing the periodic holder storage verification has implications on the organizational design, performance, and security of the remote data integrity verification protocol. The design goals to achieve secure, efficient and scalable verification can be summarized as the following:

Requirement R1 - Sound verification: If the data (or a portion of the data) has been destroyed by the holder, the latter can only pretend it is storing them to the verifier with a small probability. The holder may be tempted to destroy the data that it has promised to keep, in order to harm the storage system or simply to optimize its storage resources. It may also be prone to failure or damage.

Requirement R2 - Complete verification: If both the verifier and the holder properly follow the protocol,

holder's proof is considered as valid. The verification process does not tolerate false negatives. Soundness and completeness properties are satisfied by a proof of knowledge protocol.

Requirement R3 - Dataless verification: The verifier should not store the whole or a portion of the data to carry out the verification task. It may however keep some metadata information. For efficiency concerns, no challenged data should be retrieved in its entirety by the verifier during the verification process.

Requirement R4 - Quick verification: To avoid connection problems between the verifier and the holder, the verification should be processed quickly, i.e., the computation complexity of the challenge generation, and especially the creation of the response and its validation should be reduced as low as possible.

Requirement R5 - Stateless verification: The verifier should not need to keep state information between audits throughout the long term of data storage.

Requirement R6 - Public verifiability: Anyone, not just data owners should be enabled with the capability of verifying the integrity of some data on demand. Since deploying several trusted and dedicated servers in the network is quite expensive, the ability to take part in the verification process should be handed out to any peer in the network.

Regarding requirement R1, some verification protocols provide deterministic guarantees by verifying the stored data in its entirety. Others check a piece of the data randomly chosen at each verification operation. Thus, they offer a probabilistic assurance that increases with the periodic iteration of the verification process.

Other optional requirements can be considered like devising a verification protocol that allows an unlimited number of challenges. The importance of such requirement depends on the duration of the data storage. With this requirement, regular interaction between the owner and the verifier is not needed to carry on the verification process in a long-term basis.

Realizing requirements R2, R3, R4, and R5 allow increasing the desired general goals; on the other hand, requirements R1 and R6 present tradeoffs with respect to two different general goals. For R1, if the verification is probabilistic, the holder computes a response message based on a portion of the data. The computation at the holder side is then efficient, but the detection of data corruption is only probabilistic. For R6, the delegation of the verification task from the owner to other peers should be controlled, in order to particularly avoid Denial of Service (DoS) attacks whereby a dishonest verifier floods the holders with challenge messages. Such attacks can be mitigated by making the owner produce verification certificates that grant, the verifiers that possess such certificates, permission to check holder's storage.

2.3 Taxonomy

The use of a specific terminology for remote data possession verification protocols (remote integrity checking [8], demonstration of data possession [10], proofs of data possession [2], or proofs of retrievability [12]) emphasizes how the storage and communication overhead requirements differ between verification primitives for secure remote storage and classical proof of knowledge protocols.

Efforts by Dodis et al. in [33] endeavor to make a distinction between protocols that are bounded in the number of allowed verification operations, so-called *information-theoretic bounded-use* protocols or *computational bounded-use* protocols, and protocols that permit an infinite number of verification operations, referred to by *computational unbounded-use* protocols. The former type of protocols relies on pre-computed random challenge-response pairs that are either stored at the verifier in the information-theoretic bounded-use protocols or secretly stored at the holder in the computational bounded-use protocols. This type of verification information will be called in the rest of this survey *metadata*.

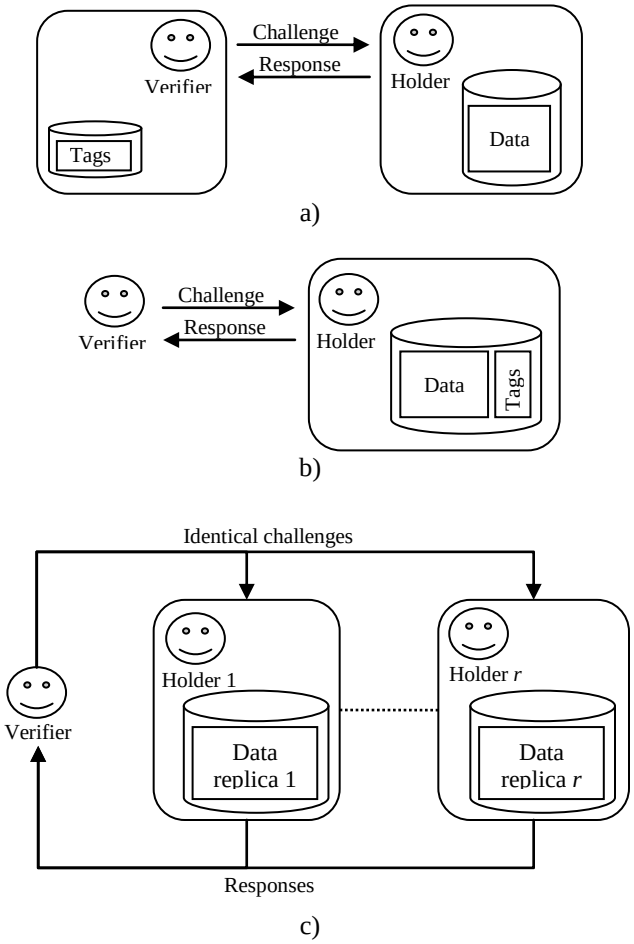

Fig. 2 Tag-based (a or b) vs. data replication-based (c) verification: (a or b) whereby the verifier checks one holder based on some tags which are either stored by the holder (encrypted) or the verifier and (c) whereby the verifier checks all data replicas at multiple holders.

To classify remote data possession verification protocols, we will not rely on their performance features but we will use different aspects related to the metadata that is used for data verification. The metadata information is generally produced by the data owner before being sent to the verifier.

Tag-based vs. data replication-based verification: tag-based verification relies on tags which are supplementary information attached to each copy of the stored data. They are either stored at the verifier (Fig. 2.a) or at the holder (Fig. 2.b) with proofs of authenticity (signatures). The verification is performed over one single storage peer at a time. On the other hand, data replication-based verification checks all the peers hosting the data replicas, with no additional information (see Fig. 2.c). It is, however, dependant on the used replication technique.

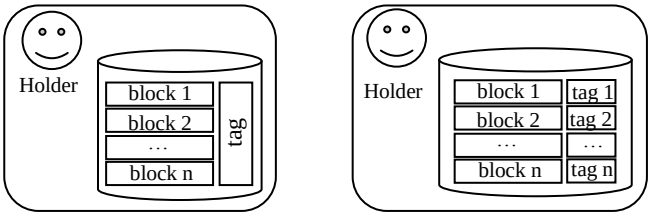

Fig. 3 Entire data dependant tags (a) vs. data block dependant tags (b). Such tags are generally stored by the holder.

Data-dependant tag vs. data-independent tag: in tag-based verification, tags are generally produced by the data owner. These tags can be either derived from the original data or completely generic. In the second case, tags can be re-used for the verification of another different data.

Entire data vs. data block dependant tag: there may be only one tag that is used to relate to the entire data or multiple tags with each tag corresponding to just one block of the data (refer to Fig. 3). The latter tag construction is more suited to support data dynamic operations like update, deletion, or insertion at block level granularity.

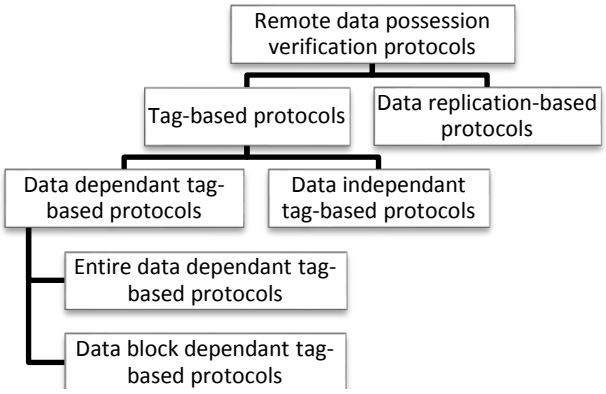

Fig. 4 Taxonomy of data possession verification protocols

The taxonomy applied in this survey is summarized in Fig. 4.

3 Classification

With P2P storage, any peer in the network can store its data at other peers, which are not necessarily trusted; however the behavior of these latter can be evaluated through the adoption of a verification routine through which the integrity of the stored data is periodically checked by verifying peers. The literature abounds with verification protocols for data integrity. We will present these protocols following the taxonomy illustrated in sub-section 2.3. The classification is recapitulated in Table 2. We will first start this section by giving a description of protocols verifying integrity of data stored locally, referred to by *local memory checking*. We deem these protocols as important, since they have paved the way to the design of protocols verifying remote data integrity.

Table 1 Used notations

Symbol	Description
T	Data tag
i	Data block index
T_i	Tag of the block of index i
d	Data
d_i	Data block of index i
f	Pseudo-random function
H	Hash function
g	Generator of a cyclic multiplicative group
x, α, s	Numbers kept secret from the holder

The key notations, summarized in Table 1, are used to describe the protocols throughout the remainder of this paper.

3.1 Earlier related work

Earlier verification schemes concentrated on the problem of securing the integrity of data stored at a local untrusted memory. A potential premise of these schemes originates from memory checking protocols (e.g., [4]). A memory checker aims at detecting any error in the behavior of an unreliable data structure while performing the user's operations. The checker steps between the user and the data structure. It receives the input user sequence of "store" and "retrieve" operations over data symbols that are stored at the data structure. The checker checks the correctness of the output sequence from the structure so that any error in the output operation will be detected with high probability. For this, it may use either its reliable storage (non-invasive checker) or the data structure (invasive checker). In [4], the checker stores hash values of the user data symbols at its reliable storage. Whenever the user requests to store a symbol, the checker computes the hash of the response of the data structure and keeps the hash value. If the user requests to retrieve a symbol, the checker computes the hash value of the response and compares it with the stored hash value. The job of the memory checker is to recover and to check responses originating from an unreliable storage, not to check the correctness of the whole stored data. With the checker, it is possible to detect corruption of one symbol (usually one bit) per user operation.

With incremental cryptographic algorithms, the checker is able to detect changes made to a whole document using one small information, the tag. The tag is a small secret stored at a reliable storage that relates to the complete stored document and that is quickly updatable if the user makes modifications. Bellare *et al.* in [3] propose several incremental schemes in which the tag is either an XORed sum of randomized document symbols or a leaf in a search tree built from a

message authentication algorithm applied to each symbol. These schemes provide tamper-proof security of the user document in its entirety. However, they require recovering the whole data which is not practical for remote data verification because of the high communication overhead.

The approach described in [17] or so-called *authenticator* differently comprehends the data possession verification problem. It extends the memory checker model by making the verifier check the consistency of the entire document in an encoded version. To achieve this, the document is encoded, to be stored at the unreliable storage disk. Small tags (or *fingerprints* in [17]) are generated, to be stored at the reliable storage disk. Based on tags, the verifier checks whether it is possible to recover the document without actually decoding it. The authors of [17] propose a construction of the authenticator where there is a public encoding of the document consisting of index tags of this form:

$$T_i = f_{seed}(i \cdot d_i)$$

for each encoded value bit d_i having f_{seed} a pseudorandom function with a *seed*. The seed is kept secret by the verifier. The authenticator is repeatedly used to verify, for a selection of random indices, whether the tags correspond to the encoding values. The detection of document corruption is thus probabilistic but improved with the encoding of the document. Moreover, the query complexity is proportional to the number of indices requested.

These presented verification schemes are the first to suggest checking data integrity. They are, however, not applicable for remote data integrity checking because they require the data to be transmitted in its entirety to the verifier. Yet, they laid the foundation by suggesting for instance an interactive proof of knowledge protocol between the verifier and the prover that provides soundness and completeness requirements defined in section 2.2. The main improvement that will be introduced by the following protocols is the ability for the storage host to provide a short and fresh proof of data integrity and availability for the verifier: this is to prevent him from storing only the resulted response instead of data.

3.2 Entire data dependant tag-based protocols

The majority of verification protocols carry on with the idea of [3] to associate data with some metadata in the form of authentication tags. These tags are generally derived by the owner from the actual data with some secret, thus allowing both authenticating their origin and proving their integrity. The size of data tag is generally small, and may be therefore kept by the verifier. The typical operation for this category of protocols is described in Fig. 5.

The first solution described in [8] requires pre-computed results of challenges to be stored at the verifier, where a challenge corresponds to the hashing of the data concatenated with a random number. This information is considered as a data dependant tag. The protocol requires low storage overhead at the verifier, yet it allows only a fixed number of challenges to be performed.

A similar scheme can be built using polynomials as described in [19]. The protocol relies on the uniqueness of the solution of the interpolation polynomial problem. The storage peer must resolve (n+1) equations using the n data blocks and a random challenge number. The solution is computed using the Lagrange interpolation. The verifier possesses the solution of this problem computed beforehand by the owner. The number of verification operations is limited here again to the number of solutions of interpolation problems stored by the verifier.

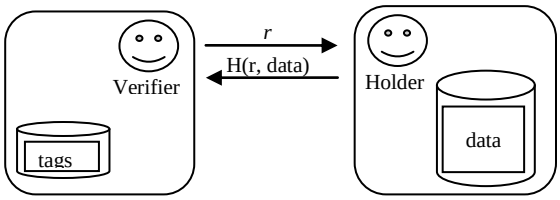

Fig. 5 Entire data dependant tag-based verification: the holder computes the response as the result of a particular function H with the data and the verifier's fresh challenge r as input, the verifier validates the response using the relation binding the tag to the data.

The second solution described in [8] requires little storage at the verifier side and no additional storage overhead at the holder side; yet it makes possible to generate an unlimited number of challenges. This solution (inspired from RSA) has been also proposed by Filho and Barreto in [10]. It makes use of a key-based homomorphic hash function F. A construction of F is also presented as $F(m)=g^m \bmod N$ where N is an RSA modulus such that the size of the message m is larger than the size of N. In this solution, the verifier keeps the tag: $T=g^d$. In each challenge, a nonce $N=g^r$ is generated by the verifier and sent to the prover (a nonce is a unique and randomly chosen value). The prover combines the nonce with the data using F to prove the freshness of the answer and obtains $R=N^d$. The prover's response will be compared by the verifier with a value computed over T. Since the verifier can perform the following operation:

$$T^r = (g^d)^r = (g^r)^d$$

[23] relies on elliptic curve cryptography [13] to construct the homomorphic function $F(.)$. Having P a generator of a special elliptic curve and m the message: $F(m)=m.P$. The verifier's challenge is a random point that will be multiplied by the data to compute the response.

In these two later protocols, the holder proves data possession by performing computation with complexity of order n for data with n symbols. To reduce the computational burden on the holder, the authors of these protocols have proposed to perform these expensive operations on small fragments of data, since these operations are generally exponential additive. The verification uses then tags that are related to a data fragment instead of the whole data.

Another protocol providing deterministic verification but with computation operations on data blocks, is proposed in [19]. The Diffie-Hellman based protocol allows remote checking using a small tag derived as:

$$T = \prod\nolimits_{i=1}^{n} {c_i}^{d_i}$$

where $\{d_i\}_{1\le i\le n}$ is the set of data blocks and $\{c_i\}_{1\le i\le n}$ is the set of random coefficients that are solely derived from a generator g of a cyclic multiplicative group and a secret number x known to the verifier. The challenge message consists of a set of

coefficients derived from the original coefficients and a random number *r*. The holder computes a response in the form of a new tag using the fresh coefficients and in particular using the data blocks$\{d_i\}_{1\le i\le n}$.

3.3 Data block dependant tag-based protocols

The exponentiation operation used in the RSA solution of [8] makes the whole data as an exponent. To reduce the computing time of verification, Sebé *et al.* in [28] propose to trade off the computing time required at the prover against the storage required at the verifier. The data is split in a number *n* of blocks $\{d_i\}_{1\le i\le n}$, the verifier holds $\{T_i=g^{d_i}\}_{1\le i\le n}$ and asks the prover to compute a sum function of the data blocks $\{d_i\}_{1\le i\le n}$ a challenge $N=g^r$ and *n* random coefficients $\{c_i\}_{1\le i\le n}$ generated from a new seed handed out by the verifier at each challenge. The response is computed as:

$$R = N^{\sum_{i=1}^{n} c_i \times d_i}$$

When the verifier receives holder's response, it compares *R* with:

$$\prod_{i=1}^{n} (T_i)^{c_i \times r}$$

The index *n* is the ratio of tradeoff between the storage kept by the verifier and the computation performed by the prover. The protocol in [25] (extended version of [23]) allows such adaptation whereby the holder performs point multiplication operations with just a data block each, and then linearly combines the results with random coefficients. Furthermore, the basic solution can be still improved as described in [7]; though the verification method is probabilistic i.e., only a randomly chosen portion of data is verified. The holder will be storing tags of:

$$T_i = g^{d_i + s_i}$$

where s_i is a random number kept secret by the verifier. A given tag T_i allows to authenticate the origin and the integrity of a block of index *i*. The holder periodically constructs compact forms of the data blocks and corresponding tags using time-variant challenge sent by the verifier. The authors of [7] argue that this solution achieves a good performance.

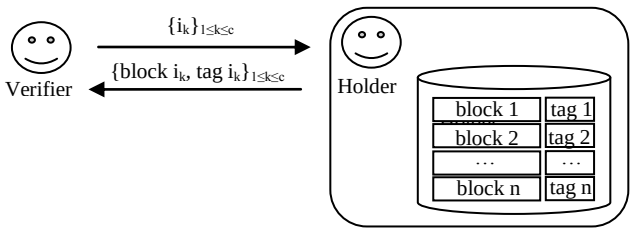

Fig. 6 Data block dependant tag-based verification: the verifier chooses a random set of block indexes (c indexes) as a challenge sent to the holder, then this latter sends back the corresponding blocks along with their tags.

[20] proposes a probabilistically-built solution (as described in Fig. 6). The index tags are formulated as block signatures where the verifier keeps their corresponding public key. Signatures are indeed generated by the data owner; though the role of the verifier can be carried out by this latter or any peer that possesses the public key. The verifier checks one data block per verification operation. It may check more than one data block and then the probability of data damage detection increases exponentially; but the response message sent to the verifier grows linearly with the number of challenges checked. The owner has to determine the right value for blocks' size or their number that achieves a good compromise between the accuracy of the verification and its costs (particularly in terms of communication overhead).

The PDP (Provable Data Possession) scheme in [2] improves the probabilistic model by presenting a new form of tags:

$$T_i = (h(v,i).\, g^{d_i})^{ks} \bmod N$$

where *h*(.) is a hash function, *v* a secret random number known only by the owner and the verifier, *N* an RSA modulus with *ks* being the owner's signature key, and *g* a generator of the cyclic group of $\mathbb{Z}_N^*$. With such homomorphic verifiable tags, any number of tags chosen randomly can be compressed into just one value by far smaller in size than the entire set, which means that communication complexity is independent of the number of indices requested per verification.

[29] further enhances the PDP protocol by considering compact tags that are associated with each data block d_i having the following form:

$$T_i = \alpha.\, d_i + s_i$$

where α and s_i are random numbers. The verifier requests random blocks from the remote holder and obtains a compact form of the blocks and their associated tags such that it is able to check the correctness of these tags just using α and the set $\{s_1, s_2, \ldots\}$ that are kept secret.

The protocol proposed by [32] is a bilinear pairing-based verification protocol (refer to [18] for a survey on bilinear mapping techniques). Using experimentation, the author in [32] shows that its approach outperforms PDP. In the proposed scheme, the data is divided into blocks that are organized into super-blocks. Each super-block is associated with a tag. The tag of a given super-block of index *i* is derived as:

$$T_i = (f_2(i,k).\, g_1^{\sum_j f_1(j,t).d_{i,j}})^{\frac{1}{x+z}}$$

where f_1 and f_2 are two pseudo-random functions that output results in respectively the first and the second group in the bilinear map's input, *z* is the file key, $d_{i,j}$ is the j^{th} block pertaining to the super-block, and finally *t* is a secret random number known only to the owner. All these functions and values except *t* are public. The generated tags evoke Ateniese *et al.*'s tags in [2] but instead of using RSA public keys, they rely on a secret key *x* shared only by the owner and the corresponding public key g_2^x where g_2 is a generator of the second group in the bilinear map's input. The use of bilinear mapping allows the verifier to check a subset of the challenged super-blocks without the need to know *x*. The use of a double block hierarchy makes it possible to improve the tradeoff between the tag size and the probability of corruption detection, since super-blocks are the information that is actually being periodically checked.

The authors of [7] described several schemes, some of them being hybrid construction of the existing and earlier presented schemes. For instance, they proposed that the holder store the data along with a redactable signature of the data: i.e., it is possible to derive the signature of any data block from the

signature of the entire data. The scheme allows the holder to compute a valid signature of any data block requested by the verifier.

Authenticating hash tables are fundamental data structures that generally optimally answer to membership queries. These tables have been used to check remote data integrity. An approach using Merkle trees [16] was proposed by Wagner and reported in [11]. The data stored at the holder is expanded with a Merkle hash tree on data blocks and the root of the tree is kept by the verifier. The verification process checks the possession of one data block chosen randomly by the verifier that requests also a full path in the hash tree from the root to this random block.

3.4 Data independent tag-based protocols

Some approaches consider tags which are not generated from data. The SEC (Storage Enforcing Commitment) scheme in [11] for instance aims at remote data deterministic verification using the following tags that are kept at the holder along with the data:

$$T = (g^x, g^{x^2}, \dots, g^{x^n})$$

where x is a secret key known to the verifier and g is a generator of a multiplicative group of same size as data blocks. The tags are independent of the stored data, but their number should be larger than two times the number m of data blocks (n=2×m+1). Verifier's challenge is a random value that will be used to shift the indexes of tags to be associated with the data blocks when constructing the response by the holder.

The POR protocol (Proof of Retrievability) in [12] explicitly expresses the independency between the data and the metadata used for verification (tags). The protocol is based on verification of sentinels which are random values independent of the owner's original data. These sentinels are disguised among data blocks. The verification is probabilistic with the number of verification operations allowed being limited to the number of sentinels.

These protocols introduce an extra storage overhead. This overhead can be, however, limited by reusing tags for the storage of different data.

3.5 Data replication-based protocols

Generally, data replication techniques are of great interest for verification protocols, since they improve the probability of data recovery in case the approach does not detect the destruction of some parts of the stored data.

There are several protocol propositions where the verifier is storing a data copy like the holder. This type of protocols is pertinent for a backup application having the owner as the verifier. For example, in the solution of [15] where the verifier is storing the same copy as the holder, the verifier requests a block out of the stored data from the holder. The response is checked by comparing it with the valid block stored at the verifier's disk space. In another simple proposed protocol in [6], the holder sends the MAC (Message Authentication Code) of data as a response to the verifier's challenge. The verifier sends a fresh nonce as the key for the message authentication code.

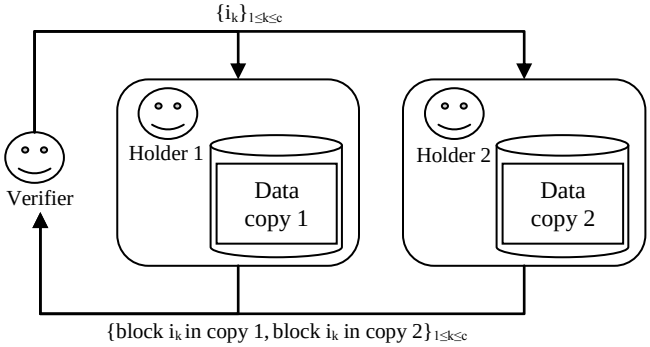

Fig. 7 Example of a storage where data is stored in two copies - Data replication-based verification: the verifier chooses a random set of block indexes (c indexes) that is sent to both holders as a challenge, then these latter send back the corresponding blocks, and finally the verifier compares the received responses together.

The verifier may not store the data; still it can exploit redundant data copies stored at multiple and distinct holders (as illustrated in Fig. 7). The scheme proposed in [27] for instance allows such verification. The scheme relies on algebraic signatures. The verifier requests algebraic signatures of data blocks stored at holders, and then compares the parity of these signatures with the signature of the parity blocks stored at holders too.

Table 2 Classification of remote data integrity verification protocols

Protocol type			Existing protocols
Tag-based	*Data dependant*	*Entire data*	Oualha [19], Deswarte *et al.* [8], Filho and Barreto [10], Oualha *et al.* [24]
		Data block	Sebé *et al.* [28], Chang and Xu [7], Oualha and Roudier [20], Ateniese *et al.* [2], Shacham and Waters [29], Zeng [32], Wagner [11], Erway *et al.* [9], Wang *et al.* [31]
	Data independent		Golle at al. [11], Juels and Kaliski [12]
Data replication-based			Lillibridge *et al.* [15], Caronni and Waldvogel [6], Bowers *et al.* [5], Schwarz and Miller [27]

The HAIL (High-Availability and Integrity Layer) protocol in [5] proposed also a data replication-based verification protocol. Similarly to [27], the verifier checks the correctness of a random subset of rows in the encoded matrix. Each server returns a linear combination of the blocks. To combine server responses, an aggregation code implemented with a Reed-Solomon code is used. The combined response is validated by first decoding and then checking that at least one of the responses of the secondary servers is valid.

The main downside of these approaches is that if the parity blocks does not match, it is difficult (depends on the number of the used parity blocks) and computationally expensive to recognize the faulty or dishonest holder.

4 Analysis and comparison

We have described a number of verification protocols after classifying then into several categories. With this description, we have identified the general features and the typical characteristics of the protocol categories. We will concentrate on these categories to go through their qualitative evaluation and comparison with respect to security, efficiency, and scalability considerations.

4.1 Security

Remote data is vulnerable to two classes of threats: accidental faults (e.g., caused by a bit error in the storage medium), and voluntary data damage due to holder selfishness whereby the entire or some parts of the data are removed in order to optimize holder's storage memory. In both cases, destruction or corruption of data stored at a holder should be detected as soon as possible. The main security problem is then the detection of such damage. We distinguish two main categories of verification schemes: probabilistic and deterministic protocols. The first type of protocols achieves only probabilistic detection of data damage that increases with the iteration of the protocol; whereas data damage detection is complete in the deterministic case.

Protocols with entire data dependant tags (e.g., [19], [8], [10], and [24]) provide generally deterministic guarantees for data damage detection; even though, the majority of verification protocols in the remaining categories probabilistically check data integrity for performance concerns.

With probabilistic verification, the assurance on data integrity preservation is increased with the iteration of the verification protocol. For one stateless protocol instance, [20] computed the probability of detection as:

$$Prob_{detection} = 1 - (1 - d)^c$$

where c is the number of blocks that are randomly checked and d the number of blocks destroyed by the holder. The authors showed that the probability of detection may approximate 100% for a small number of blocks being checked (about 10 blocks are checked for data probabilistic destruction < 0.5). The probability exponentially increases by increasing the number of aggregated blocks in a challenge message.

Besides a prototype implementation, [5] formally analyzed the security of their probabilistic protocol HAIL against an active and mobile adversary. This kind of adversary can corrupt multiple (but up to a certain number) storage hosts within any given time step. This adversary is capable of corrupting all storage hosts at different times. The security analysis demonstrated that it is possible to realize a high probability of detection, even with partial data verification.

Data dissemination into the system may expose the verification protocol to new attacks. Collusion attacks whereby multiple holders collude so that only one of them keeps a data copy that will be used to correctly answer all verifier's challenges directed toward these holders. Preventing this attack may consist on personalizing each copy to its holder, as explained in [6] and [25], or using an erasure coding mechanism for data replication. Even without a collusion attack, the verification protocols based on the replication technique are still vulnerable to replay attacks where one of the holders may derive or retransmit a response captured from another holder's response message. This attack can be hampered by using common authentication and encryption mechanisms during the verification process.

4.2 Efficiency

The costs of verifying the proper storage of some data should be considered for the two parties that take part in the verification process, namely the verifier and the holder.

Communication overhead. The size of challenge response messages must be optimized. Still, the fact that the proof of knowledge has to be significantly smaller than the data whose knowledge is proven should not significantly reduce the security of the proof.

Communication overhead is significantly low for the majority of the presented protocols, notably the more cryptographically advanced ones (using for instance elliptic curve cryptography or bilinear maps in respectively [24] and [32]), or replication based schemes (e.g., [5], [27]) since challenges and responses are aggregated between all replica holders.

The size of challenge and response messages is generally fixed for deterministic verification approaches. On the other hand, some probabilistic approaches, such as [20], have traded-off the size of these messages with the precision of the verification process: the larger the challenge-response messages are, the more accurate the verification should be.

Storage overhead. The verifier must store a meta-information that makes it possible to generate a time-variant challenge based on the proof of knowledge protocol mentioned above for the verification of the stored data. The size of this meta-information must be reduced as much as possible even though the data being verified is very large. The effectiveness of storage at the holder must also be optimized. The holder should store the minimum extra information along with the data.

Protocols that rely on data independent and reusable metadata or on the replication technique have proved that it is possible to reduce storage overhead at maximum; even though such techniques were less stringent with respect to other considerations (e.g., concerning data damage detection).

Additionally tags relating to the whole data are generally smaller than the sum of tags relating to single data blocks. Actually, the number of block tags is the point of tradeoff between storage overhead and computation complexity at the holder side.

CPU usage. Response generation and response verification respectively at the holder and at the verifier should not be computationally expensive. Generally probabilistic verification protocols that require from the holder to look up for a limited number of blocks (not the entire data) to compute a correct response are able to achieve good performance. This is the case with protocols using block-dependant tags; even though, this latter type of protocols requires more storage space to store such tags.

Computation complexity at the holder side is the main drain of protocols using entire data-dependant tags. These protocols are generally based on expensive functions that are applied to the whole data. To overcome this problem, improved versions (e.g., [28], [25]) have been proposed whereby the data is split into multiple blocks from which tags are produced. With this technique, computation complexity at the holder is reduced; on the other hand, the holder or the verifier should store more tags. Nevertheless, data damage detection continues to be deterministic in these protocols since computation operations over all blocks and tags are aggregated using random coefficients.

Table 3 Comparison of some examples of remote data integrity verification protocols in terms of performance. The variable *n* is denotes the number of blocks that structure the remote data. The variable *c* gives the portion of the data (in number of blocks) that is being checked.

Protocols / *Perf. parameters*		Tag-based			
		Data dependant			
		Entire data e.g., [24]	Data block e.g., [20]	Data independent e.g., [12]	Data replication-based, e.g., [27]
Comm. overhead		$O(1)$	$O(c)$	$O(1)$	$O(c)$
Storage over.	At verifier	$O(1)$	$O(1)$	$O(c)$	$O(1)$
	At holder	$O(1)$	$O(n)$	$O(c)$	$O(1)$
CPU usage	At verifier	$O(1)$	$O(c)$	$O(1)$	$O(c)$
	At holder	$O(n)$	$O(1)$	$O(1)$	$O(1)$

Table 3 summarizes the discussed performance effects of some of the described protocols from both the holder and the verifier sides. For instance, the table presents some tradeoff points, for example, in [24] where the computation complexity at the holder is extensive but can be counterweighed with probabilistic verification (using the variable *c*, rather than *n*) like in [20]. The table shows also that the storage overhead at the holder has been reduced at maximum in [27] compared to [20] and [12].

The performance analysis provided in [33] gives also the order of magnitude of storage and communication overhead for some of the verification protocols. The estimated orders of magnitude include a security parameter to stress on the security-performance tradeoff that these protocols should realize. Performance analysis based on real implementation, as in Bowers at al.'s proof of retrievability protocol [34], shows noticeable overheads in the time to access the file from the holder's disk. Probabilistic approaches are then more appropriate for large files, compared to deterministic protocols.

4.3 Scalability

The self-organizing style of the P2P storage system entails specific features of the verification protocol. The verification protocol should for instance scale to large populations of data owners. Verification information should be either self-carried by the data verifiers, or stored by the holders along with the data, or even made available in a public repository. The latter alternative is more robust since the information essential to the protocol realization is reliably stored in the system rather than kept by a single entity.

Verification protocols that rely on the replication technique (like [5] and [27]) do not need more storage space than it is required by the replication mechanism. Additionally, protocols with data independent tags (such as [11]) require information storable at a public space and reusable for any data. We argue that these two types of verification protocols are the most scalable and manageable for large population of peers.

Moreover, self-organization addresses highly dynamic environments like P2P networks in which peers frequently join and leave the system: this assumption implies that the owner is able to delegate data storage evaluation to verifiers. In this scenario, verifiers should act as third parties ensuring a periodic evaluation of holders after the owner leaves. The need for scalability also pleads for distributing this verification function, in particular to distribute verification costs among several entities. Last but not least, ensuring fault tolerance means preventing the system from presenting any single point of failure. To this end, data verification should be distributed to multiple peers as much as possible. The data should also be replicated to ensure its availability, which can only be maintained at a given level if it is possible to detect storage faults.

Protocols that allow delegating the verification task to other peers from the network can scale to a large and dynamic network. The protocol in [24] is the first to call for such property in order to deploy the storage application into the P2P context (as well [20] in the ad hoc context). The majority of the presented protocols may provide such delegability if the metadata needed for verification is either public or just individually secret (i.e., revealing the secret will only compromise the verification protocol taking place between a given verifier and a given holder). Some tag-based protocols (e.g., [8], [10], [24], [28]) employ tags derived from the data and some public information and rely on the random challenge to include the secret. Other tag-based protocols (e.g., [23], [2], [9]) construct tags produced from the data and a secret key but verifiable by the corresponding public key.

Since, it is difficult to fulfill all the desired requirements as explained in 2.2, such protocols realize tradeoffs among these requirements.

5 Extensions

A verification protocol allows checking remote data integrity. The protocol should be transportable to any storage application. For backup or archiving applications, data should survive during a long period of time, and therefore maintenance mechanisms are needed in order to maintain the replication rate at an optimal level. On the other hand, for distributed storage applications and file systems, data is generally very dynamic, thus it should be possible to perform dynamic operations on remote data in an efficient and secure way.

5.1 Storage maintenance

Data maintenance can be performed *proactively* whereby data is re-copied periodically in the network, as proposed by [14]. The degree of data redundancy in the network is frequently adjusted in function of locally estimated network dynamic properties. This technique provides only probabilistic guarantees for data availability. It may be sufficient if the degree of data redundancy is considerably high. Otherwise, deterministic guarantees should be provided by regularly checking storage at data holders. Furthermore, the technique of [14] takes into account peer dynamics (i.e., transient or permanent connection or failure), but not their potential misbehavior.

With *reactive* maintenance, data is re-copied if the corruption or destruction of at least one data copy is discovered. Data possession verification protocols allow the verifier to detect (either deterministically or probabilistically) whether the holder destroyed data. The detection of data destruction triggers regeneration of a new data copy at another holder in order to maintain a high (or at least a minimum) replication rate. [24] suggested that this task should not be solely tackled by the owner, since it does not often participate in the verification process. Therefore, the task should be conferred to the other participants in the verification process: verifiers and holders should then cooperate in order to regenerate a new data replica that will be stored at a volunteer peer. For this, the authors proposed to use random linear erasure coding to make possible the generation of new codes supplying the network with new blocks. They showed based on a theoretical model that with such maintenance mechanism data is made persistently available and reliable at a high replication rate in average.

5.2 Supporting dynamic data operations

The verification protocol should allow the owner to perform operations on the remote data while maintaining the same level of data correctness assurance. Several protocols supporting dynamic data operations have been proposed with a majority using data block tags. The authors of [1] have proposed to rely on pre-computed challenge-response couples that have been processed over multiple different blocks from the data. These couples are able to be updated if some blocks are modified, deleted, or inserted without the knowledge of the remaining blocks stored at the holder.

[9] and [31] suggested to rely on authenticating hash tables (e.g., Merkle hash tree, skip-list) to guarantee the update of the verification metadata without having to transmit the whole data to the owner.

A prominent advantage in using block-dependant tags is that it enables the protocol to support dynamic operations at block level. This has been demonstrated by [9] that proposed a rank-based skip list [26]. A skip-list is another authenticating hash table but it does not take into account the blocks' indexes; hence the notion of ranks introduced by Erway *et al.* The verifier keeps the label of the start node in the skip list. The bottom level nodes store block tags derived for a block d_i as:

$$T_i = g^{d_i}$$

When the owner updates its data that is stored at the holder, this latter computes a new root that can be validated using the old one and having the blocks that have been modified or inserted.

[31] proposed to use a Merkle Hash tree [16] taking data blocks as tree leaves and the metadata as the tree root. For verification, the authors have proposed a bilinear pairing-based verification where tags have this form:

$$T_i = (h(d_i).u^{d_i})^{\alpha}$$

where h is a hash function, u is a random element in the first group of the bilinear map's input, and α is the owner secret. The verifier makes a double check. It first verifies the position of a set of data blocks chosen randomly. It then checks the integrity of these blocks using a bilinear map function (cf. [18]) of the public key ${g_2}^{\alpha}$ (g_2 is the generator of the second group in the bilinear map's input). Since block indices are verified using the Merkle hash tree, they are not represented in the tags. Data update causes a modification of the root of the Merkle hash tree. The new root is computed by the holder and is verifiable having the old root and the updated blocks.

Both schemes use hash tables to generate proofs of data update at, however, block level. For example, to update a byte of a certain block, the whole block should be modified. Moreover, deleting a byte from a block changes the following blocks. [9] described a technique that allows supporting variable-sized blocks. The technique consists in changing the rank which is one of the attributes associated to the nodes forming the skip-list [26]. The rank of a node at the bottom level becomes equal to the size of its associated block. For the internal node rank, it becomes equal to the number of bytes reachable from each of them. Thus, data update can be performed at byte-level.

Dynamic data implies that the system requires a versioning control in order to synchronize data replicas at holders. [9] proposed another technique to extend their scheme with a versioning system based here again on the skip-list.

6 Conclusion

In this survey, we analyzed a large list of protocols for remote data integrity verification and compared them. From this list, we have identified the advantages and drawbacks of each of these protocols; hence, identifying the uses to which some of these protocols are more fitting than the others. For instance, probabilistic verification is the most retained technique if the holders store large amount of data, since the lookup of the entire data in their memory incurs more significant costs than fetching a small amount of data. In the case where the holders are very dynamic, replication-based verification becomes less appealing because of the difficulty to discover a large number of holders connected to the network during the same period of time.

The proposed verification protocols can be employed in several other distributed storage systems to control data integrity. In a cloud storage for instance, the client, by periodically checking the integrity of its remote data, may detect damage that is not necessarily due to the misconduct of the dedicated storage servers in the cloud, but because these latter may be compromised and then the storage system may be subject to attacks.

The majority of the existing verification protocols are built with data integrity verification as primary objective. Other security primitives like guarantying data confidentiality or owner privacy are generally neglected; though, such properties are of great interest in an open system like the P2P network. Moreover, these protocols consider a storage system with a single reader/writer. To support multiple readers and/or writers, a versioning system controlled by an access control protocol should be envisioned.

Designing remote data integrity verification is still a hot topic in the research community and further performance improvement and practical extensions (e.g., [30]) to suit a large spectrum of storage applications are underway.

References

[1] Ateniese G., Di Pietro R., Mancini, L. V., and Tsudik, G. 2008. Scalable and Efficient Provable Data Possession. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks (SecureComm'08)*, pp. 1–10, 2008.

[2] Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., and Song, D. 2007. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, ACM, 2007, 598-609.

[3] Bellare, M., Goldreich, O., and Goldwasser, S.. Incremental Cryptography and Application to Virus Protection. In Proceedings of the 27th annual ACM symposium on Theory of computing, p.45-56, May 29-June 01, 1995, Las Vegas, Nevada, United States.

[4] Blum, M., Evans, W. S., Gemmell, P., Kannan, S., and Naor, M. 1991. Checking the Correctness of Memories. In *32nd Annual Symposium on Foundations of Computer Science*, pages 90-99, San Juan, Puerto Rico, 1-4 October 1991.

[5] Bowers, K. D., Juels, and A., Oprea, A. 2009. HAIL: a high-availability and integrity layer for cloud storage. *16th ACM Conference on Computer and Communications Security CCS*, November 9 - 13, 2009.

[6] Caronni, G. and Waldvogel, M. 2003. Establishing Trust in Distributed Storage Providers. In *Proceedings of 3rd IEEE International Conference on P2P Computing*, pp. 128-133, Linkoping, Sweden, September 2003.

[7] Chang, E. C. and Xu, J. 2008. Remote Integrity Check with Dishonest Storage Server. *13th European Symposium on Research in Computer Security (ESORICS)*, pp.223-237, 2008.

[8] Deswarte, Y., Quisquater, J.-J., and Saïdane, A.. Remote Integrity Checking. In *Proceedings of 6th Working Conference on Integrity and Internal Control in Information Systems (IICIS)*, 2004.

[9] Erway, C., Kupcu, A., Papamanthou, C., and Tamassia, R. 2008. In Proceedings of the ACM International Conference on Computer and Communications Security (CCS), pages 213-222, Chicago IL, USA, 2009.

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

[11] Golle, P., Jarecki, S., and Mironov, I. 2002. Cryptographic Primitives Enforcing Communication and Storage Complexity. In *Proceeding of Financial Cryptography*, pages: 120-135, 2002.

[12] Juels, A., and Kaliski, B. S. 2007. PORs: Proofs of retrievability for large files. Cryptology ePrint archive, June 2007. Report 2007/243.

[13] Koblitz, N. 1987. Elliptic curve cryptosystems. *Mathematics of Computation*, Volume 48, pages: 203-209, 1987.

[14] Leng, C., Terpstra, W. W., Kemme, B., Stannat, W. and Buchmann, A. P.. Maintaining replicas in unstructured P2P systems. CoNEXT, page 19. ACM, 2008.

[15] Lillibridge, M., Elnikety, S., Birrell, A., Burrows, M., and Isard, M. 2003. A Cooperative Internet Backup Scheme. *In Proceedings of the 2003 Usenix Annual Technical Conference* (General Track), pp. 29-41, San Antonio, Texas, June 2003.

[16] Merkle, R. C. 1987. A Digital Signature Based on a Conventional Encryption Function. *In CRYPTO '87, Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology* (1988), pp. 369-378.

[17] Naor, M., and Rothblum, G. N. 2005. The Complexity of Online Memory Checking. In *Proceeding of 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005)*, pp. 573-584.

[18] Okamoto, T. 2006. Cryptography based on bilinear maps. *The 16th International Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes* - AAECC-16, Lecture Notes in Computer Science Vol.3857, pp.35-50, Springer-Verlag, 2006.

[19] Oualha, N. 2009. Security and cooperation for peer-to-peer data storage. PhD Thesis, EURECOM/Telecom ParisTech, June, 2009.

[20] Oualha, N. and Roudier, Y. 2007. Securing ad hoc storage through probabilistic cooperation assessment. *3rd Workshop on Cryptography for Ad hoc Networks*, July 8th, 2007, Wroclaw, Poland. *Electronic Notes in theoretical computer science*, Volume 192, N°2, May 26, 2008, pp 17-29.

[21] Oualha, N. and Roudier, Y. 2008. Reputation and Audits for Self-Organizing Storage. In *Proceedings of the 1st Workshop on Security in Opportunistic and SOCial Networks (SOSOC 2008)*, Istanbul, Turkey, September 2008.

[22] Oualha, N. and Roudier, Y. 2010. Securing P2P Storage with a self-organizing Payment Scheme. *3rd International Workshop on Autonomous and Spontaneous Security (SETOP 2010)*, September 23, 2010, Athens, Greece.

[23] Oualha, N., Önen, M., and Roudier, Y. 2008. A Security Protocol for Self-Organizing Data Storage. *23rd International Information Security Conference (IFIP SEC 2008)*, Milan, Italy, pp. 675-679, September 2008.

[24] Oualha, N., Önen, M., and Roudier, Y. 2010. Secure P2P Data Storage and Maintenance. *Hindawi International Journal of Digital Multimedia Broadcasting*, vol. 2010, Article ID 720251, 2010.

[25] Oualha, N., Önen, M., and Roudier, Y., 2008. A security protocol for self-organizing data storage. EURECOM Research Report RR-08-208 (extended version), 2008.

[26] Pugh, W. 1989. Skip Lists: A Probabilistic Alternative to Balanced Trees. *In Workshop on Algorithms and Data Structures* (1989), pp. 437-449.

[27] Schwarz, T., and Miller, E. L. 2006. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *Proceedings of the IEEE Int'l Conference on Distributed Computing Systems (ICDCS '06)*, July 2006.

[28] Sebé, F., Domingo-Ferrer, J., Martínez-Ballesté, A., Deswarte, Y., and Quisquater, J.-J. 2007. Efficient Remote Data Possession Checking in Critical Information Infrastructures. *IEEE Transactions on Knowledge and Data Engineering*. Vol. 20, pp. 1034-1038. Aug 2008. ISSN: 1041-4347

[29] Shacham, H. and Waters, B. 2008. Compact Proofs of Retrievability. In *Proceedings of Asiacrypt 2008*, Lecture Notes in Computer Science, Vol. 5350, pp. 90-107, Springer-Verlag, 2008.

[30] Wang, C., Wang, Q., Ren, K., and Lou, W.. Privacy-preserving public auditing for data storage security in cloud computing. In *Proceedings of the 29th conference on Information communications*, p.525-533, March 14-19, 2010, San Diego, California, USA.

[31] Wang, Q., Wang, C., Li, J., Ren, K., and Lou, W. 2009. Enabling public verifiability and data dynamics for storage security in cloud computing. *14th European Symposium on Research in Computer Security (ESORICS 2009)*, Saint Malo, France, pp. 355–70, September 21-25, 2009.

[32] Zeng, K. 2008. Publicly Verifiable Remote Data Integrity. *10th International Conference on Information and Communications Security (ICICS 2008)*, pp. 419-434, 20 - 22 October, 2008.

[33] Dodis Y., Vadhan S., and Wichs D. 2009. Proofs of Retrievability via Hardness Amplification. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography (TCC '09)*, Omer Reingold (Ed.). Springer-Verlag, Berlin, Heidelberg, 109-127.

[34] Bowers K. D., Juels A., and Oprea A. 2009. Proofs of retrievability: theory and implementation. In *Proceedings of the 2009 ACM workshop on Cloud computing security (CCSW '09)*. ACM, New York, NY, USA, 43-54.