Institut Eurecom
Department
2229, route des Crêtes
B.P. 193
06904 Sophia Antipolis
FRANCE

Research Report RR-07-201

Verifying Self-Organized Storage with Bilinear Pairings

June 15th, 2007

Nouha Oualha, Suna Melek Önen, Yves Roudier[1]

Tel: (+33) 4 93 00 81 00
Fax: (+33) 4 93 00 82 00
Email: {oualha, onen, roudier}@eurecom.fr

[1] Institut Eurecom research is partially supported by its industrial members: BMW Group Research & Technology – BMW Group Company, Bouygues Telecom, Cisco Systems, France Telecom , Hitachi, SFR, Sharp, STMicroelectronics, Swisscom, Thales

Verifying Self-Organized Storage with Bilinear Pairings

Nouha Oualha, Suna Melek Önen, Yves Roudier[2]

Abstract

This paper describes a cryptographic protocol for securing self-organized data storage through periodic verifications. Such verifications are beyond simple integrity checks since a storage node generates a proof that it still keeps the data that were uploaded to it. The proposed verification protocol is efficient, deterministic, and scalable and successfully prevents most of the security threats to self-organizing storage verification. In particular, a data owner can prevent data destruction at a specific holder by storing personalized replicas crafted thanks to the use of bilinear pairings. Furthermore, the protocol also makes it possible for the data owner to delegate the verification operation to other nodes without revealing any secret information.

Keywords

Secure self-organized storage, proof of knowledge, bilinear pairings, verification delegation, selfishness.

[2] This work was also supported by the GET (Groupement des Ecoles de Télécoms) Initiative research program on autonomic and spontaneous networks, the ACI SI MOSAIC project, and the PACALAB research project.

Verifying Self-Organized Storage with Bilinear Pairings

Nouha Oualha, Suna Melek Önen, Yves Roudier

1 INTRODUCTION

Self-organized data storage is becoming an increasingly important application in distributed systems, especially with the development of the peer-to-peer paradigm. It however proves far more demanding in terms of security than classical distributed or remote storage approaches. In particular, no public key infrastructure can be assumed to be available in such a setting, and the nodes participating to a storage application are constantly joining and leaving on a large scale. P2P file sharing has also brought to light the novel issue of free-riders, or selfish nodes. Selfishness represents an entire new class of attacks whereby nodes try to optimize their resource consumption at the expense of other nodes. Attempts at securing file sharing however have essentially focused on eliciting a fair distribution of upload and download contributions of nodes.

Self-organized data storage goes one step further in trying to ensure data availability on a long term basis. This objective requires developing appropriate primitives, that is, storage verification protocols, for detecting dishonest nodes free riding on the self-organized storage infrastructure. Contrary to simple integrity checks, which make sense on a potentially defective yet trusted server, self-organized storage requires defining efficient primitives for detecting voluntary data destruction by a remote node: in particular verifying the presence of these data remotely should not require transferring them back in their entirety.

This paper presents such a verification protocol exhibiting a low resource overhead. This protocol was designed with scalability as an essential objective: it enables generating an unlimited number of verification challenges from the same security metadata; it is also especially original in allowing third parties to be delegated verification operations, thanks to the bilinear pairing based proof that constitutes its core. The latter feature proves specifically interesting to ensure the scalability of storage systems built on top of it since verification, and not only storage, can be distributed in a self-organized manner.

This paper is structured as follows: Section 2 introduces the problem statement by describing the architectural and security requirements of self-organized storage that the verification protocol must meet. Section 3 describes the bilinear pairing based solution that we propose to implement an adequate verification protocol. Section 4 finally analyzes the security and performance of the solution outlined and compares it with related work.

2 PROBLEM STATEMENT

This section briefly describes the architectural requirements for storage verification and how the protocol operates. It then goes on to describe the security threats that have to be addressed by the verification protocol. We consider a cooperative storage application in which a node, called the data owner, replicates its data by storing them at several nodes, called data holders. The latter nodes agree to keep data for a predefined period of time negotiated with the former one. In an open and self-organized environment like peer-to-peer or ad hoc networks, the correct operation of the storage application depends on the cooperation of user nodes: in particular, holders that have promised to keep data for owners must fulfill their pledge.

Such a behavior might be evaluated through the adoption of a routine check through which the holder should be periodically prompted to respond to a time-variant challenge as a proof that it holds its promise. That interactive check may be formulated as a proof of knowledge [1] in which the holder attempts to convince a verifier that it possesses some data: this is demonstrated by correctly responding to queries that requires the knowledge of the very data. However, the data corresponding to the secret in a traditional proof of knowledge protocol is very large while the proof must be much significantly smaller in order to achieve a reasonable performance. The proof in the verification protocol must therefore, at least, not entirely reveal the data itself. This paper exhibits a solution answering such requirements.

2.1 Architectural Requirements

Data storage involves highly dynamic scenarios, as illustrated in [2] for instance: these often require the holder to delegate data storage evaluation to third parties because the owner might not remain in touch with holders (see Figure 1). Other rationales for distributing the verification function include scalability, which in particular require balancing verification costs among several entities, as well as fault tolerance, notably preventing the system from presenting any single point of failure.

Data Replication is another element that ensures a high availability in dynamic self-organized storage system. However, the level of data replication, hence its availability, can only be maintained if it is possible to detect storage defection, which can be done through periodic verifications, provided they are not too expensive to perform.

We describe the actors of the system using the following notations:

- the owner is denoted by O stores data at different holders;
- holders: the set of m holders of data X is denoted $\{H_i\}_{1 \le i \le m}$, ;

- verifiers: each holder H_i is monitored by a set of n verifiers denoted $\{V_{i,j}\}_{1\leq j\leq n}$ (the owner may participate in the verification process).

We assume that all information related to storage or its verification, such as the storage duration agreed, the number of verifiers, and the frequency of their verifications, etc. are negotiated in a preliminary phase between the owner and each holder.

While cooperation incentives are not inherent to the verification protocol, they are useful for achieving better storage results. Such mechanisms involve the use of either reputation or remuneration. The investigation of such mechanisms is however far beyond the scope of this paper.

The verification must be efficient in terms of resource usage, and most importantly secure. The following section examines security threats that must be addressed to satisfy the latter requirement.

2.2 Security Threats

The verification protocol must address or at least mitigate the following threats:

- **Data destruction.** The destruction of data stored at a holder must be detected as soon as possible. It may be due to:
    - *Data corruption*: data may be altered by a faulty or malicious data holder;
    - *Holder selfishness*: the holder may destroy data in order to optimize its storage resources thereby taking unfair advantage of the storage application;
    - *Colluding holders*: one holder may store a unique instance of the data for several other holders of the same data replica, thereby defeating the purpose of replication to their sole profit, i.e., reducing data storage footprint.
- **Denial-of-Service (DoS).** DoS attacks aim at indirectly disrupting the storage application through attacks against the storage verification:
    - *Flooding attack*: the holder may be flooded by verification requests from malicious verifiers, or from attackers that have not been selected as verifiers by the owner. Verifiers may also be subject to a similar attack.
    - *Replay attack*: a valid challenge or response message is maliciously or fraudulently repeated so as to disrupt the verification.
    - *Collusion between a holder and its verifier*: a verifier that is in collusion with the holder claims positive verification results in order to boost the trust that the owner puts into this holder
    - *Verifier selfishness*: a verifier claims a random verification result without contacting the holder in order to optimize its resource usage by not sending messages.

The following section introduces a verification protocol that addresses the security threats and architecture requirements presented above. This protocol in particular introduces the delegation capability through the use of bilinear pairing cryptography.

3 A NEW VERIFICATION PROTOCOL

This section first introduces the security primitives that are used in our protocol. It then completely describes the verification protocol for proving data possession.

3.1 Bilinear Maps

Let G_1 be a cyclic additive group generated by P, whose order is a prime q, and G_2 be a cyclic multiplicative group with the same order q. Let e: $G_1\times G_1 \rightarrow G_2$ be a map with the following properties:

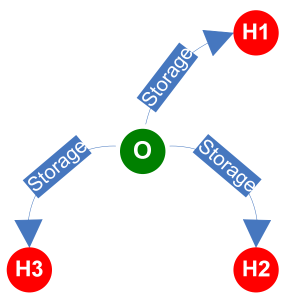
- **Bilinearity:** $e(aP, bQ) = e(P, Q)^{ab}$ for all $P, Q, R \in G_1$ and $a, b \in \mathbb{Z}_q$;
- **Non-degeneracy:** there exists $P, Q \in G_1$ such that $e(P, Q) \neq 1$, in other words, the map does not send all pairs in $G_1\times G_1$ to the identity in G_2;
- **Computability:** there is an efficient algorithm to compute $e(P, Q)$ for all $P, Q \in G_1$.

Such maps are considered as admissible bilinear maps.

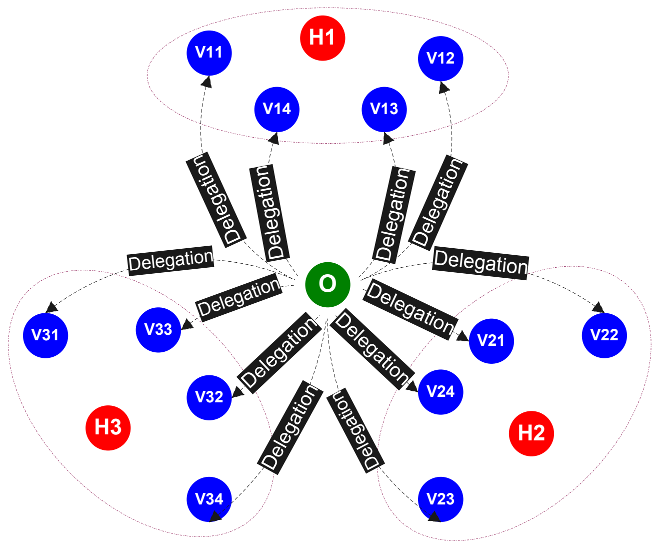
3.2 The Elliptic Curve Discrete Logarithm Problem

The security of our protocol is based on the following problem that is defined as the Elliptic Curve Discrete Logarithm problem and denoted by DLP.

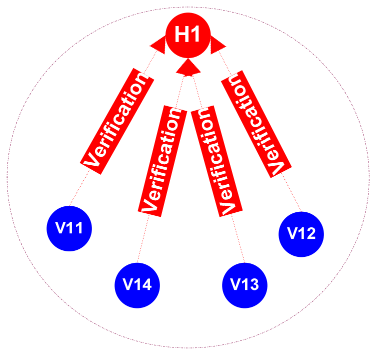
Consider G_1 an additive cyclic group of prime order q and P a generator of G_1. The *Discrete Logarithm Problem (DLP)* in G_1 consists in finding an integer $n \in \mathbb{Z}_q^*$, such that, given two group elements P and Q, $Q = nP$ whenever such an integer exists.



(a) Storage



(b) Delegation

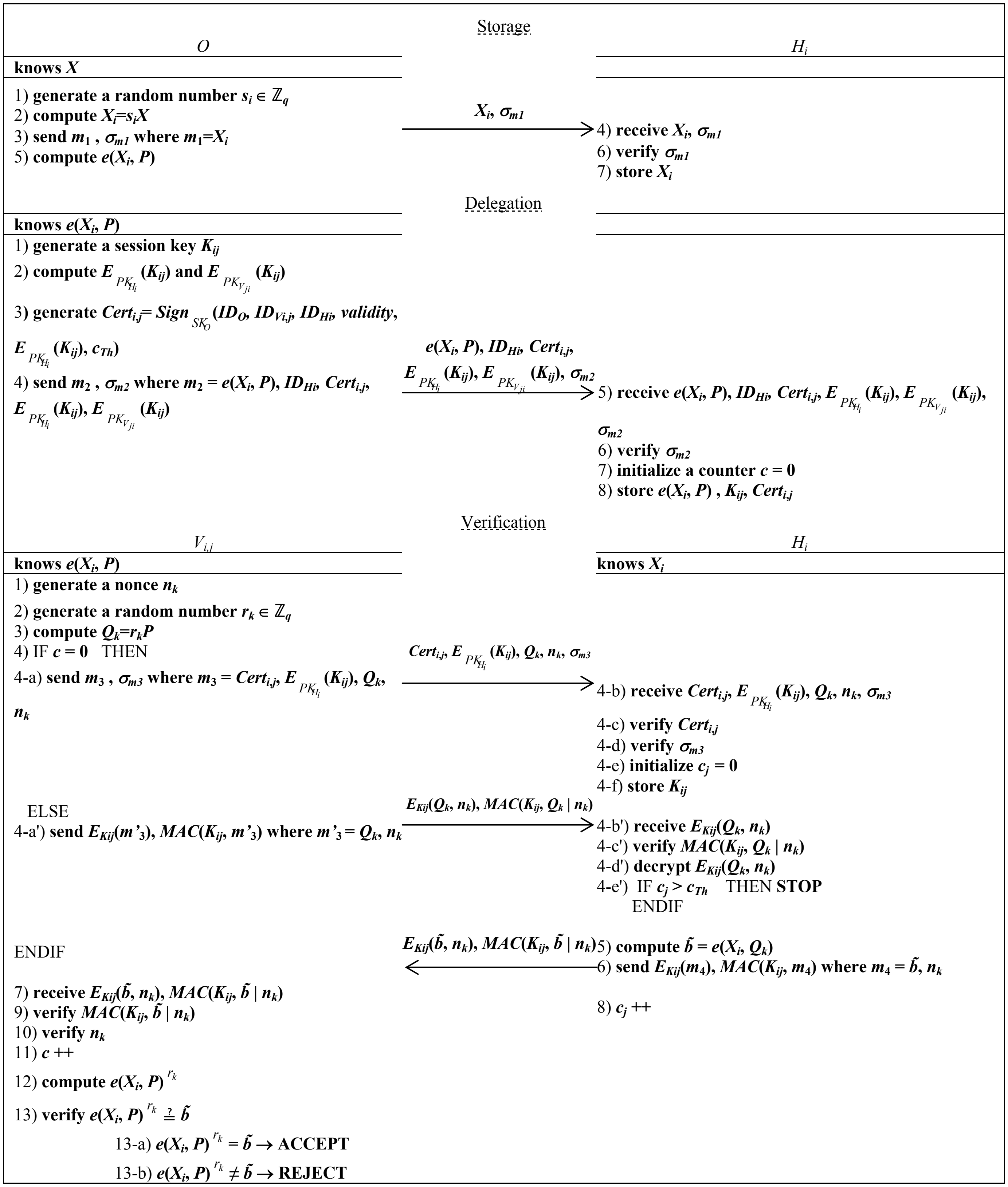


(c) Verification

Figure 1. System Architecture: (a) the owner O stores data at holders H1, H2, and H3; (b) O delegates the verification of data to 4 verifiers per holder; (c) the verifiers assigned to H1 periodically check storage at H1.

3.3 Verification Protocol

The verification protocol consists of three phases: storage, delegation, and verification. In the storage phase, the owner sends each holder a replica that is personalized to prevent potential collusions between holders. In the delegation phase, the owner generates credentials for a set of chosen verifiers that are thereby authorized to proceed to the verification phase. Thanks to the use of a bilinear scheme, the owner does not need to share any secret with the verifiers, which therefore do not need to be particularly trusted. The verification phase is the most important part of the protocol since it corresponds to the computation of the data possession proof by the holder.



Storage

O

knows X

1) **generate a random number $s_i \in \mathbb{Z}_q$**
2) **compute $X_i = s_i X$**
3) **send m_1 , σ_{m1} where $m_1 = X_i$**
5) **compute $e(X_i, P)$**

$\xrightarrow{X_i,\ \sigma_{m1}}$

H_i

4) **receive X_i, σ_{m1}**
6) **verify σ_{m1}**
7) **store X_i**

Delegation

knows $e(X_i, P)$

1) **generate a session key K_{ij}**
2) **compute $E_{PK_{H_i}}(K_{ij})$ and $E_{PK_{V_{ji}}}(K_{ij})$**
3) **generate $Cert_{i,j} = Sign_{SK_O}(ID_O, ID_{Vi,j}, ID_{Hi}, validity, E_{PK_{H_i}}(K_{ij}), c_{Th})$**
4) **send m_2 , σ_{m2} where $m_2 = e(X_i, P), ID_{Hi}, Cert_{i,j}, E_{PK_{H_i}}(K_{ij}), E_{PK_{V_{ji}}}(K_{ij})$**

$\xrightarrow{e(X_i, P),\ ID_{Hi},\ Cert_{i,j},\ E_{PK_{H_i}}(K_{ij}),\ E_{PK_{V_{ji}}}(K_{ij}),\ \sigma_{m2}}$

5) **receive $e(X_i, P), ID_{Hi}, Cert_{i,j}, E_{PK_{H_i}}(K_{ij}), E_{PK_{V_{ji}}}(K_{ij}), \sigma_{m2}$**
6) **verify σ_{m2}**
7) **initialize a counter $c = 0$**
8) **store $e(X_i, P)$, K_{ij}, $Cert_{i,j}$**

Verification

$V_{i,j}$

knows $e(X_i, P)$

1) **generate a nonce n_k**
2) **generate a random number $r_k \in \mathbb{Z}_q$**
3) **compute $Q_k = r_k P$**
4) IF **$c = 0$** THEN
4-a) **send m_3 , σ_{m3} where $m_3 = Cert_{i,j}, E_{PK_{H_i}}(K_{ij}), Q_k, n_k$**

$\xrightarrow{Cert_{i,j},\ E_{PK_{H_i}}(K_{ij}),\ Q_k,\ n_k,\ \sigma_{m3}}$

H_i

knows X_i

4-b) **receive $Cert_{i,j}, E_{PK_{H_i}}(K_{ij}), Q_k, n_k, \sigma_{m3}$**
4-c) **verify $Cert_{i,j}$**
4-d) **verify σ_{m3}**
4-e) **initialize $c_j = 0$**
4-f) **store K_{ij}**

ELSE
4-a') **send $E_{Kij}(m'_3)$, $MAC(K_{ij}, m'_3)$ where $m'_3 = Q_k, n_k$**

$\xrightarrow{E_{Kij}(Q_k, n_k),\ MAC(K_{ij}, Q_k \mid n_k)}$

4-b') **receive $E_{Kij}(Q_k, n_k)$**
4-c') **verify $MAC(K_{ij}, Q_k \mid n_k)$**
4-d') **decrypt $E_{Kij}(Q_k, n_k)$**
4-e') IF $c_j > c_{Th}$ THEN **STOP**
ENDIF

ENDIF

5) **compute $\tilde{b} = e(X_i, Q_k)$**
6) **send $E_{Kij}(m_4)$, $MAC(K_{ij}, m_4)$ where $m_4 = \tilde{b}, n_k$**

$\xleftarrow{E_{Kij}(\tilde{b}, n_k),\ MAC(K_{ij}, \tilde{b} \mid n_k)}$

8) c_j ++

7) **receive $E_{Kij}(\tilde{b}, n_k)$, $MAC(K_{ij}, \tilde{b} \mid n_k)$**
9) **verify $MAC(K_{ij}, \tilde{b} \mid n_k)$**
10) **verify n_k**
11) **c ++**
12) **compute $e(X_i, P)^{r_k}$**
13) **verify $e(X_i, P)^{r_k} \overset{?}{=} \tilde{b}$**
    13-a) **$e(X_i, P)^{r_k} = \tilde{b} \rightarrow$ ACCEPT**
    13-b) **$e(X_i, P)^{r_k} \neq \tilde{b} \rightarrow$ REJECT**

Figure 2. The bilinear pairing based verification protocol

We also assume the existence of a preliminary discovery phase of the public keys of participant nodes, which are then used for signature generation. Every node N participating to the cooperative storage application possesses a pair of public and private signature keys designated by $\{PK_N, SK_N\}$. We furthermore assume that each node N is uniquely identified by ID_N (for instance, ID_N might simply be PK_N). We denote by $Sign_{SK}(m)$ the signature of a message m computed with secret key SK: we also denote this signature in an abbreviated form as σ_m. Finally, $E_K(m)$ represents the encryption of m using key K. Based on these notations, the detailed protocol phases are as follows (see also **Figure 2**):

- **Storage phase:** Let X denote the data that the owner O wishes to store at holder H_i. In order to prevent a collusion between the holders, O generates a random number $s_i \in \mathbb{Z}_q$ that is kept secret and that is used to personalize the replica stored at H_i. O computes such a personalized replica in the form of $X_i=s_iX$. Finally, O sends X_i to H_i for storage.
- **Delegation phase:** O generates a credential $Cert_{i,j}$ to assert that it delegates the verification of X to a verifier $V_{i,j}$, that it selected. O also generates a session key K_{ij} intended to secure the communication that will be held between a verifier $V_{i,j}$ and its assigned holder H_i. This key is encrypted for $V_{i,j}$ and H_i. O needs to send $V_{i,j}$ adequate metadata for performing subsequent verifications: these consist of the encrypted challenge $e(X_i,P)$, which the verifier can use to generate time-variant challenges on H_i's replica of X, together with the credential and the session key cryptographic envelopes.
- **Verification phase:** This phase is carried out between a holder H_i and one of its appointed verifiers $V_{i,j}$. The verifier first challenges the holder by sending $Q_k=r_kP \in G_1$ which is generated using a random number $r_k \in \mathbb{Z}_q$. In order to prevent a denial of service through the replay of the challenge message, $V_{i,j}$ sends a nonce n_k with every such message. If it is the first time that $V_{i,j}$ checks the data stored at H_i, $V_{i,j}$ sends H_i a signed challenge message including the credential $Cert_{ij}$ concatenated to Q_k, n_k, and the encrypted session key $E_{PK_{H_i}}(K_{ij})$. Upon reception of this message (see 4-b in **Figure 2**), H_i initializes a new quota counter c_j to zero (a similar counter is also kept at the verifier). Imposing a quota on the number of verifications allowed prevents a malicious verifier from flooding the holder with challenge messages. H_i also stores the session key K_{ij} that will be used as a key for ensuring entity authentication in subsequent verifications (see 4-b' in **Figure 2**): as such, it replaces the public key signature. The session also makes it possible for $V_{i,j}$ not to send the credential again. In the case where c_j is less than the quota of verification operations allowed c_{Th}, H_i computes $\tilde{b} = e(X_i, Q_k)$, which it sends $V_{i,j}$ together with n_k, the whole being encrypted with K_{ij}. $V_{i,j}$ compares $\tilde{b}$ to $e(X_i, P)^{r_k}$: if $\tilde{b} = e(X_i, P)^{r_k}$, then thanks to the bilinearity property of map e, $V_{i,j}$ is sure that X_i is still stored at H_i. Indeed, $e(X_i, P)^{r_k} = e(X_i,r_kP) =e(X_i,Q_k)$. Whenever the quota counter c_j (and c) exceeds threshold c_{Th}, the verifier is not allowed to challenge the holder anymore during the current timeframe.

4 PROTOCOL EVALUATION

This section assesses the proposed verification protocol from security and performance perspectives.

4.1 Security Analysis

This section first proves that the proposed verification protocol is a proof of knowledge and then discusses in which respect it addresses the security threats described in Section 2.2.

In order to validate the correctness of the proposed protocol, we analyze its completeness and soundness that are the two essential properties of a proof of knowledge protocol [1]: a protocol is *complete* if, given an honest claimant and an honest verifier, the protocol succeeds with overwhelming probability, i.e., the verifier accepts the claimant's proof; a protocol is *sound* if, given a dishonest claimant, the protocol fails, i.e., the claimant's proof is rejected by the verifier, except with a small probability.

Theorem 1- *The proposed protocol is* ***complete*** *if the verifier and the holder correctly follow the proposed protocol, the verifier always accepts the proof as valid.*

Proof: Thanks to the bilinearity property of map e, we have:

$$e(X_i\,, r_kP) = e(X_i\,,P)^{r_k}$$

Theorem 2- *The proposed protocol is* ***sound*** *if the claimant does not store the data X_i , then the verifier will not accept the proof as valid.*

Proof: If H_i does not keep X_i , it can only generate a correct response to a challenge r_kP, by retrieving r_k and computing $e(X_i,P)^{r_k}$. However finding r_k given r_kP is equivalent to solving the DLP and is thus considered as NP-hard. This proves the soundness of the proposed protocol.

Thanks to the correctness of the protocol, if a holder does not provide correct responses to the challenges generated by a certain verifier, the verifier will immediately detect that some data has been corrupted or destroyed at this specific holder.

Regarding threat coverage, collusion between holders may still imply data destruction in some of them. However, such attacks are prevented in the proposed protocol since each holder stores a personalized replica and this cannot collude with another holder in order to correctly finalize the verification phase. This shows that the proposed protocol globally prevents data destruction.

As shown in the previous section, at each step of the protocol, messages are authenticated with common signature algorithms such as RSA. Thanks to this preliminary authentication phase, a holder cannot store bogus data. Since the stored data are assumed to be dense, the cost of storage is assumed to be much more expensive than the cost of verifying a digital signature. Thus, this additional mechanism inherently mitigates external denial-of-service (DoS) attacks whereby intruders perform flooding attacks against holders.

In addition to external DoS attacks, authorized verifiers might also perform flooding attacks against holders. In this particular case the computational and communication overhead are reduced thanks to the use of MAC; however, authentication is enough to prevent such attacks since verifiers are authorized to participate in the communication. We thus first propose to limit the number of verifiers that can send request to a given holder. This number can be predefined in the storage phase between the owner and the holder by considering the capacity of the holder. We also propose to define a threshold value for requests originating from a verifier. Thanks to these mechanisms, DoS attacks originating from authorized verifiers are mitigated.

In addition to flooding attacks originating from either intruders or authorized verifiers, simple replay attacks can also constitute a serious threat against the proposed framework and should thus be taken into consideration. In the proposed protocol, replay attacks are prevented thanks to the use of nonces in the verification phase. Such attacks can also be prevented by a preliminary protocol based on the exchange of cookies. However, such solutions may not be fully adapted for wireless communications.

Another potential attack that does not compromise the storage verification, but rather the locality of storage, is the proxying attack. In this attack, a holder can pretend to be storing data while in fact proxying requests and responses in front of another data holder. the random-read protocol presented in [3] can provide a solution to this attack: both parties establish a data verification session using a commitment protocol. Distance-bounding protocols [4] may provide an alternative solution for ad hoc or sensor networks but should be integrated into the verification process to detect proxying attempts. The latter solution also assumes that it is possible to estimate the distance of the queried node.

The owner automatically detects selfish verifiers thanks to the absence of a valid signature originating from a holder. However, the collusion of a verifier with one of the holders it is assigned is harder to prevent since the verifier may provide the holder with a correct verification result that the holder can sign without storing the data. Such attacks can be mitigated with quorum or vote based approaches, whereby several verifiers are requested to assess the behavior of the same holder.

To summarize, while the proposed protocol fully prevents data destruction attacks thanks to its correctness and thanks to the use of personalized replicas, DoS attacks can only be mitigated, thanks to the introduction of an additional authentication phase and to the definition of bounds on the number of verifiers and on the number of verification requests. This mitigation should however cover most situations arising in the verification process.

4.2 Performance Analysis

Signature algorithms used for node authentication may be RSA or ECDSA [5]. [6] and [7] provide some elements of comparison: for example, a 163-bit ECDSA offers the same level of protection as 1,024-bit modulus RSA with a significant reduction of computational overhead; both produce 320-bits signatures. Moreover, the protocol limits the penalty performance of such signatures to only once per established session.

The first two phases being performed only occasionally compared with the challenge-response, the protocol performance however essentially depends on the verification phase. In the proposed protocol, challenges with bilinear properties are computed using elliptic curve cryptography. The verification protocol requires the verifier to store only an elliptic pairing-based proof of knowledge. This proof allows producing on demand challenges for the verification. Finally, the verification of a response message relies on one exponentiation over the proof, the exponent being a random number chosen by the verifier. Compared with solutions that use the RSA encryption ([8] and [9], see next Section), our solution in intrinsically less expensive. The exponentiation operation used in the RSA solution makes use of the whole data as an exponent; in our solution the exponent is a much smaller random number.

4.3 Related Work

Efforts for storage verification mainly focus on two types of applications: backup and file system. For the former, the data to be verified is still kept by the owner that plays as well the verifier role. For the latter, data is not stored anymore at the owner. Additionally verification approaches that allow checking the presence of the whole data should be distinguished from approaches that rely on probabilistic checking (portion of data).

In the cooperative Internet backup scheme described in [3], each peer periodically challenges its partners by requesting a block out of the stored data. The response is checked by comparing it with the valid block stored at the verifier's disk space. A similar approach is proposed in [10] where the prover has to send the MAC of data as the response to the challenge message. The data originator sends a fresh nonce (a unique and randomly chosen value) as the key for the message authentication code: this is to prevent the holder node from storing only the hash value of the data. Another probabilistic verification approach based on the algebraic signatures of data blocks was proposed in [11]: it relies on the homomorphic properties of such signatures with respect to the parity. Compared to [3] and [10], the verifier does not need to store any information for verification. However, if the parity blocks do not match, it is difficult (depending on the number of parity blocks used) and computationally expensive to recognize the faulty holder. In contrast, our solution ensures the identification of misbehaving holders.

[8] describes two different approaches: the first one requires pre-computed results of challenges to be stored at the verifier, where a challenge corresponds to requesting a fingerprint of the data concatenated with a random number. Compared with [3], [10], and [11], the protocol requires less verification metadata storage at the verifier, yet it allows only a fixed number of challenges to be performed. The second solution relies on an RSA-based proof: the public key is the data itself, whereas the homomorphic properties inherent to RSA make it possible to use the inverse of this data as the secret key. This solution does not require the verifier to store the whole data. It also makes it possible to generate an unlimited number of challenges, similarly to our protocol. A similar RSA-based solution is described by Filho and Barreto in [9]. However,. Contrary to these solutions, our protocol provides a secure delegation primitive that does not necessitate revealing the owner's secret key to the verifier.

The main characteristics of the existing verification protocols seen in this section are summarized in Table 1.

Table 1: Comparison of existing verification protocols

Verification protocols	**Resource usage**			**Security**
	Verification metadata	Communication overhead (response message)	Computation overhead at verifier	
Lillibridge et al. [3]	data	data block	linear comparison with original data	probabilistic
Caronni and Waldvogel [10]	data	hash of data	hash computation	deterministic
Deswarte et al. [8]: hash solution	pre-computed challenges	hash of data	linear comparison with hash of data	deterministic
Deswarte et al. [8]: RSA solution	data hash	hash of data	RSA exponentiation with data as the exponent	deterministic
Filho and Barreto [9]	data hash	hash of data	RSA exponentiation with data as the exponent	deterministic
Schwarz and Miller [11]	nothing	block signatures + parity	signature computation	probabilistic
Our protocol	Elliptic curve point (G_2 element)	Elliptic curve point	bilinear pairing exponentiation with a small random number as the exponent	deterministic

5 CONCLUSION

We proposed in this paper a verification protocol that allows checking the integrity and availability of storage in a cooperative storage application. We showed that our protocol satisfies high security requirements with low resource consumption compared with existing verification protocols. This verification exploits the bilinearity properties available in some elliptic curve cryptography pairings to enable an efficient storage verification that additionally can be delegated to third parties.

Assessing the actual state of storage in such an application represents the first step towards efficient reaction to misbehavior: active replication strategies can be built based on such evaluations; we are also actively working on the construction of cooperation incentives using this protocol as an observation primitive.

6 REFERENCES

[1] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, “Handbook of Applied Cryptography”, CRC Press, 1996.

[2] N. Oualha, Y. Roudier, "Securing Ad Hoc Storage through Probabilistic Cooperation Assessment", In Proceedings of the 3rd Workshop on Cryptography for Ad Hoc Networks, Wroclaw, Poland, 2007.

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

[4] Stefan Brands and David Chaum, “Distance-Bounding Protocols”, In *Workshop on the theory and application of cryptographic techniques on Advances in cryptology — CRYPTO 1994*, volume 839 of Lecture Notes in Computer Science, pages 344–359. Springer-Verlag, August 1994.

[5] D.J. Johnson, A.J. Menezes, S.A. Vanstone, “The Elliptic Curve Digital Signature Algorithm (ECDSA)”, *International Journal of Information Security*, Vol. 1, pp. 36-63, 2001

[6] N. R. Potlapally, S. Ravi, A. Raghunathan and N. K. Jha, “A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols”, In *IEEE Transactions in Mobile Computing*, vol. 5, no. 2, pp. 128-143, February 2006.

[7] V. Gupta, S. Gupta, S. Chang, and D. Stebila, "Performance Analysis of Elliptic Curve Cryptography for SSL," Proc. *ACM Workshop Wireless Security*, pp. 87-94, Sept. 2002.

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

[9] D. G. Filho and P. S. L. M. Barreto, "Demonstrating data possession and uncheatable data transfer", In *IACR Cryptology ePrint Archive*, 2006.

[10] G. Caronni and M. Waldvogel, "Establishing Trust in Distributed Storage Providers", In *Third IEEE P2P Conference*, Linkoping 03, 2003.

[11] T. Schwarz, and E. L. Miller, "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.

[12] P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility", In *Proceedings of HotOS VIII*, May 2001.