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

Research Report RR-08-208

A Security Protocol for Self-Organizing Data Storage

January 21st, 2008

Nouha Oualha, 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

A Security Protocol for Self-Organizing Data Storage

Nouha Oualha, 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 peers generate a proof that they still conserve the data they are supposed to be storing. 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 elliptic curve cryptography. The security of this scheme relies both on the ECDLP intractability assumption and on the difficulty of finding the order of some specific elliptic curve over $\mathbb{Z}_n$. 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: cryptographic protocols, proof of data possession, proof of knowledge, self-organization, online data storage

[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.

A Security Protocol for Self-Organizing Data Storage

Nouha Oualha, Melek Önen, Yves Roudier

1. INTRODUCTION

Online backup and storage has become an increasingly popular and important application, especially given the increasingly nomadic use of data and the ubiquity of data producing processes in our environment. Self-organization, as illustrated by the development of P2P applications like AllMyData, today represents the best approach to achieving scalable and fault-tolerant storage, even though it proves far more demanding in terms of security than classical distributed or remote storage. 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-organizing 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-organizing storage infrastructure. In contrast with simple integrity checks, which make sense only with respect to a potentially defective yet trusted server, efficient primitives need to be defined 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; it should neither make it necessary to store the entire data at the verifier. The latter requirement simply forbids the use of plain message integrity codes as a protection measure since it prevents the construction of time-variant challenges based on such primitives.

This paper presents a secure and self-organizing storage 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 small-sized security metadata. The security of the storage scheme relies on the hardness of specific problems in elliptic curve cryptography. The protocol is also especially original with respect to scalability: it both enables data replication while preventing peer collusion, and delegation of data storage verification to third parties. The latter feature proves specifically interesting since verification, and not only storage, can be distributed.

The paper is structured as follows: Section 2 outlines the security and efficiency requirements that a self-organized data storage protocol must meet, and also discusses existent solutions that have been proposed to address data storage selfishness. Section 3 describes all security primitives required both for storing, personalizing and verifying the storage of data. Section 4 describes how to define a selfishness-resilient storage protocol based on this primitive and analyzes its security and performance. Section 5 finally describes a refinement of this protocol to handle other security threats.

2. PROBLEM STATEMENT

This section describes the requirements that should be met by a self-organizing storage protocol and related work about secure data storage.

2.1.Protocol requirements

We consider a self-organizing storage application in which a node, called the data *owner*, replicates its data by storing them at several peers, called data *holders*. The latter entities agree to keep data for a predefined period of time negotiated with the owner. Their 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. Enforcing such a periodic verification of the data holder has implications on the architecture, performance, and security of the storage protocol, which must fulfill requirements reviewed under the following three subsections.

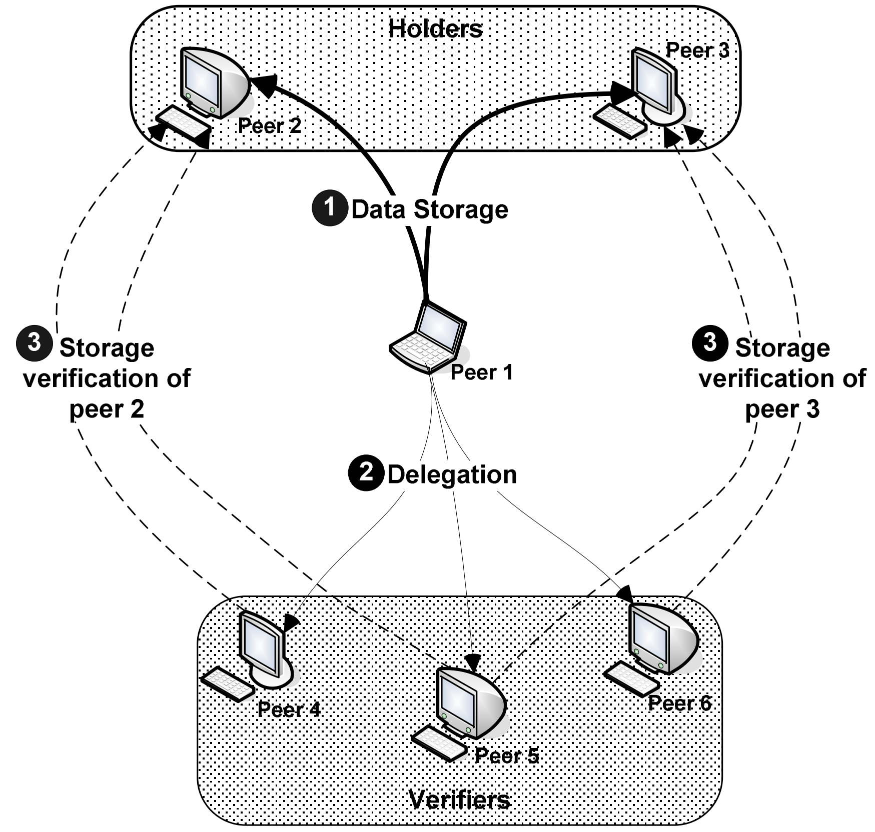


Figure 1. Architecture of a self-organizing storage system. (1) The data owner (Peer 1) requests storage from two holders (peers 2 and 3). (2) Peer 1 delegates the verification of its data to three verifiers (peers 4, 5, and 6) (two verifiers per holder). (3) The verifiers periodically check the behavior of holders.

2.1.1. Architecture

Self-organization addresses highly dynamic environments like P2P networks in which peers frequently join and leave the system: this assumption implies the need for the owner to delegate data storage evaluation to third parties, termed *verifiers* thereafter, to ensure a periodic evaluation of holders after his leave (see Figure 1). That interactive check may be formulated as a proof of knowledge [16] in which the holder attempts to convince a verifier that it possesses some data, which is demonstrated by correctly responding to queries that require computing on the very data.

The need for scalability also pleads for distributing the verification function, in particular to balance 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; data should also be replicated to ensure their high availability, which can only be maintained at a given level if it is possible to detect storage defection.

2.1.2. 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.

- **Storage usage.** 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 holder must also be optimized. The holder should store the minimum extra information in addition to the data itself
- **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.
- **CPU usage.** Response verification and its checking during the verification process respectively at the holder and at the verifier should not be computationally expensive.

Some authors have emphasized how the storage usage and communication overhead requirements differ between verification primitives for secure self-organizing storage and classical proof of knowledge protocols through the use of a specific terminology: *proofs of data possession* for [5], and *proofs of retrievability* for [7].

2.1.3. Security

The verification mechanism must address the following potential attacks which the data storage protocol is exposed to:

- **Detection of data destruction.** The destruction of data stored at a holder must be detected as soon as possible. Destruction may be due to generic data corruption or to a faulty or dishonest holder.
- **Denial-of-Service (DoS) prevention.** DoS attacks aim at disrupting the storage application. Possible DoS attacks are:
    - Flooding attack: the holder may be flooded by verification requests from dishonest verifiers, or from attackers that have not been delegated by the owner. The verifier may be as well subject to the same attack.
    - Replay attack: a valid challenge or response message is maliciously or fraudulently repeated or delayed so as to disrupt the verification.
- **Collusion-resistance.** Collusion attacks aim at taking unfair advantage of the storage application. There is one possible attack: replica holders may collude so that only one of them stores data, thereby defeating the purpose of replication to their sole profit.

The main security problem is the detection of data destruction combined with the risk of collusion between holders. We propose security primitives to handle this problem based on proof of possession and personalization mechanisms. Prevention means against DoS attacks are presented in the refined version of the secure storage protocol.

2.2. Related work

The security of online backup and storage applications has been increasingly addressed in recent years, which has resulted in various approaches to the design of storage verification primitives. The literature distinguishes two main categories of verification schemes: probabilistic ones that rely on the random checking of portions of stored data and deterministic ones that check the conservation of a remote data in a single, although potentially more expensive operation.

2.2.1. Probabilistically secure storage

Probabilistic schemes allow the verification of a portion of data at holders. The POR protocol (Proof of Retrievability) in [7] is based on verification of sentinels which are random values independent of the owner's data. These sentinels are disguised among owner's data blocks. The verification is probabilistic with the number of verification operations allowed being limited to the number of sentinels.

In the solution of [1], the owner periodically challenges its holders by requesting a block out of the stored data. The response is checked by comparing it with the valid block stored at the owner's disk space. Thus, the owner obtains a proof of possession from the holder corresponding to one data chunk out of the full data. Compared to [7], the number of challenges that the owner can conduct is unlimited.

A probabilistic verification approach using Merkle trees proposed by Wagner is reported in [6]. The data stored at the holder is expanded with a Merkle hash tree on data chunks and the root of the tree is kept by the verifier. It is not required from the verifier to store the data. The verification process checks the possession of one data chunk chosen randomly by the verifier that requests as well a full path in the hash tree from the root to this random chunk.

[3] proposes a probabilistic verification approach similar to [1] and the Merkle-based solution mentioned in [6]. However, the verifier is not required to keep the whole data for verification. In counterpart, the holder keeps not only the data but an extra-information used for the verification: the holder is holding data chunks along with their signatures. The verifier chooses probabilistically one index of these chunks. The index is sent to the holder which will answer with the corresponding chunk with its signature. The verifier checks then the signature validity of the chunk. [4] demonstrates that this scheme succeeds with high probability in detecting selfish holders, and also that this probability of success increases with the number of chunks verified per challenge sent by the verifier. However, the communication costs linearly increase with this because the holder has to send all requested chunks with their signatures. The PDP model (Provable Data possession) in [5] does not employ signature as integrity proofs but presents new forms of integrity proofs of data chunks: homomorphic verifiable tags whereby any number of them chosen randomly can be compressed into just one value by far smaller in size than these actual tags.

A further probabilistic approach proposed in [11] does not need any extra information for verification to be stored in the system. The approach permits checking data by requesting algebraic signatures of data blocks stored at holders, and then comparing the parity of these signatures with the signature of the parity blocks stored at holders too. The main drawback of the approach is that if the parity blocks do not match, it is difficult (depends on the number of parity blocks used) and computationally expensive to recognize the faulty holder.

2.2.2. Deterministically secure storage

Deterministic solutions allow verifying the storage of the full data at each holder. 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. One can also follow a similar approach using the modulo operation (less expensive than the hash operation): the verifier keeps a finite set $\{d \bmod n_i\}_i$ where d is an integer that maps to the data and n_i are random numbers, and challenges the holder by sending one of these random numbers. Both protocols require less storage at the verifier, yet they allow only a fixed number of challenges to be performed.

A simple deterministic approach with unlimited number of challenges is proposed in [2] where the verifier like the holder is storing the data. In this approach, the holder has to send the MAC of data as the response to the challenge message. The verifier 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 result of the hashing of the data.

The SEC scheme (Storage Enforcing Commitment) in [6] proposes a deterministic verification approach where the verifier is not required to keep neither data nor any extra information. The approach uses similar homomorphic tags as in [5] whose number is equal to two times the number of data chunks and makes the verifier choose a random value that will be used to shift the indexes of tags to be associated with the data chunks when constructing the response by the holder.

The second solution described in [8] uses an RSA-based proof. This solution requires little storage at the verifier side yet makes it possible to generate an unlimited number of challenges. A similar RSA-based solution is described by Filho and Barreto in [9] that makes use of a key-based homomorphic hash function H. In each challenge of this solution, a nonce is generated by the verifier which the prover combines with the data using H to prove the freshness of the answer. The prover's response will be compared by the verifier with a value computed over $H(data)$ only, since the secret key of the verifier allows the following operation (d for data, and r for nonce): $H(d + r) = H(d) \times H(r)$. The exponentiation operation used in the RSA solution makes the whole data as an exponent. To reduce the computing time of verification, Sebé et al. in [10] 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 m of chunks $\{d_i\}_{1\leq i\leq m}$, the verifier holds $\{H(d_i)\}_{1\leq i\leq m}$ and asks the prover to compute a sum function of the data chunks $\{d_i\}_{1\leq i\leq m}$ and m random numbers $\{r_i\}_{1\leq i\leq m}$ generated from a new seed handed by the verifier for every challenge. Here again, the secret key kept by the verifier allows this operation: $\sum_{1\leq i\leq m} H(d_i + r_i)= \sum_{1\leq i\leq m} H(d_i) \times H(r_i)$. The index m is the ratio of tradeoff between the storage kept by the verifier and the computation performed by the prover.

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

Table 1. Comparison of existing verification protocols.
The variable n corresponds to the data size.

			Storage at verifier	CPU at holder	Communication overhead
Probabilistic verification	Limited verifications	POR protocol [7]	$O(1)$	$O(1)$ hash transformation	$O(1)$
	Unlimited verifications	Lillibridge et Al. [1]	$O(n)$	$O(1)$ simple comparison	$O(1)$
		Wagner [6]	$O(1)$	$O(log(n))$ hash transformation	$O(log(n))$
		Oualha and Roudier [3]	$O(1)$	$O(1)$ signature validation	$O(1)$
		PDP model [5]	$O(1)$	$O(1)$ exponentiation	$O(1)$
		Schwarz, and Miller [11]	$O(1)$	$O(1)$ signature validation	$O(1)$
Deterministic verification	Limited verifications	Deswarte et Al. [8]: pre-computed challenges	$O(1)$	$O(1)$ hash transformation	$O(1)$
	Unlimited verifications	Caronni and Waldvogel [2]	$O(n)$	$O(n)$ hash transformation	$O(1)$
		SEC scheme [6]	$O(1)$	$O(1)$ exponentiation	$O(1)$
		Deswarte et Al. [8], Filho and Barreto [9]: RSA solution	$O(1)$	$O(n)$ exponentiation	$O(1)$
		Sebé et Al. in [10]	$O(1)$	$O(1)$ exponentiation	$O(1)$

Considering all these security and performance goals, we propose in the next section a verification protocol whereby an owner that whishes to store a data generates individual replicas for holders, and delegates the verification to different nodes.

3. SECURE STORAGE SCHEME

A secure storage scheme $\mathscr{V}$ is a three-party protocol executed between an owner, a holder, and a verifier. The scheme is specified by four phases: *Setup, Storage, Delegation,* and *Verification*. The owner communicates the data to the holder at the *storage* phase and the meta-information to the verifier at the *delegation* phase. At the *verification* phase, the verifier checks the holder's possession of data by invoking an interactive process. This process may be executed an unlimited number of times.

- ***Setup***: The owner setups the scheme by running a `setup` algorithm that takes in input a security parameter k and returns system parameters (see Figure 2).

Owner

$params \leftarrow \texttt{Setup}(k)$

Figure 2. Setup phase

- ***Storage***: The owner stores its data at one or several holders. Each holder is storing a personalized version of data. For data personalization, the owner runs a `Personalize` algorithm that takes in input the data and the identity of the holder and returns an identity-based encrypted version of data (we assume that nodes are uniquely identified in the system) (see Figure 3).

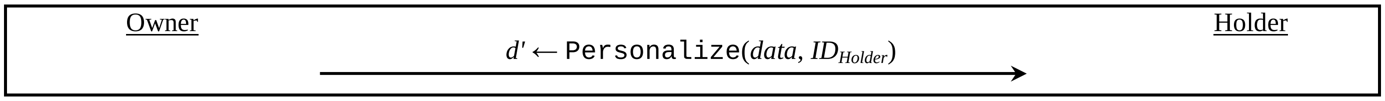


Figure 3. Storage phase

- ***Delegation***: The owner generates meta-information to be used by the verifier for verifying the data possession of one holder. Therefore, the owner runs a `MetaGen` algorithm that outputs the meta-information needed for verification, by taking as input the identity of the very holder and the data that it stores. The meta-information is a reduced-size digest of the data stored at the holder. This meta-information is sent for storage to the verifier that is responsible of verifying the holder (see Figure 4).

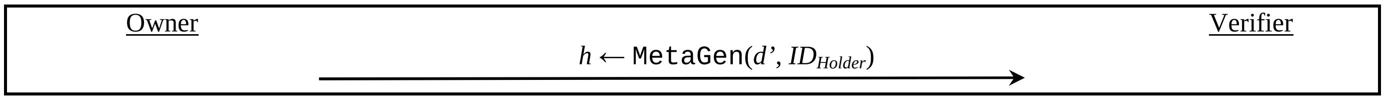


Figure 4. Delegation phase

- ***Verification:*** The verifier checks the presence of data at the holder. It generates a challenge by running a `Challenge` algorithm that returns the variable to be sent to the holder. Upon reception of this message, the holder runs the `Response` algorithm that takes in input the data stored and the challenge variable and outputs a proof to be sent to the verifier. With this proof as input, the verifier runs the `Check` algorithm that returns a Boolean variable to decide whether the proof is accepted or rejected (see Figure 5).

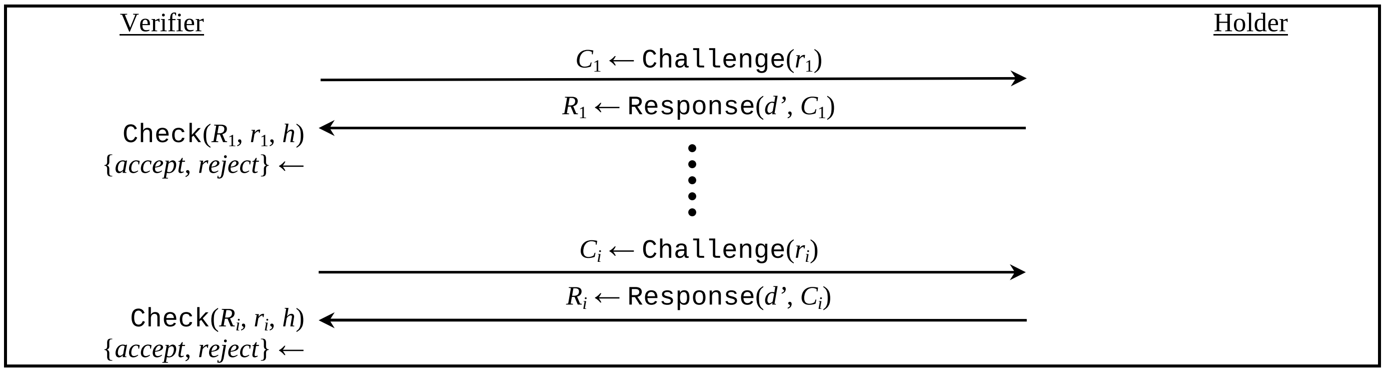

Verifier
Holder
$C_1 \leftarrow \texttt{Challenge}(r_1)$
$R_1 \leftarrow \texttt{Response}(d', C_1)$
$\texttt{Check}(R_1, r_1, h)$
$\{accept, reject\} \leftarrow$
$C_i \leftarrow \texttt{Challenge}(r_i)$
$R_i \leftarrow \texttt{Response}(d', C_i)$
$\texttt{Check}(R_i, r_i, h)$
$\{accept, reject\} \leftarrow$

Figure 5. Verification phase

The remainder of this section details our verification scheme incrementally: essential notions in elliptic curve cryptography and hard problems used in our security scheme are first introduced; two versions of the security primitives are then described.

3.1. Security background

Our secure storage protocol relies on elliptic curve cryptography ([12], [13]). The security of the protocol is based on two different hard problems. First, given some required conditions, it is hard to find the order of an elliptic curve. Furthermore, one of the most common problems in elliptic curve cryptography is the Elliptic Curve discrete logarithm problem denoted by ECDLP. Thanks to the hardness of these two problems, our secure storage protocol ensures that the holder must use the whole data to compute the response for each challenge. In this section, we formalize these two problems in order to further describe the security primitives that rely on them.

3.1.1. Elliptic Curves over $\mathbb{Z}_n$

Let n be an odd composite square free integer and let a, b be two integers in $\mathbb{Z}_n$ such that $gcd(4a^3 + 27b^2, n) = 1$ ("*gcd*" means greatest common divisor). An elliptic curve $E_n(a, b)$ over the ring $\mathbb{Z}_n$ is the set of the points $(x, y) \in \mathbb{Z}_n \times \mathbb{Z}_n$ satisfying the equation: $y^2 = x^3 + ax + b$, together with the point at infinity denoted O_n.

3.1.2. Solving the order of elliptic curves

The order of an elliptic curve over the ring $\mathbb{Z}_n$ where $n=pq$ is defined in [14] as $N_n = lcm(\#E_p(a, b), \#E_q(a, b))$ ("*lcm*" for least common multiple, "#" means order of). N_n is the order of the curve, i.e., for any $P \in E_n(a, b)$ and any integer k, $(k N_n + 1)P = P$.

If ($a = 0$ and $p \equiv q \equiv 2 \ mod\ 3$) or ($b = 0$ and $p \equiv q \equiv 3 \ mod\ 4$), the order of $E_n(a, b)$ is equal to $N_n = lcm(p+1, q+1)$. We will consider for the remainder of the paper the case where $a = 0$ and $p \equiv q \equiv 2 \ mod\ 3$. As proven in [14], given $N_n = lcm(\#E_p(a, b), \#E_q(a, b)) = lcm(p + 1, q + 1)$, solving N_n is computationally equivalent to factoring the composite number n.

3.1.3. The elliptic curve discrete logarithm problem

Consider K a finite field and $E(K)$ an elliptic curve defined over K. **ECDLP** in K is defined as given two elements P and $Q \in K$, find an integer r, such that $Q = rP$ whenever such an integer exists.

3.2. Data-based security primitives

The data file, stored in the system, is uniquely mapped into a number $d \in \mathbb{N}$ in some publicly known way (for example, conversion from binary representation into decimal representation). In our context, the terms data file or data and the number d are often used interchangeably. As shown above, the secure storage scheme relies on six polynomial time algorithms: `Setup`, `Personalize`, `MetaGen`, `Challenge`, `Response`, and `Check`. The construction of these algorithms is presented as follows.

- **`Setup`**: The algorithm (Figure 6) is run by the owner during the *setup* phase. It takes in input a chosen security parameter k (k > 512 bits). It generates two large primes p and q of size k both congruent to 2 modulo 3, and computes their product $n = pq$. Then, it considers an elliptic curve over the ring $\mathbb{Z}_n$ denoted by $E_n(0, b)$ where b is an integer such that $gcd(b, n)=1$, to compute a generator P of $E_n(0, b)$. The order of $E_n(0, b)$ is $N_n = lcm(p+1, q+1)$. The algorithm outputs b, n, P, and N_n. The parameters b, P, and n are published and the order N_n is kept secret by the owner.

```
Setup(k):
  1. generate two primes $p$ and $q$ of size $k$: $p, q \equiv 2 \bmod 3$
  2. compute $n = pq$
  3. compute $N_n = lcm(p+1, q+1)$
  4. generate random integer $b < n$, $gcd(b, n)=1$
  5. compute $P$ a generator of $E_n(0, b)$
  6. output (public = $(n, b, P)$, secret = $N_n$)
```

Figure 6. Setup algorithm

- **Personalize**: The algorithm (Figure 7) is run by the owner during the *storage* phase. The algorithm takes in input the data d to be personalized and a secret random number s (of size l). This algorithm requires a keyed incompressible function $f_{ID_{Holder}}: \{0, 1\}^l \rightarrow \{0, 1\}^{|d|}$ (like in [15]) where ID_{Holder} is the unique identity of the holder and $|d|$ is the size of the data file. The algorithm outputs $d' = d + f_{ID_{Holder}}(s)$.

```
Personalize(d, s):
  1. compute $d' = d + f_{ID_{Holder}}(s)$
  2. output $d'$
```

Figure 7. Personalize algorithm

- **MetaGen**: The algorithm (Figure 8) is run by the owner during the *delegation* phase. The algorithm takes in input the data d' and returns $T = (d' \bmod N_n)P \in E_n(0, b)$. The point T is stored by the verifier.

```
MetaGen(d', n, N_n, b, P):
  1. compute $T = (d' \bmod N_n)P$
  2. output $T$
```

Figure 8. MetaGen Algorithm

- **Challenge**: The algorithm (Figure 9) is run by the verifier to start the *verification* phase. It takes in input a random number r and returns the point $Q = rP \in E_n(0, b)$. The point Q will be sent to the holder as a challenge.

```
Challenge(r, n, b, P):
  1. compute $Q = rP$
  2. output $Q$
```

Figure 9. Challenge algorithm

- **Response**: The algorithm (Figure 10) is run by the holder during the *verification* phase. It takes in input a point Q and data d' and outputs $R = d'Q \in E_n(0, b)$. The point R is sent to the verifier as a response to a challenge. To reduce communication overhead, the holder may just send the abscissa of the point R.

```
Response(d', Q, n, b):
  1. compute $R = d'Q$
  2. output $R$
```

Figure 10. Response algorithm

- **Check**: The algorithm (Figure 11) is run by the verifier at the end of the *verification* phase. The algorithm takes in input the response R, the random number r of the challenge, and the metadata T. It verifiers if R is equal to rT, and decides if the holder's proof is accepted or rejected.

```
Check(R, r, T, n, b):
    1. compute $rT$
    2. if $R = rT$ then output “accept” else then output “reject”
```

Figure 11. Check algorithm

With the presented security primitives, the verifier keeps an extra-information (T) needed for the verification that is twice the size of n ($< 2k$) which is assumed smaller than the size of the stored data (2Kb compared with 100Mb or 1Gb of data). For verification, the verifier has to compute two point multiplications with a small random number, in contrast with the holder who has to compute a point multiplication with the whole data.

3.3. Chunk-based security primitives

This section introduces an improved version of these security primitives described above whereby the computation complexity at the holder is reduced. In this version, the data is split into m chunks, denoted $\{d_i\}_{1\leq i\leq m}$, and the verifier stores the corresponding elliptic curve points $\{T_i = d_iP\}_{1\leq i\leq m}$. We assume that the size of each data chunk is much larger than $4k$ where k is the security parameter that specifies the size of p and q, because the verifier must keep less information than the full data.

The owner runs the `Setup` algorithm during the *setup* phase like in the previous version. It personalizes the data by running the `Personalize` algorithm of the previous version, then splits the personalized data in m chunks of the same size (the last chunk is padded with zeroes): $\{d'_i\}_{1\leq i\leq m}$.

The owner runs a new algorithm, `ChunkMetaGen`, to obtain the curve points $\{T_i\}_{1\leq i\leq m}$ sent to the verifier.

- **`ChunkMetaGen`:** This algorithm (Figure 12) is run by the owner during the *delegation* phase. It takes in input the set of personalized data chunks $\{d'_i\}_{1\leq i\leq m}$ and returns the set of metadata $\{T_i = d'_iP\}_{1\leq i\leq m}$.

```
ChunkMetaGen($\{d'_i\}_{1\leq i\leq m}$, n, b, P):
    for $i$ in 1 to $m$ do
        1. $T_i \leftarrow$ MetaGen($d'_i$, $n$, $b$, $P$)
    end
    2. output $\{T_i\}_{1\leq i\leq m}$
```

Figure 12. ChunkMetaGen algorithm

During the *verification* phase, the verifier generates a random seed c (size of $c > 128$ bits). Then, it runs the `Challenge` algorithm as in the previous version. It sends the resulted output Q and the seed c to the holder. Upon reception of this, the holder runs a new algorithm, `ChunkResponse`, to obtain the response to the challenge that is sent back to the verifier.

- **`ChunkResponse`:** This algorithm (Figure 13) is run by the holder at the *verification* phase. It takes in input the data chunks $\{d'_i\}_{1\leq i\leq m}$ and the seed c. It generates m random numbers $\{c_i\}_{1\leq i\leq m}$ from the seed c. Finally, it outputs the point $R = \sum_{1\leq i\leq m} c_i d'_i Q$.

```
ChunkResponse($\{d'_i\}_{1\leq i\leq m}$, Q, c, n, b):
    1. generate $m$ random numbers from the seed $c$: $\{c_i\}_{1\leq i\leq m}$
    2. compute $d'_c = \sum_{1\leq i\leq m} c_i d'_i$
    3. $R \leftarrow$ Response($d'_c$, $Q$, $n$, $b$)
    4. output $R$
```

Figure 13. ChunkResponse

To decide whether holder’s proof is accepted or rejected, the verifier runs a new algorithm: the `ChunkCheck` algorithm.

- **`ChunkCheck`**: This algorithm (Figure 14) is run by the verifier at the end of the *verification* phase. It takes in input the point R sent back by the holder, the random number of the challenge r, the seed c, and the stored set of metadata $\{T_i\}_{1\leq i\leq m}$. The algorithm uses these metadata points to compute a sum that will be compared with the holder’s response. The algorithm returns an accept statement if the equality holds, and a reject statement otherwise.

```
ChunkCheck($R, r, c, \{T_i\}_{1 \le i \le m}, n, b$):
    1. generate $m$ random numbers from the seed $c$: $\{c_i\}_{1 \le i \le m}$
    for $i$ in 1 to $m$ do
        2. compute $c_i T_i$
    end
    3. compute $\sum_{1 \le i \le m} c_i T_i$
    4. compute $r(\sum_{1 \le i \le m} c_i T_i)$
    5. if $R = r(\sum_{1 \le i \le m} c_i T_i)$ then output “accept” else then output “reject”
```

Figure 14. ChunkCheck algorithm

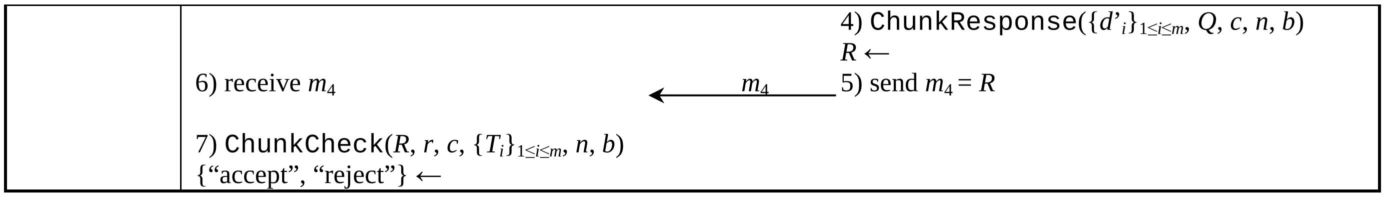
Compared with the previous version of primitives where the data is considered as a whole, these new security primitives make the holder compute m point multiplications of the same elliptic curve point where the size of the scalar is the size of the data chunk instead of the full data. Also, the verifier has to keep m points instead of one point in the previous version. The number of chunks m is the ratio of tradeoff between the storage required at the verifier and the computation consumed at the holder (the case $m = 1$ corresponds to the previous version of primitives).

For performance reasons, it is possible to dispense the holder and the verifier from using the seed c. The holder computes, for each i in $[1, m]$ $R_i = d'_i Q$, and then the sum $\sum_{1 \le i \le m} i[R_i]_x \bmod n$ where $[R_i]_x$ denotes the abscissa of the elliptic curve point R_i. The verifier computes the same sum but with the metadata it is storing: $\sum_{1 \le i \le m} i[rT_i]_x \bmod n$. Then, it compares this sum with the holder’s response, which significantly reduces the CPU usage.

4. PROTOCOL EVALUATION

In this section, we evaluate the proposed secure storage protocol considering both the security and the performance aspects. We remind the protocol in Figure 15.

Phases	Description		
Setup	Owner 1) (public = (n, b, P), secret = N_n) $\leftarrow$ `Setup`(k)		
Storage	Owner 1) $d' \leftarrow$ `Personalize`(d, s) 2) split d' into m chunks: $\{d'_i\}_{1 \le i \le m}$ 3) send $m_1 = \{d'_i\}_{1 \le i \le m}$	$\xrightarrow{m_1}$	Holder 4) receive m_1 5) keep $\{d'_i\}_{1 \le i \le m}$
Delegation	Owner 1) `ChunkMetaGen`($\{d'_i\}_{1 \le i \le m}, n, b, P$) $\{T_i\}_{1 \le i \le m} \leftarrow$ 2) send $m_2 = \{T_i\}_{1 \le i \le m}$	$\xrightarrow{m_2}$	Verifier 3) Receive m_2 4) keep $\{T_i\}_{1 \le i \le m}$
Verification	Verifier 1) $Q \leftarrow$ `Challenge`(r, n, b, P) 2) send $m_3 = Q$	$\xrightarrow{m_3}$	Holder 3) receive m_3



4) `ChunkResponse`($\{d'_i\}_{1\le i\le m}$, Q, c, n, b)
$R \leftarrow$
6) receive m_4 $\xleftarrow{m_4}$ 5) send $m_4 = R$

7) `ChunkCheck`(R, r, c, $\{T_i\}_{1\le i\le m}$, n, b)
{“accept”, “reject”} $\leftarrow$

Figure 15. Secure storage verification protocol

4.1.Security analysis

This section analyzes the completeness and the soundness of the protocol that are the two essential properties of a proof of knowledge protocol [16]: 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: i****f the verifier and the holder correctly follow the proposed protocol, the verifier always accepts the proof as valid.*

Proof: Thanks to the commutative property of point multiplication in an elliptic curve, we have for each i in [1,m]: $d'_i rP = rd'_i P$. Thus, the equation: $\sum_{1\le i\le m} c_i d'_i rP = r(\sum_{1\le i\le m} c_i d'_i P)$. □

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

Proof: If the holder does not store the data chunks $\{d'_i\}_{1\le i\le m}$, it may try first to collude with other holders storing the same data. However, this option is not feasible since data stored at each holder is securely personalized during the storage phase. Since f is a one-way function and the parameter s is secret, no node except the owner can retrieve the original data d from d'. The other way to generate a correct response without storing the data relies on only storing $\{d'_i P\}_{1\le i\le m}$ (which is much smaller than the full data size) and retrieving r from the challenge rP in order to compute the correct response. Finding r is hard based on ECDLP. The last option for the holder to cheat is to keep $\{d'_i \bmod N_n\}_{1\le i\le m}$ instead of d' (whose size is very large). The holder cannot compute N_n based on the hardness of solving the order of $E_n(0, b)$. Thus, if the response is correct then the holder keeps the data correctly. □

4.2.Performance analysis

The suggested verification protocol consists of four phases, only three of them are considered indispensable since the owner can be the verifier of the data (delegation phase is optional). The first and the second phase correspond to a traditional storage of data in the system. The *verification* phase comprises authentication messages, then challenge-response messages. Thus, we consider for analysis only the verification part of the protocol.

In the proposed protocol, challenge-response messages mainly consist each of an en elliptic curve point $\in \mathbb{Z}_n^2$. Thus, messages’ size is function of the security factor k (size of $n \approx 2k$). Reducing communication overhead then means decreasing the parameter of security. So, the parameter k is one parameter of tradeoff between efficiency and security.

The verification protocol requires from the verifier to store a set of elliptic curve points that allows producing on demand challenges for the verification. Finally, the creation of proof and its verification rely on point multiplication operations.

The number of data chunks m is a variable of tradeoff between the storage required at the verifier and the computation expected from the holder: when increasing m, the verifier is required to keep more information for the verification task, but at the same time the holder is required to perform one point multiplication operation using much smaller scalar. Thus, m is a ratio of tradeoff between the amount of storage the verifier volunteers to concede and the complexity of computation the holder agrees to perform.

Table 2. Summary of resource usage of the verification protocol.
The variable n corresponds to the data size (number of data chunks is assumed constant).

Communication overhead		Storage space usage		Computation complexity	
Challenge size	Response size	At holder	At verifier	At holder	At verifier
$O(1)$	$O(1)$	$O(n)$	$O(1)$	$O(1)$ point multiplication operations + $O(1)$ sum of points	$O(1)$ sum of integers + $O(1)$ point multiplication operations

5. PROTOCOL REFINEMENT

As explained previously, four phases are defined in the secure storage protocol: *setup*, *storage*, *delegation,* and *verification*. At each phase, nodes run the secure storage primitives that are described in Section 3.3. However, the protocol is still weak against some security threats described in section 2.1.3. This section refines the security mechanisms introduced in the protocol to address these additional attacks. Every node in the framework is assumed to be uniquely identified by an identifier denoted by ID_N.

External DoS attacks. At each phase of the protocol, messages are authenticated with common signature algorithms such as RSA. Therefore each node possesses a pair of public and private keys $\{PK_N, SK_N\}$.This authentication inherently prevents external Denial of Service (DoS) attacks whereby intruders generate some flooding attacks against holders. Only authorized verifiers are allowed to run the verification phase. In order to provide this security restriction, during the delegation phase, the owner provides each verifier a credential that allows the verifier to run the verification phase (see Figure 16). Therefore, verifiers generate a signature σ for each message at the verification phase and send this signature and their credentials together with the challenge message. 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 or generating a digital signature. Moreover, thanks to this technique message integrity is also provided.

	Owner	Verifier
Delegation	1 – b) compute $Cert(O, ID_{Verifier}, ID_{Holder}) = Sign_{SK_{Owner}}(ID_{Owner}, ID_{Verifier}, ID_{Holder}, validity, c_{Th})$ 2) send $m_2 = \{T_i\}_{1\leq i\leq m}, Cert(O, ID_{Verifier}, ID_{Holder}), \sigma_{m2}$ $\xrightarrow{m_2,\ \sigma_{m2}}$	

Figure 16. Delegation certification

Internal DoS attacks. In addition to external DoS attacks, some authorized verifiers might also generate some flooding attacks against holders. In this particular case, authentication is not a direct solution since verifiers are authorized to participate to the communication. We thus first propose to limit the number of verifiers that can request each holder. This number is 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. Hence, the holder keeps a quota counter c_{Th} for each authorized verifier that is incremented at each new challenge (see Figure 17). If this counter exceeds the given threshold value during a time interval, the verifier is not allowed to challenge the holder and the challenge message is automatically dropped.

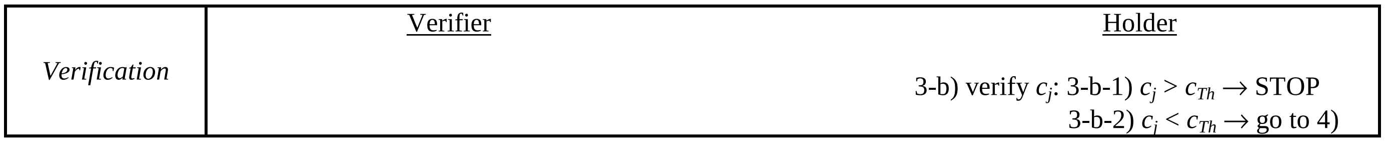

Verification
Verifier
Holder
3-b) verify c_j: 3-b-1) $c_j > c_{Th} \rightarrow$ STOP
3-b-2) $c_j < c_{Th} \rightarrow$ go to 4)

Figure 17. Quota enforcement

Replay attacks. Replay attacks whereby valid challenge messages are maliciously repeated or delayed so as to disrupt the verification phase are also taken into consideration. Indeed, during the verification phase, in addition to the challenge message, its signature and the corresponding credential, the verifier will send a newly generated nonce n_k (see Figure 18).Thanks to this well-known technique, the holder will be able to automatically detect replay attacks.

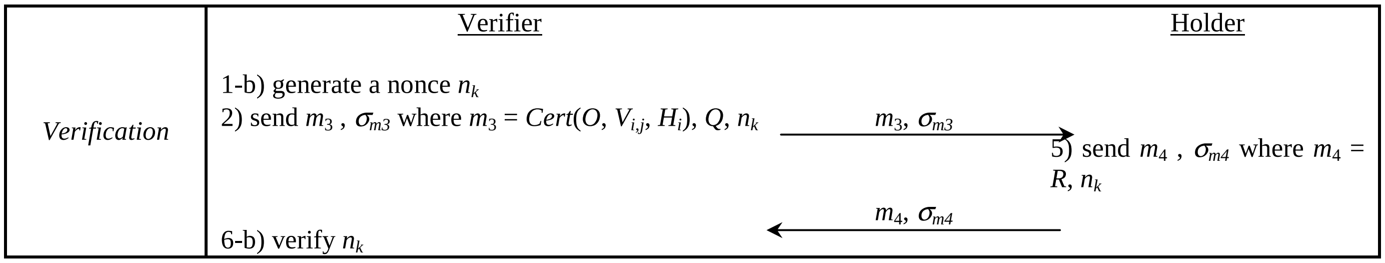

Verification
Verifier
Holder
1-b) generate a nonce n_k
2) send m_3 , σ_{m3} where $m_3 = Cert(O, V_{i,j}, H_i), Q, n_k$
m_3, σ_{m3}
5) send m_4 , σ_{m4} where $m_4 = R, n_k$
m_4, σ_{m4}
6-b) verify n_k

Figure 18. Replay protection

6. CONCLUSION

We presented in this paper a protocol that satisfies the performance, and security requirements of self-organizing storage applications. Its security relies on an elliptic curve cryptographic scheme which we showed makes use of very limited resource. The security mechanisms which were developed in this paper make it possible to verify whether a data storing peer that responds to a challenge still possesses some data as it claims, and without sacrificing security for performance. This verification can also be delegated to third party verifiers, thereby fulfilling an essential architectural requirement of self-organizing storage.

Assessing the actual state of storage in such an application represents the first step towards efficiently reacting to misbehavior: active replication strategies can be built based on such evaluations and we are investigating how to securely rejuvenate the replicas of some data under attack. We are also actively working on the construction of cooperation incentives using this protocol as an observation primitive. Stimulating peer cooperation is however more complex than assessing their instantaneous cooperation with the execution of a challenge-response protocol. The use of a cooperation stimulation scheme should ultimately make it possible to detect and isolate selfish and malicious peers.

REFERENCES

[1] 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.

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

[3] N. Oualha, Y. Roudier, “Securing ad hoc storage through probabilistic cooperation assessment”, WCAN '07, 3rd Workshop on Cryptography for Ad hoc Networks, July 8th, 2007, Wroclaw, Poland.

[4] N. Oualha, P. Michiardi, Y. Roudier, “A game theoretic model of a protocol for data possession verification”, TSPUC 2007, IEEE International Workshop on Trust, Security, and Privacy for Ubiquitous Computing, June 18, 2007, Helsinki, Finland.

[5] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson and D. Song, “Provable data possession at untrusted stores”CCS '07: *Proceedings of the 14th ACM conference on Computer and communications security, ACM, 2007*, 598-609.

[6] P. Golle, S. Jarecki and I. Mironov**,** “Cryptographic Primitives Enforcing Communication and Storage Complexity”, In proc. of *Financial Crypto 2002*.

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

[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] Francesc Sebe, Josep Domingo-Ferrer, Antoni Martínez-Ballesté, Yves Deswarte, Jean-Jacques Quisquater, “Efficient Remote Data Possession Checking in Critical Information Infrastructures,” *IEEE Transactions on Knowledge and Data Engineering*, 06 Aug 2007. IEEE Computer Society Digital Library. IEEE Computer Society, 6 December 2007 http://doi.ieeecomputersociety.org/10.1109/TKDE.2007.190647

[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] N. Koblitz, “Elliptic curve cryptosystems”, Mathematics of Computation, 48 (1987), 203-209.

[13] V. Miller, “Uses of elliptic curves in cryptography”, advances in Cryptology, *Proceedings of Crypto’85*, Lecture Notes in Computer Science, 218 (1986), Springer-Verlag, 417-426.

[14] Kenji Koyama, Ueli Maurer, Tatsuaki Okamoto, and Scott Vanstone, “New Public-Key Schemes Based on Elliptic Curves over the Ring Z_n**”,** *Advances in Cryptology - CRYPTO '91*, Lecture Notes in Computer Science, Springer-Verlag, vol. 576, pp. 252-266, Aug 1991.

[15] Cynthia Dwork, Jeffrey Lotspiech, Moni Naor, “Digital signets: self-enforcing protection of digital information (preliminary version)”, *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, p.489-498, May 22-24, 1996, Philadelphia, Pennsylvania, United States.

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