J. Math. Cryptol. **8** (2014), 189–207
DOI 10.1515/jmc-2013-0034

Is extracting data the same as possessing data?

Douglas R. Stinson and Jalaj Upadhyay

Communicated by Spyros Magliveras

Abstract. Proof-of-retrievability schemes have been a topic of considerable recent interest. In these schemes, a client C gives a file M to a server S with the understanding that S will securely store M. A suitable challenge-response protocol is invoked by C in order for C to gain confidence that M is indeed being correctly stored by S. The definition of proof-of-retrievability schemes is based on the notion of an extractor $\mathcal{E}$ that can recover the file once the challenge-response protocol is executed a sufficient number of times. In this paper, we propose a new type of scheme that we term a *proof-of-data-observability scheme*. Our definition tries to capture the stronger requirement that S must have an actual copy of M in its memory space while it executes the challenge-response protocol. We give some examples of schemes that satisfy this new security definition. As well, we analyze the efficiency and security of the protocols we present, and we prove some necessary conditions for the existence of these kinds of protocols.

Keywords. Proof-of-retrievability, proof-of-data-observability.

2010 Mathematics Subject Classification. 94A60.

1 Introduction

There has been considerable recent interest in proof-of retrievability and proof-of-data-possession schemes for cloud storage [1, 3, 4, 8, 11, 13]. We begin by briefly introducing these schemes. Suppose a client C gives a file M (also called a *message*) to a server S (e.g., a cloud storage service provider) with the understanding that S will securely store M. A suitable challenge-response protocol is invoked by C in order for C to gain confidence that M is indeed being correctly stored by S, without actually requiring S to transmit the entire file to C. The server constructs a *proving algorithm* $\mathcal{P}$ that it uses to respond to C's challenges. The success probability of $\mathcal{P}$, denoted $\mathsf{succ}(\mathcal{P})$, is defined to be the probability that $\mathcal{P}$ gives a correct response to a random challenge. We consider the server S along with the proving algorithm $\mathcal{P}$ (which is constructed by S) to comprise the *adversary* in one of these schemes.

D. Stinson's research is supported by NSERC discovery grant 203114-11.

Suppose the message is of the form $M = (M_1, \ldots, M_m)$, where $M_1, \ldots, M_m$ are the *message blocks*. Typical challenge-response protocols require S to supply a certain subset of the message blocks (e.g., [8, 11]), or to compute a specified linear combination of a subset of the message blocks (e.g., [13]). In these example schemes, the *challenge* would specify the message blocks and (if required) the exact linear combination that the server is required to compute. Note that C might store the file M (in this case it can verify the response of S by performing the same computation as S). Alternatively, C might have precomputed and stored one or more responses (in this case it would not be necessary for C to store M). Finally, C might have a "key", which is smaller than the file, but which still can be used to verify responses.

In a *proof-of-retrieval scheme* (*POR scheme*) (e.g., [8]), it is required that there should exist an *extractor* $\mathcal{E}$ with the property that it can recover M, given $\mathcal{P}$, whenever $\mathsf{succ}(\mathcal{P})$ is sufficiently high. In general, $\mathcal{E}$ will perform the following operations:

- $\mathcal{E}$ is given read access to the code describing $\mathcal{P}$,
- $\mathcal{E}$ is allowed to run $\mathcal{P}$ on challenges that are chosen by $\mathcal{E}$,
- $\mathcal{E}$ can perform additional computations, and
- $\mathcal{E}$ should compute and correctly output the entire file M.

The effectiveness of an extractor $\mathcal{E}$ for a POR scheme is usually quantified as follows. Let $0 < \delta \leq 1$ and let $0 < \epsilon \leq 1$. We say that $\mathcal{E}$ is a (δ, ϵ)-*extractor* provided that the success probability of $\mathcal{E}$ is at least ϵ whenever $\mathsf{succ}(\mathcal{P}) \geq \delta$. Recall that $\mathsf{succ}(\mathcal{P})$ is the probability that $\mathcal{P}$ gives the correct response to a random challenge; the success probability of $\mathcal{E}$ is the probability that it is able to extract the entire file M correctly. The success probability of $\mathcal{E}$ takes into account any random choices it makes during its execution.

POR schemes can be analyzed in the setting of computational security [1, 3, 4, 8, 13] or unconditional security [4, 11]. In the setting of computational security, $\mathcal{E}$, S and $\mathcal{P}$ would be modelled as polynomial-time probabilistic Turing machines. In the setting of unconditional security, we would not place any restrictions on the computations that $\mathcal{E}$, S and $\mathcal{P}$ are allowed to perform.

The idea of a *proof-of-data-possession scheme* (*PDP scheme*) is also found in the literature, e.g., a formal definition can be found in [1, Definition 4.2]. This definition also requires an extractor $\mathcal{E}$ to actually recover or restore the data whose "possession" is guaranteed by the responses received by C when the challenge-response protocol is carried out. Furthermore, in [1], it is explicitly permitted for S to store the file in altered form (e.g., using some form of lossless data com-

pression). Thus, it seems in the literature that there is little or no fundamental difference between a POR scheme and a PDP scheme.

Our main goal is to separate the notions of retrievability and possession. (Before proceeding further, we note that all the results presented in this paper are in the standard client-server model in which proof-of-retrievability schemes and proof-of-data-possession schemes have been studied.) To make a distinction between retrievability and possession, it is helpful to ask what it means conceptually or philosophically for S to actually "possess" data. We might argue that possessing data should mean that S has an *exact copy* of the file M in its storage at some given time. However, it would be difficult to force S to do this, because (as mentioned above) S can store M in some altered form, e.g., involving encryption or (lossless) compression. However, it is conceivable that the challenge-response protocol $\mathcal{P}$ might necessitate the reconstruction of M, in the sense that a response cannot be generated without first restoring M to its initial form. In this case, an exact copy of M would exist in the memory space of S at the time the response to the challenge is computed, and we could say that S "possesses" the data at this point in time.

1.1 Formal definitions for proof-of-data-observability

Based on the previous discussion, we now define the new concept of a *proof-of-data-observability scheme* (*PDO scheme*). This definition captures the idea that S actually has a complete copy of M in its memory space during the execution of $\mathcal{P}$. To formalize this stronger security notion, we consider an *observer* $\mathcal{O}$, who has considerably less power than $\mathcal{E}$. The observer $\mathcal{O}$ is permitted to do the following:

- $\mathcal{O}$ is given read access to the code describing $\mathcal{P}$,
- $\mathcal{O}$ is allowed to observe a *single run* of the challenge-response protocol with a *random* challenge supplied by C,
- $\mathcal{O}$ can perform additional computations, and
- $\mathcal{O}$ should copy the correct file M from the contents of S's memory space to an output tape, at some time during the execution of $\mathcal{P}$.

Intuitively, this definition is trying to capture the idea that it is possible to take a "snapshot" of S's memory space (at an appropriate time) that precisely contains the file M.

For a PDO scheme, we have a somewhat different objective than we did for a POR scheme. As before, let $0 < \delta \leq 1$ and let $0 < \epsilon \leq 1$. We say that an observer $\mathcal{O}$ is a (δ, ϵ)-*observer* provided that the expected fraction of M that $\mathcal{O}$ copies is at least ϵ whenever $\mathsf{succ}(\mathcal{P}) \geq \delta$. That is, on average, $\mathcal{O}$ is able to copy at least ϵn bits of M, where M is n bits in length. This average is computed over the

random challenge chosen by C, as well as any random choices $\mathcal{O}$ makes during its execution.

It is important to emphasize that $\mathcal{O}$ watches a single run of the challenge-response protocol that is being executed with a random challenge that is chosen by C (i.e., the challenge is *not* chosen by $\mathcal{O}$). Based on $\mathcal{O}$'s knowledge of how the proving algorithm $\mathcal{P}$ works, $\mathcal{O}$ is able to recognize the presence of the file M in the memory space of S as $\mathcal{P}$ is executed, and copy M to its output tape.

Remark 1.1. It is most useful to think of $\mathcal{E}$ and $\mathcal{O}$ as conceptual entities employed in the security proof. They are not normally used in the standard operation of a POR or PDO scheme (respectively).

It is reasonable to ask why this new notion of observability might be preferable in practice to the standard notion of retrievability. One of the motivations of cloud computing is to outsource computations to the cloud. In general, performing some computation on data stored at a remote server S will be possible only if S has the actual data M in hand. The notion of retrievability does not guarantee that the actual file is readily available. This is because the operations performed by the extractor $\mathcal{E}$ may be very complicated (even if they are required to be polynomial-time operations). We would argue that a PDO scheme provides a better guarantee than a POR scheme that the file M is "available" at a given point in time.

Suppose that an n-bit file M consists of m message blocks, and assume that a response in a challenge-response protocol depends on a random subset of m' message blocks chosen by C. For $\delta > 0$, the best observer we could hope for would be a $(\delta, m'/m)$-observer, because a single run of $\mathcal{P}$ only requires performing a computation on $m'n/m$ bits of the file M. On the other hand, a trivial example of a PDO scheme that has a $(1, 1)$-observer would be one in which S must give the entire file M to C.

1.2 A basic PDO scheme

In this paper, we will be investigating certain types of PDO schemes in the *random oracle model*. Our basic scheme is very simple; it is based on S returning a response of the form $h(M \| c)$, given a random challenge c, where h is a random oracle. We follow the usual random oracle model as defined in [2], where h is regarded as a function that takes a binary string of arbitrary length as input and produces a completely random d-bit output. We call this output the *message digest* or *hash value*. An important property of random oracles is that any hash value $h(x)$ is completely independent of all the other hash values $h(x')$, $x' \neq x$.

This fact is used either implicitly or explicitly in our analyses of the protocols we discuss.

Here is some intuition regarding the schemes we analyze. Because h is a random oracle, the only way to compute $h(M \| c)$ is to supply h with all the bits in M and c. However, a possible complication is that a response $h(M \| c)$ could be *precomputed* ahead of time, instead of being computed *online* during the execution of the proving algorithm $\mathcal{P}$. Nevertheless, if we upper-bound the number of oracle accesses that are permitted in the pre-computation stage, then we can upper-bound the number of precomputed responses that $\mathcal{P}$ can utilize. If $\mathcal{P}$ is required to have a sufficiently high success probability and c is chosen from a large enough challenge space, then it must be the case that, with high probability, $h(M \| c)$ will be computed online during $\mathcal{P}$'s execution. In this case, the file M is necessarily stored in S's memory space immediately prior to the call $h(M \| c)$ that is made to the random oracle h. Therefore we will have a PDO scheme, according to our definition given above.

We note that we are not placing any restrictions on the computations performed by $\mathcal{P}$ or S. That is, we analyze our schemes in the setting of unconditional security. However, we will be bounding the storage available to S and/or the number of calls to the random oracle that S or $\mathcal{P}$ are allowed to make. Basically, it is impossible to prove anything meaningful without some restrictions of this type: if S can precompute and store the responses to all possible challenges, then it does not need to store the file M at all. In this case, the challenge-response protocol cannot assist in providing any guarantee to C. When we consider "reasonable" parameters for our schemes, we will allow large number of accesses to the random oracle, e.g., 2^{60} or 2^{80}, which would make the resulting security guarantees meaningful and believable.

1.3 PDO is a stronger notion than POR and PDP

We describe a simple example of a POR/PDP scheme that is not a PDO scheme. Let $M = (M_1, \ldots, M_m)$ be a message, where $\mathbb{F}_q$ is a finite field and $M_i \in \mathbb{F}_q$ for $1 \leq i \leq m$. A challenge consists of a random vector $c = (c_1, \ldots, c_m) \in (\mathbb{F}_q)^m$ and the corresponding response is $r = c \cdot M \in \mathbb{F}_q$. This is easily seen to be a POR scheme (e.g., see [11]). However, there exists $\mathcal{P}$ that can generate correct responses to challenges without explicitly storing or performing computations on the original file M; the existence of this $\mathcal{P}$ shows that we do not have a PDO scheme. When S is originally given the file M, it chooses a random $\gamma \in \mathbb{F}_q$ such that $\gamma \neq 1$. Then S computes $M' = \gamma M$. The values M' and γ are stored and M is discarded. Later, when $\mathcal{P}$ receives a challenge c, it computes the (correct)

response $r = \gamma^{-1}(c \cdot M')$. This computation does not use M, so no observer can obtain M simply by observing the online computations of $\mathcal{P}$.

1.4 Our contributions

Our main goal is to show that there are simple, practical schemes that satisfy our new notion of observability. We present a couple of PDO schemes and analyze them in different adversarial settings. We believe that simplicity of the schemes and the fact that they can be analyzed in a straightforward manner is an argument in favor of utilizing these types of schemes.

In Section 2, as a warm-up, we analyze our basic hash-based challenge-response scheme when the server S stores part of the file M along with some number of precomputed responses. We analyze the associated proving algorithm $\mathcal{P}$ and show, under reasonable assumptions regarding the storage available to S, that this scheme is indeed a practical PDO scheme.

In Section 3, we consider a more general adversarial situation, where S and $\mathcal{P}$ are only restricted in the number of accesses to the random oracle that they can make. Here we show that the scheme is still a PDO scheme in this stronger adversarial setting, and we provide a concrete bound for δ as a function of ϵ.

The schemes in Section 2 and Section 3 require hashing the entire file every time a response is computed. This may not be desirable. Therefore, in Section 4, we consider observers who are allowed to watch $\sigma > 1$ executions of the challenge-response protocol, obtaining part of the file each time the protocol is run. This permits schemes where responses can be computed more efficiently. In this section, we study another simple protocol and analyze its properties as a PDO scheme.

1.5 Previous work on POR/PDP and related concepts

In recent literature, there has been a considerable amount of work on various notions related to proof-of-storage on a cloud. All this research involves a client C who wants to verify whether S is storing its file properly or not. Here are a few papers on this general topic. Naor and Rothblum [10] studied primitives called *memory checking* and *sublinear authenticators* and gave lower bounds on the efficiency of these systems in Yao's simultaneous message model [14]. Juels and Kaliski [8] and Ateniese et al. [1] independently introduced the concept of proof-of-retrievability and proof-of-data-possession, respectively. Ateniese et al. [1] also introduced the idea of using *homomorphic authenticators* to reduce the communication complexity of the system. Shacham and Waters [13] showed that the scheme of Ateniese et al. can be transformed to a POR scheme by constructing an extractor that extracts the file from the responses of the server to the chal-

lenges. Bowers, Juels, and Oprea [3] used error-correcting codes, in particular the idea of an "outer" and an "inner" code (in much the same vein as concatenated codes), to get a good balance between the server storage overhead and computational overhead in responding to the audits. Dodis, Vadhan, and Wichs [4] used hardness amplification and error-correcting codes to propose a more efficient POR scheme, with extraction performed through list decoding and an almost-universal hash function. Paterson, Stinson, and Upadhyay [11] studied POR schemes in the setting of unconditional security, as well as their connections to error-correcting codes.

A related line of research investigates the question of *storage enforcement*, wherein, if the server passes the challenge-response protocol, then the client is assured that the server has allotted the amount of storage that it committed to the client. This problem is studied, for example, in [5–7]. In these types of schemes, the prover does not prove directly that it is storing the actual file, but it proves that it has allotted sufficient resources to do so.

2 Analysis of the basic PDO scheme

We begin by presenting in Figure 1 our basic PDO scheme that hashes the file along with a random challenge provided by C. The server S constructs the proving algorithm $\mathcal{P}$ after it has been given a copy of the file M. The server S is also permitted to access the random oracle and use the results to help construct $\mathcal{P}$. For example, various responses of the form $h(M \,\|\, c)$ could be precomputed and stored in S's memory.

For future reference, we list the parameters of our scheme in Table 1. In this scheme, d and ℓ will typically be much smaller than n. As an illustrative example, we could take $\ell = 100$, $d = 256$, and $n = 2^{25}$.

Remark 2.1. We will be analyzing the scheme in Figure 1 in the random oracle model. Nevertheless, it might be of interest to consider how this scheme might be instantiated in practice, where one would replace the random oracle with a "real" hash function. We note that this would have to be done carefully in order to prevent certain types of attacks. For example, if h is an iterated hash function, then $h(M)$ can be precomputed and stored, and then $h(M \,\|\, c)$ can be computed from $h(M)$ and c. These types of issues are studied in detail in [12]. A good practical solution might be to use a *zipper hash* [9].

In this section, we give an exact combinatorial analysis of a specific type of adversary that we name $\mathsf{A}(t, s)$. This adversary incorporates a *storage-bounded* server S who has access to t bits of storage. This storage will be used to store part

Initialization: The client C and the server S are given access to a random oracle h that maps an arbitrary length input to a d-bit output. C gives a file M to S, where the length of M is n bits. S is permitted to access h some fixed number of times, and then S creates a proving algorithm $\mathcal{P}$.

Challenge: C chooses an ℓ-bit random challenge, c, and sends it to S.

Response: S uses $\mathcal{P}$ to compute the response $r = \mathcal{P}(c)$ and sends it to C.

Verification: C accepts the response r as valid if and only if $r = h(M \| c)$.

Figure 1. Basic random oracle PDO scheme.

Parameter	Meaning
t	size of S's storage
n	size of the file M
s	number of bits of M stored by S
d	size of a message digest (hash value)
ℓ	size of a challenge
s^*	number of accesses to the random oracle

Table 1. Parameters of a PDO scheme.

or all of the file M and/or some number of precomputed responses. Although not completely general, this type of adversary is a very natural and plausible one to consider.

It may be useful to distinguish adversaries according to their intent. Perhaps the adversary is just being *selfish* in that it wants to achieve a "reasonable" success probability without actually storing the whole file. The adversary $\mathsf{A}(t, s)$ would be selfish only if $t < n$. Alternatively, the adversary could be *malicious*, which means that he might be willing to devote a large amount of storage to precomputed responses in an attempt to deceive the client C who is expecting the server S to store the file. The adversary $\mathsf{A}(t, s)$ would be considered to be malicious only if $t \geq n$.

The adversary $\mathsf{A}(t, s)$ stores s bits of M, along with s^* precomputed responses obtained from s^* accesses to the random oracle. Clearly, $s \leq \min\{n, t\}$. Since each response has d bits, it must be the case that $s^* \leq \lfloor (t - s)/d \rfloor$. Additionally, there are 2^ℓ possible responses, so $s^* \leq 2^\ell$. We can assume without loss of generality that the adversary makes the largest possible number of accesses to the

random oracle. Therefore, it follows that

$$s^* = \min\left\{2^\ell, \left\lfloor \frac{t-s}{d} \right\rfloor\right\}$$

for the adversary $\mathsf{A}(t,s)$.

Remark 2.2. The data precomputed by $\mathsf{A}(t,s)$ can be thought of as a list of ordered pairs of the form $(c, h(M \,\|\, c))$. In our analysis, we are only counting the storage needed for the responses $h(M \,\|\, c)$, as various methods could be used to reduce or even eliminate the storage required for the challenges (i.e., the c values). For example, the stored responses might correspond to s^* consecutive values of c.

Given a challenge c, the proving algorithm $\mathcal{P}$ operates as follows:

(i) If the (correct) response $r = h(M \,\|\, c)$ has been precomputed and stored, then $\mathcal{P}$ outputs r.

(ii) Otherwise (when the response has not been precomputed and stored), $\mathcal{P}$ randomly guesses the $n - s$ non-stored bits of M, creating a file M^*. The adversary then computes $r = h(M^* \,\|\, c)$ and outputs r.

Note that $\mathcal{P}$ makes at most one (on-line) call to the random oracle each time the challenge-response protocol is run.

The case where $t \geq 2^\ell d$ is not very interesting. Here the adversary can precompute all 2^ℓ possible responses when $s = 0$. Therefore, the resulting proving algorithm has success probability equal to 1 and no bits of the file M are stored. Therefore, in what follows, we will assume that $t < 2^\ell d$, which will imply that

$$s^* = \left\lfloor \frac{t-s}{d} \right\rfloor. \tag{2.1}$$

Theorem 2.3. *Assume that $t < 2^\ell d$. Then the success probability of $\mathsf{A}(t,s)$ is*

$$p(t,s) = 1 - \left(1 - \frac{2^s}{2^n}\right)\left(1 - \frac{1}{2^d}\right)\left(1 - \frac{t-s}{2^\ell d}\right). \tag{2.2}$$

Proof. Let E_1 denote the event that $\mathsf{A}(t,s)$ has precomputed the response to a given random challenge. Let E_2 denote the event that $\mathcal{P}$ correctly guesses the $n - s$ non-stored bits of n. Finally, let E_3 denote the event that the on-line output of the random oracle on an incorrect input yields the correct response (i.e., $h(x) = h(M \,\|\, c)$ for a given $x \neq M \,\|\, c$). For $1 \leq i \leq 3$, let $p_i = \mathsf{Pr}[E_i]$. Clearly, from (2.1), we have

$$p_1 = \frac{s^*}{2^\ell} = \frac{t-s}{2^\ell d}.$$

Here, we are assuming for convenience that $(t-s)/d$ is an integer, so we can omit the "floor" computation. It is also clear that

$$p_2 = \frac{2^s}{2^n} \quad \text{and} \quad p_3 = \frac{1}{2^d}.$$

Now the success probability of $\mathsf{A}(t,s)$ is

$$p(t,s) = p_1 + (1-p_1)(p_2 + (1-p_2)p_3) = 1-(1-p_1)(1-p_2)(1-p_3), \quad (2.3)$$

which simplifies to give the desired result. □

Suppose we fix a value of $t < 2^\ell d$ and, for brevity, we denote $p(s) = p(t,s)$. We will analyze the behavior of $p(s)$ as a function of s. In order to do this, it is helpful to define

$$q(s) = (2^\ell d - t + s)(2^s - 2^n).$$

It is easy to see that

$$q(s) = C(p(s) - 1), \quad (2.4)$$

where C is a constant (independent of s). Using elementary calculus, we have

$$q'(s) = 2^s\big(1 + (2^\ell d - t + s)\ln 2\big) - 2^n. \quad (2.5)$$

From (2.5), we see that $q''(s) > 0$ for all $s > 0$. Therefore, from (2.4), we also have $p''(s) > 0$ for all $s > 0$, so the function $p(s)$ is "concave up".

First, let's consider the situation where $s = 0$. We have

$$q'(0) = \big(1 + (2^\ell d - t)\ln 2\big) - 2^n.$$

Under the reasonable assumption that $2^\ell d \ll 2^n$, we will have $q'(0) < 0$ and $p'(0) < 0$.

At this point, we have shown that $p'(0) < 0$ and $p'(s)$ is an increasing function of s. The maximum value of $p(s)$ must occur at one of the two endpoints (i.e., when $s = 0$ or when $s = \min\{n,t\}$). Therefore, there are two possibilities to consider:

(1) The maximum value of $p(s)$ is attained when $s = 0$.

(2) The maximum value of $p(s)$ is attained when $s = \min\{n,t\}$. Note that this case can happen only when $p'(\min\{n,t\}) > 0$.

In order to determine which case would apply to a given parameter situation, it is helpful to estimate the value $\tilde{s} > 0$ such that $p(0) = p(\tilde{s})$. From (2.2), we see that $p(0) = p(\tilde{s})$ if and only if

$$\Big(1 - \frac{1}{2^n}\Big)\Big(1 - \frac{t}{2^\ell d}\Big) = \Big(1 - \frac{2^{\tilde{s}}}{2^n}\Big)\Big(1 - \frac{t - \tilde{s}}{2^\ell d}\Big).$$

If we ignore terms that are products of two "small" numbers, then the equation to be solved for $\tilde{s}$ can be approximated as

$$\frac{t}{2^\ell d} + \frac{1}{2^n} \approx \frac{t - \tilde{s}}{2^\ell d} + \frac{2^{\tilde{s}}}{2^n}.$$

If $\tilde{s}$ is reasonably large, then we can ignore the term $1/2^n$. In this case, our equation can be rewritten as

$$\frac{2^{\tilde{s}}}{2^n} \approx \frac{\tilde{s}}{2^\ell d},$$

which is equivalent to

$$\tilde{s} + \ell + \log_2 d \approx \log_2 \tilde{s} + n.$$

Ignoring the term $\log_2 \tilde{s}$, we finally obtain the estimate $\tilde{s} \approx s'$, where

$$s' = n - \ell - \log_2 d. \tag{2.6}$$

We see that $s' > 0$ because we assumed that $2^\ell d < n$. Now, in order for case (2) to occur, we need $s' \le \min\{n, t\}$. It is clear that $s' < n$. However, it may or may not be the case that $s' \le t$.

The two possible cases mentioned above will be distinguished by whether $s' \le t$ or $s' > t$. Roughly speaking, we have the following possibilities:

(i) If $t < s'$, then $p(s) \le p(0)$ for $0 < s < \min\{n, t\}$.

(ii) If $t \ge s'$, then $p(s) > p(0)$ for $s' < s < \min\{n, t\}$.

The second case is of course the more interesting one. This is because, in the first case, the adversary is better off storing nothing.

The above discussion can be rephrased by saying that, if the adversary's success probability exceeds $p(0)$, then $t \ge s'$, $s \ge s'$ and $s \le t$. In this situation, the adversary must be storing all of the file M, except for $\ell + \log_2 d$ (or fewer) bits.

We present some numerical examples.

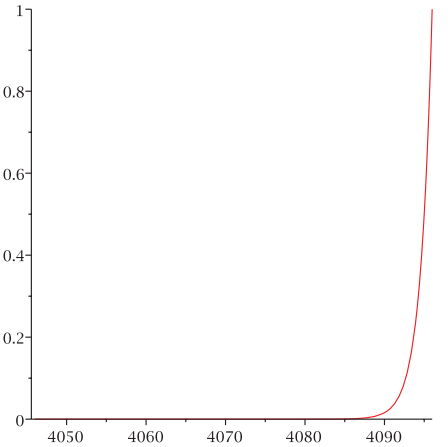

Figure 2. Success probability as a function of s when $n = 4\,096$, $\ell = 50$ and $d = 64$.

Example 2.4. Suppose $n = 2^{12} = 4\,096$, $\ell = 50$ and $d = 64$. Then $s' = 4\,040$. The exact value for $\tilde{s}$ is $\tilde{s} = 4\,041$, so the estimated value s' from (2.6) is very accurate. Suppose the adversary's storage is $t = s'$ bits (this would be a selfish adversary). Then $p(0) = 5.6 \times 10^{-14}$. Therefore, if the adversary's success probability exceeds 5.6×10^{-14}, it must be the case that the adversary is storing at least 4 041 of the 4 096 bits of M (i.e., all but 55 bits). Figure 2 presents a graph of $p(s)$ as a function of s. Note that $p(s)$ is very close to 0 until s is almost n, and then $p(s)$ increases very quickly to 1. This behavior is typical of most "reasonable" parameter situations.

Example 2.5. Suppose $n = 2^{25} = 33\,554\,432$, $\ell = 100$ and $d = 256$. Then $s' = 33\,554\,324$. The exact value for $\tilde{s}$ is $\tilde{s} = 33\,554\,329$, so the estimated value s' from (2.6) is very accurate. Suppose the (selfish) adversary's storage is $t = s'$ bits. Then $p(0) = 1.03 \times 10^{-25}$. Therefore, if the adversary's success probability exceeds 1.03×10^{-25}, it must be the case that the adversary is storing at least 33 554 329 of the 33 554 432 bits of M (i.e., all but 103 bits).

Now we consider how to construct an observer $\mathcal{O}(t, s)$ for the adversary $\mathsf{A}(t, s)$ and we analyze the success probability of the observer we construct. $\mathcal{O}(t, s)$ will simply copy any value M where the output of $\mathcal{P}$ is obtained by an online evaluation of $h(M \| c)$, where c is the challenge. Suppose we define events E_i and probabilities p_i $(1 \leq i \leq 3)$ as in the proof of Theorem 2.3. It is clear that $\mathcal{O}(t, s)$ outputs the correct file M whenever E_2 holds and E_1 does not hold. Therefore the

success probability of $\mathcal{O}(t,s)$ is

$$(1-p_1)p_2 = \Big(1 - \frac{t-s}{2^\ell d}\Big)2^{s-n}.$$

If the success probability of $\mathcal{O}(t,s)$ is denoted by ϵ, then, from (2.3), we can compute

$$\epsilon = p(t,s) - p_1 - (1-p_1)(1-p_2)p_3.$$

It is clear that

$$\epsilon > p(t,s) - p_1 - p_3. \tag{2.7}$$

For most "reasonable" parameter values, p_1 and p_3 are very small, and hence ϵ is very close to $p(t,s)$.

2.1 A variation

One possible variation/generalization of the adversary $\mathsf{A}(t,s)$ would be to consider storing *partial* precomputed information about responses rather than "all or nothing". However, it is not hard to see that this will not increase the adversary's success probability. Suppose that an adversary A stores d_1 bits of the response $h(M \,\|\, c_1)$ and d_2 bits of the response $h(M \,\|\, c_2)$, where $0 < d_1 \le d_2 < d$. Consider the modified adversary A' who stores $d_1 - 1$ bits of the response $h(M \,\|\, c_1)$ and $d_2 + 1$ bits of the response $h(M \,\|\, c_2)$. These two adversaries have the same storage requirement. However, we show below that the success probability of A' is at least as large as the success probability of A. Iterating this process enough times, we can transform the adversary A into one that does not store any "partial" responses.

Here is the proof of the claim: The probability of A guessing the response is 2^{-d+d_1} when the challenge is c_1, and 2^{-d+d_2} when the challenge is c_2. For the adversary A', the probabilities are 2^{-d+d_1-1} when the challenge is c_1, and 2^{-d+d_2+1} when the challenge is c_2. The claim is proven by observing that

$$2^{-d+d_1} + 2^{-d+d_2} \le 2^{-d+d_1-1} + 2^{-d+d_2+1}$$

when $d_1 \le d_2$.

2.2 Discussion

We conclude this section by discussing how the results presented in this section should be interpreted. Theorem 2.3 gives an exact success probability of a particular type of bounded storage adversary in terms of different parameters. If parameters of the scheme are chosen in a suitable way, this probability is close to 1 only

when $\tilde{s}$, the number of bits of the file stored properly by the server S, is very close to the actual size of the file. Therefore, if the adversary succeeds in the challenge-response protocol with a probability close to 1, then this implies that it has stored almost all of the file. The construction of the observer and equation (2.7) then guarantee that $\mathcal{O}$ outputs almost all of the file with probability very close to the success probability of the adversary.

3 A general bound

In this section, we consider a much more general adversary than the one studied in Section 2. We consider an arbitrary proving algorithm $\mathcal{P}$ for which $\mathsf{succ}(\mathcal{P}) \geq \delta$. Instead of bounding the storage utilized by S (as we did in the previous section), we upper-bound the number of calls to the random oracle that the adversary is allowed to make. We denote the maximum allowed number of random oracle calls by s^* and we permit S to store all the responses to these calls. We can assume without loss of generality that $s^* \leq 2^\ell$, since there are only 2^ℓ challenges $h(M \,\|\, c)$ that are pertinent to the challenge-response protocol. We refer to this adversary as a (δ, s^*)-*adversary*. The proving algorithm $\mathcal{P}$ can be either a deterministic or a randomized algorithm.

We want to quantify observability as a function of the success probability of $\mathcal{P}$. In order to do this, it is helpful to define some notation. First, we define

$$S_P = \big\{c : h(M \,\|\, c) \text{ has been precomputed}\big\}.$$

Observe that $|S_P| \leq s^*$.

Second, for $c \notin S_P$, let p_c denote the probability that $\mathcal{P}$ computes $h(M \,\|\, c)$ online, in response to the challenge c. It is possible that a given value $p_c \neq 0$ or 1, because $\mathcal{P}$ is permitted to be a randomized algorithm.

The next theorem upper-bounds the success probability of $\mathcal{P}$ in terms of s^* and the p_c values.

Theorem 3.1. *Suppose that $\mathcal{P}$ is a (δ, s^*)-adversary for a challenge-response protocol. Then*

$$2^\ell \delta \leq \Big(1 - \frac{1}{2^d}\Big)\Big(s^* + \sum_{c \notin S_p} p_c\Big) + 2^{\ell - d}. \tag{3.1}$$

Proof. There are $|S_P|$ values of c for which $h(M \,\|\, c)$ has been precomputed. For a challenge $c \notin S_P$, the response $h(M \,\|\, c)$ is computed online with probability p_c. If $c \notin S_P$, then with probability $1 - p_c$, the response $h(M \,\|\, c)$ is not computed

online and in this situation the probability that the response given by $\mathcal{P}$ is correct is 2^{-d}. It follows that

$$\mathsf{succ}(\mathcal{P}) \leq \frac{|S_P|}{2^\ell} + \frac{1}{2^\ell} \sum_{c \notin S_p} \big(p_c + (1 - p_c)2^{-d}\big).$$

After some simplification, the right-hand side of the above inequality can be rewritten as

$$\frac{1}{2^\ell}\Big(1 - \frac{1}{2^d}\Big)\Big(|S_P| + \sum_{c \notin S_p} p_c\Big) + 2^{-d}.$$

Using the fact that $|S_P| \leq s^*$, we obtain

$$\mathsf{succ}(\mathcal{P}) \leq \frac{1}{2^\ell}\Big(1 - \frac{1}{2^d}\Big)\Big(s^* + \sum_{c \notin S_p} p_c\Big) + 2^{-d}.$$

Since $\mathsf{succ}(\mathcal{P}) \geq \delta$, the inequality (3.1) follows. □

Next, we consider how to construct an observer $\mathcal{O}$ and we relate the success probability of $\mathcal{O}$ to the success probability of $\mathcal{P}$. The basic idea is to copy any value M such that the output of $\mathcal{P}$ is obtained by an online evaluation of $h(M \,\|\, c)$, where c is the challenge. However, a possible complication would be arise if $h(M \,\|\, c) = h(M' \,\|\, c)$ for some $M' \neq M$ and for some c. For example, perhaps $\mathcal{P}$ computes $h(M \,\|\, c)$, and then it manages to find $M' \neq M$ such that $h(M \,\|\, c) = h(M' \,\|\, c)$. It might not be obvious to $\mathcal{O}$ whether to copy M or M'.

One possible way to address this issue would be restrict relevant parameters of the scheme so that it is very unlikely that *any* collision will ever be encountered by the adversary. In general, this will follow from the birthday paradox provided that

$$s^* \ll 2^{d/2}. \tag{3.2}$$

In the following analysis, we will assume that (3.2) holds and hence we can assume that the adversary is not able to find any $M \neq M'$ such that $h(M \,\|\, c) = h(M' \,\|\, c)$ for some c.

Theorem 3.2. *Suppose that* A *is a* (δ, s^*)*-adversary for a challenge-response protocol for a* PDO *scheme, and suppose that* (3.2) *holds. Then there exists a* (δ, ϵ)*-observer, where*

$$\epsilon \geq \frac{2^d \delta - 1}{2^d - 1} - \frac{s^*}{2^\ell}. \tag{3.3}$$

Proof. As described above, the observer $\mathcal{O}$ copies any value M such that the output of $\mathcal{P}$ is obtained by an online evaluation of $h(M \| c)$, where c is the challenge. Denote

$$p^* = \sum_{c \notin S_p} p_c.$$

Since (3.2) holds, it follows that any value M copied by $\mathcal{O}$ is correct (except with negligible probability). Therefore $\mathcal{O}$ is a (δ, ϵ)-observer, where $\epsilon = p^*/2^\ell$. Applying (3.1) from Theorem 3.1 and simplifying, it follows that

$$\epsilon = \frac{p^*}{2^\ell} \geq \frac{2^d\delta - 1}{2^d - 1} - \frac{s^*}{2^\ell}.$$ □

Let's look a bit more closely at the lower bound (3.3). For any "reasonable" values of the parameters, the first term is essentially equal to δ and the second term can be ignored. As an example, suppose we take $\ell = 100$ and $d = 256$. Even if the adversary precomputes $s^* = 2^{60}$ responses, the second term is at most 2^{-40}. When δ is somewhat larger than 2^{-40}, we see that ϵ is very close to δ.

4 Multiple runs of the challenge-response protocol

Our definition of an observer given in Section 1.1 requires that an ϵ fraction of the file M can be obtained from a *single run* of the challenge-response protocol. This is a strong requirement, obviously, and protocols satisfying this requirement must necessarily involve challenges that "depend on" an ϵ fraction (or more) of M. For ϵ close to 1, this could be problematic because a frequent goal of POR-type protocols is to only require responses that depend on a "small" part of the whole file M (this is important mainly for efficiency reasons). In this section, we give a relaxation of our definition of an observer that allows a prover to respond to the challenges more efficiently while preserving the same type of security guarantees considered in Section 2 and Section 3. Specifically, we allow $\mathcal{O}$ to watch some number $\sigma \geq 1$ of executions of the challenge-response protocol, possibly copying part of M from each run. After σ runs of the protocol, $\mathcal{O}$ should have copied an ϵ fraction of file M, on average.

More precisely, let $0 < \delta \leq 1$, let $0 < \epsilon \leq 1$ and let $\sigma \geq 1$ be an integer. We say that an $\mathcal{O}$ is a $(\delta, \epsilon, \sigma)$-*observer* provided that the expected fraction of M that $\mathcal{O}$ copies is at least ϵ whenever $\mathsf{succ}(\mathcal{P}) \geq \delta$, where $\mathcal{O}$ watches σ runs of the challenge-response protocol with random challenges chosen by C. In general, ϵ will increase as σ is increased.

We give an analysis of a simple scheme in this setting. Suppose that a challenge c consists of a randomly-chosen k-subset of $\{1, \ldots, m\}$, say $c = \{i_1, \ldots, i_k\}$,

where $i_1 < \cdots < i_k$. The correct response is $r = (M_{i_1}, \ldots, M_{i_k})$. This scheme has been called the "multiblock challenge scheme" and a detailed analysis of it, from the point of view of coding theory, can be found in [11].

Theorem 4.1. *For any integer $\sigma \geq 1$, there is a $(\delta, \epsilon, \sigma)$-observer for the multiblock challenge scheme, where $\epsilon = \delta(1 - (1 - \frac{k}{m})^\sigma)$.*

Proof. Denote σ random challenges by $c^1, \ldots, c^\sigma$. The observer $\mathcal{O}$ will simply copy every message block supplied by $\mathcal{P}$ during the σ executions of the challenge-response protocol. It is possible that $\mathcal{O}$ observes a message block M_i and later observes a possibly different version of the same message block M_i, because some of the adversary's responses might be erroneous. In this case, only the "last" copy of M_i is retained. However, any *specific* copy of M_i obtained by $\mathcal{O}$ is correct with probability at least δ.

For $1 \leq j \leq m$, define a random variable $\mathbf{X}_j$, where

$$\mathbf{X}_j = \begin{cases} 1 & \text{if } j \in \bigcup_{i=1}^{\sigma} c^i, \\ 0 & \text{otherwise.} \end{cases}$$

Then define $\mathbf{X} = \sum_{j=1}^{m} \mathbf{X}_j$. It is easy to see that

$$E[\mathbf{X}_j] = \mathsf{Pr}[\mathbf{X}_j = 1] = 1 - \left(1 - \frac{k}{m}\right)^\sigma$$

for $1 \leq j \leq m$. By linearity of expectation, we have

$$E[\mathbf{X}] = m\left(1 - \left(1 - \frac{k}{m}\right)^\sigma\right).$$

This says that the expected fraction of message blocks that are queried is $1 - (1 - \frac{k}{m})^\sigma$. The probability that a specific response to a query is correct is at least δ. Therefore, our observer is a $(\delta, \epsilon, \sigma)$-observer, where $\epsilon = \delta(1 - (1 - \frac{k}{m})^\sigma)$. □

For fixed k and m, we have that $(1 - \frac{k}{m})^\sigma \approx e^{-k\sigma/m}$, so ϵ approaches δ exponentially quickly as a function of σ. This is exactly what we want to happen.

5 Summary and conclusion

We have introduced the new notion of proof-of-data-observability as a strengthening of the now-standard concepts of proof-of-recovery and proof-of-data-possession schemes for cloud storage. This new concept provides stronger guarantees relating to the server's behavior than the traditional notions do. We described

and analyzed some simple schemes for proof-of-data-observability in the random oracle model. We also proved some general necessary conditions for the existence of these schemes, and we studied an extension where the observer is allowed to accumulate the stored file over time, by observing a sequence of runs of the challenge-response protocol.

Bibliography

[1] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson and D. X. Song, Provable data possession at untrusted stores, in: *ACM Conference on Computer and Communications Security* (2007), 598–609.

[2] M. Bellare and P. Rogaway, Random oracles are practical: A paradigm for designing efficient protocols, in: *ACM Conference on Computer and Communications Security* (2003), 62–73.

[3] K. D. Bowers, A. Juels and A. Oprea, Proofs of retrievability: theory and implementation, in: *Proceedings of the First ACM Cloud Computing Security Workshop* (2009), 43–54.

[4] Y. Dodis, S. Vadhan and D. Wichs, Proofs of retrievability via hardness amplification, in: *Theory of Cryptography* (TCC 2009), Lecture Notes in Comput. Sci. 5444, Springer, Berlin (2009), 109–127.

[5] P. Golle, S. Jarecki and I. Mironov, Cryptographic primitives enforcing communication and storage complexity, in: *Financial Cryptography* (FC 2002), Lecture Notes in Comput. Sci. 2357, Springer, Berlin (2002), 120–135.

[6] M. I. Husain, S. Ko, A. Rudra and S. Uurtamo, Storage enforcement with Kolmogorov complexity and list decoding, preprint (2011), `http://arxiv.org/abs/1104.3025`.

[7] M. I. Husain, S. Ko, A. Rudra and S. Uurtamo, Almost universal hash families are also storage enforcing, preprint (2012), `http://arxiv.org/abs/1205.1462`.

[8] A. Juels and B. S. Kaliski Jr., PORs: Proofs of retrievability for large files, in: *ACM Conference on Computer and Communications Security* (2007), 584–597.

[9] M. Liskov, Constructing an ideal hash function from weak ideal compression functions, in: *Selected Areas in Cryptography* (SAC 2006), Lecture Notes in Comput. Sci. 4356, Springer, Berlin (2007), 358–375.

[10] M. Naor and G. N. Rothblum, The complexity of online memory checking, *J. ACM* **56** (2009), no. 1, article 2.

[11] M. B. Paterson, D. R. Stinson and J. Upadhyay, A coding theory foundation for the analysis of general unconditionally secure proof-of-retrievability schemes for cloud storage, *J. Math. Cryptol.* **7** (2013), 183–216.

[12] T. Ristenpart, H. Shacham and T. Shrimpton, Careful with composition: Limitations of the indifferentiability framework, in: *Advances in Cryptology* (EUROCRYPT 2011), Lecture Notes in Comput. Sci. 6632, Springer, Berlin (2011), 487–506.

[13] H. Shacham and B. Waters, Compact proofs of retrievability, in: *Advances in Cryptology* (ASIACRYPT 2008), Lecture Notes in Comput. Sci. 5350, Springer, Berlin (2008), 90–107.

[14] A. C.-C. Yao, Some complexity questions related to distributive computing (preliminary report), in: *Symposium on the Theory of Computing* (STOC), ACM Press, New York (1979), 209–213.

Received September 27, 2013; revised November 24, 2013; accepted January 8, 2014.

Author information

Douglas R. Stinson, David R. Cheriton School of Computer Science,
University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada.
E-mail: dstinson@uwaterloo.ca

Jalaj Upadhyay, David R. Cheriton School of Computer Science,
University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada.
E-mail: jkupadhy@cs.uwaterloo.ca