Remote Integrity Check without the Original

No Author Given

No Institute Given

Abstract. Bob promises to keep a large file $\mathbf{x}$ for Alice. The irresponsible Bob may throw away portion of $\mathbf{x}$ or ask another unreliable party to keep it. To prevent that, periodically, Alice wants to remotely check that Bob indeed has $\mathbf{x}$ readily available for reading. If Alice has the original $\mathbf{x}$, this can be easily checked by querying Bob. We consider the more challenging case where Alice does not has the original. This problem includes a component on lossy compression that is not present in typical requirement of integrity. We propose a formulation capturing the notion of an irresponsible Bob who is able to compress $\mathbf{x}$ with respect to the verification. We also propose two schemes and analyze their security.
Keywords: Remote Integrity Check, Private Information Retrieval, Remote File System, Memory Integrity Verification.

1 Introduction

Alice (the *owner*) has a large file $\mathbf{x}$ which she does not want to keep in her local storage. Alice may or may not need this file in the future. However, whenever she needs it, she wants it readily available. Bob (the *server*), who is connected to Alice through a low bandwidth network, promises to keep the file for Alice. However, the irresponsible Bob may discard portion of $\mathbf{x}$ as he is taking the risk that Alice may not need it. The irresponsible Bob may also temporarily move the file to a slower storage device. Let us call such an irresponsible entity a *cheater*. To prevent cheating, periodically, Alice wants to remotely check that Bob indeed has $\mathbf{x}$ readily available for reading. If Alice has the original $\mathbf{x}$, this can be easily checked by querying Bob, for example, by asking Bob to compute a keyed hash. However, Alice does not has the original and thus can not verify Bob's reply. In addition, since the checks are done frequently, after logging sufficient number of queries, Bob may be able to discard the file and yet able to answer future queries by using the logged data. These technical challenges leads to the problem studied in this paper: *how to remotely check the integrity and presence of a file, without any information of the original in the local storage.* Protocols that solve the problems could be useful in remote file system [9, 10, 3], peer-to-peer file backup [4] and memory integrity verification [13].

Beside the security requirement, there are a few efficiency considerations. (1) Communication bits required for the verification should be low. (2) As we shall see later, some secure schemes use large number of keys. Although technically, such keys do not contains any information of the original file and thus fit into the requirement of not keeping the original, in practice, the total key

size should be significantly smaller than the file. (3) Since the file is large, the verification algorithm has to be fast, for example quadratic algorithm is not acceptable. Preferably, the verification can be done without reading the whole file. (4) Additional storage at the server, beside the original file, should also be small.

An unsatisfactory scheme. To illustrate the problem, let us look at an unsatisfactory scheme. Both Alice and Bob agree on a keyed hash function. Alice keeps some secret keys, $s_1, s_2, \ldots$, and the hashed values, $h_1, h_2, \ldots$, of the respectively secret keys on the file $\mathbf{x}$. During each checking, Alice reveals a secret key and Bob must send the corresponding hashed value to Alice. Alice verifies whether Bob is a cheater by matching the value she received with h_i's. This method is unsatisfactory because a secret key is revealed during each check. When Bob thinks that he has seen and remembered sufficient keys, he can discard $\mathbf{x}$ and just keeps those s_i's and h_i's that have appeared. Furthermore, for certain known keyed hash, Bob is able to extract small number of bits from $\mathbf{x}$ that are sufficient to carry out the computations. For example, many popular keyed hash functions follow the hash-and-sign framework where the data are first hashed (without using the key), and then signed with the key. For these schemes, a cheater can simply keep the hashed value and discard $\mathbf{x}$.

Three approaches. Here are a few possible approaches to check against cheaters. The first approach can be viewed as a form of secure mobile code[1, 12] or authentication of out-sourced data[5]. Each bit x_i of the file $\mathbf{x}$ is tagged with an additional value t_i that are to be stored by Bob. During verification, Alice sends a random key s to Bob and Bob must reply with the $h_s(\mathbf{x})$ and a proof computed from the tags. Although Alice does not know the actual hashed value, she can use the proof to verified that the computations are carried out correctly.

Perhaps a more interesting approach employs a *trap-door compression*. With the private key d, Alice is able to compress the file $\mathbf{x}$ to obtain a compressed data $\mathtt{comp}_d(\mathbf{x})$. Alice has an algorithm $\mathcal{H}$ that operates in the compressed domain and able to compute the keyed hash from $\mathtt{comp}_d(\mathbf{x})$. That is,

$$\mathcal{H}(s, \mathtt{comp}_d(\mathbf{x})) = h_s(\mathbf{x}), \quad \text{for all } s$$

On the other hand, without the private key, Bob is unable to compress $\mathbf{x}$ with respect to the keyed hash $h_s(\mathbf{x})$ and thus forced to keep the whole file. Note that Alice is required to remember $\mathtt{comp}_d(\mathbf{x})$. In order not to store any information in the local storage, Alice can choose another key, encrypt $\mathtt{comp}_d(\mathbf{x})$, and send it to Bob. During verification, Bob is supposed to send back the ciphertext. Since data size of $\mathtt{comp}_d(\mathbf{x})$ is small, bandwidth required is not significant. In this approach, Bob only need to provide small amount of additional storage compared to the first approach.

Another approach, which we do not explore in this paper, is to employ a generalized form of Private Information Retrieval (PIR)[2, 8]. Alice keeps a secret key s and the corresponding hashed $h_s(\mathbf{x})$. During verification, PIR is carried out where the query is asked for the value of the keyed hashed. After completion

of the protocol, Alice obtains the hashed value and can verify whether Bob's reply is valid. Yet, by the properties of PIR, Bob still does not know s. We do not explore this approach since it is not clear how to extend PIR to support computation of hash, and typically the computing cost in PIR is high.

The security requirement of our problem has a "compressibility" component that is not present in the usual application on integrity. A naive cheater, based on a lossy-compressed file, may try to guess the most likely original and then carries out the verification protocol. However, it is not necessary to have the original in order to pass the verification. So, alternatively, the cheater may also exploit the weakness in the verification protocol. Since we can not prevent the cheater to guess the original from the compressed file, we should focus on the design of the verification protocol. Thus, we formulate a measure that quantifies the advantage the cheater has in evading the verification check, over what the best possible algorithm (without restriction on its running time) can do in guessing the original.

Outlines and Contributions. In this paper, we give a formulation of a security requirement (Section 2) that captures the notion of a cheater who has an advantage in suppressing the ambiguity introduced by compression. For illustration, we analyze two unsatisfactory simple schemes by giving cheaters with significant advantage (Section 3). We next propose two more schemes (Scheme 3 & 4). Scheme 3 follows the first approach mentioned above. We prove that Scheme 3 is vigilant to any cheater (Section 4) assuming an underlying hash function satisfies a property and factorization is difficult. This scheme requires small number of communication bits, but large number of keys to be stored in the local storage. Probably the size of the keys can be reduced by using a secure random number sequence. However further analysis is required. It is possible to modify Scheme 3 to achieve speedup by using only a fraction of the file during verification. Another drawback of the scheme is that it requires linear (with respect to the file size) additional storage at the server. Scheme 4 follows the second approach mentioned above (Section 5). This scheme is very simple and efficient. It require small amount of additional storage and communication bits during verification. However, we are only able to give a partial proof of its security.

2 Problem Formulation

2.1 Overview

Essentially, a cheater has a many-to-one compression that maps a few files to a same object. The cheater may has access to auxiliary information, for example previous communications and public key of the owner, to help him compressing the file. After a file is compressed, the cheater does not know the original and thus unable to carry out the verification process. The cheater can try to predict what is the most likely original from the compressed file, and then carries out the protocol accordingly. However, it is not necessary to have the original in

order to evade detection. For instance, the verification protocol may only ask for a hashed value of the original. Hence, a cheater may also exploit properties of the verification protocol. As we can not prevent the cheater to guess the original, we focus on the protocol. If the cheater, when given only the compressed data, is able to pass the verification with a probability that is better than the probability that he can successfully predict the original, we say that this cheater has an advantage. Our security goal is to design an owner so that no cheater has a constant advantage (*vigilant*) or no cheater has non-negligible advantage (*strongly vigilant*), over its verification protocol.

2.2 Definitions & Notations

The two entities involved are the *owner* (Alice) and the *server* (Bob). When the server intends to cheat, we call him a *cheater*. As mentioned in the introduction, the owner has a file $\mathbf{x} = x_1 x_2 \dots x_m$ where x_i is the i-th bit and thus the size of $\mathbf{x}$ is m. Let $\mathcal{X}$ be the collections of all files, and $\mathcal{X}_m$ be the collection of files with size m.

There are two phases in the remote integrity check. During the *setup* phase, the owner chooses a sequence of secrets, $\mathbf{s} = (s_1, \dots)$. Let k be the total size of the secrets, which is the security parameter in our main result, and let U_k be the random variable of the secret which is uniformly distributed among all bits strings of size k. During the *verification* phase, the owner interacts with the server and decides whether the server is a cheater.

Compression. The goal of a cheater is to "compress" the file $\mathbf{x}$. We view a compression:

$$\texttt{comp} : \mathcal{X} \times \mathcal{Y} \to \mathcal{C}$$

as a deterministic function, where $\mathcal{Y}$ is the collection of *auxiliary* data, and $\mathcal{C}$ is the compressed domain, and we call an element in $\mathcal{C}$ the compressed data.

The auxiliary data can be the additional data sent by the owner, or history of previous interactions. Although there may be a few algorithms involved in producing the data, eventually, the auxiliary data are extracted from the file $\mathbf{x}$ and the secret $\mathbf{s}$ by a probability algorithm. Thus, we write the auxiliary data as

$$f(\mathbf{x}, \mathbf{s}; b)$$

where f is a polynomial-time probabilistic function and b is the random bits utilized by the probabilistic algorithm. The actual range of f is of no interest in our analysis as long as the size of each compressed file is in $poly(k)$, where k is the size of the secret $\mathbf{s}$.

Given a compressed data, it could be the case that any algorithm is unable to confidently predict which is the original file among a set of candidates.

DEFINITION 1 *The* ***predictability*** *of a finite set $F \subset \mathcal{X}$ given the compression* comp*, the auxiliary function f and an integer k, is the average maximum*

likelihood,

$$\epsilon = \mathbb{E}_{\mathbf{x}\leftarrow X, \mathbf{s}\leftarrow U_k, b\leftarrow R}\left\{\max_{\mathbf{x}_0 \in F} \mathsf{Pr}[X = \mathbf{x}_0 \mid \mathtt{comp}(\mathbf{x}, f(\mathbf{x}, \mathbf{s}; b))]\right\}$$

where the expectation is taken over U_k, X which is uniformly distributed among elements in F, and the random choices R made by f.

The logarithm of the predictability $(-\log \epsilon)$ is also known as the average min-entropy [6]. The predictability can be viewed as the probability of success of the *minimum-error probability decoding* [7] in the studies of error correcting code. Given a received sequence transmitted over a noisy channel, the decoder guesses the original message while minimizing the probability of error. In our setting, it maximizes the probability of correct prediction.

Owner and Cheater. The owner $(V, tag, q(\cdot))$ is associated with two probabilistic algorithms V, tag and a size function $q : \mathbb{Z} \to \mathbb{Z}$ which is bounded by a polynomial. Given a security parameter $k \in \mathbb{Z}$, the owner chooses the k-bits secret $\mathbf{s}$ (recall that we write U_k as the random variable of the secret). The key is applicable for files of size not more than $q(k)$. For a file $\mathbf{x}$, the owner computes the tag $\mathbf{t} = tag(\mathbf{x}, \mathbf{s})$. The file $\mathbf{x}$ and the tag $\mathbf{t}$ are sent to the server during setup phase, whereas the secret $\mathbf{s}$ is kept in the local storage.

A *cheater* $(\mathtt{comp}, L, \widetilde{L})$ is a server who intends to discard partial information of $\mathbf{x}$ but pretends keeping the whole file. It is associated with three probabilistic polynomial-time algorithms: a compression $\mathtt{comp}$ that compresses the file $\mathbf{x}$, an algorithm L that interacts with the verifier V during verification, and a probe $\widetilde{L}$ who tries to extract secrets from the server via interactions. If the cheater does not has a probe, we write it as $(\mathtt{comp}, L, \cdot)$ and say that it is *memoryless*.

Let us write:

$$\langle\ L(c), V(\mathbf{s})\ \rangle$$

to be the output of V after the interaction with L, where c and $\mathbf{s}$ are the respective inputs.

We are more concerned about cheaters with probes. For clarity, we first give the definition for memoryless cheaters.

Definition 2 *A polynomial-time cheater $(L, \mathtt{comp}, \cdot)$ has a δ advantage over the owner $(V, tag, q(\cdot))$ if for infinitely many k,*

$$\mathsf{Pr}\left[\ \langle\ L\left(\mathtt{comp}(X, tag(X, U_k))\right), V(U_k)\ \rangle = accept\ \right] > \epsilon + \delta$$

where X is uniformly distributed among files in $\mathcal{X}_{q(k)}$ and ϵ is the predictability of the set $\mathcal{X}_{q(k)}$ with respect to $\mathtt{comp}$, $tag(\cdot, \cdot)$ and k. The probability is taken over X, U_k, and random choices made by V and $L, \mathtt{comp}, tag$.

We can substitute δ in Definition 2 by a function of k. Thus, we can say that a cheater has $1/q(\cdot)$ advantage. For owners, we define:

DEFINITION 3 *An owner is γ-**vigilant** over memoryless cheaters if, there is no cheater that has an advantage more than γ over the owner.*

We say that an owner $(V, tag, q_1(\cdot))$ is **strongly vigilant** *over memoryless cheaters if, for any positive polynomial $q_2(\cdot)$, there is no cheater that has $1/q_2(\cdot)$ advantage.*

In other words, an owner is strongly vigilant if no cheater can evade detection non-negligibly better than ϵ, which is the performance of the minimum probability error decoder in predicting the original.

A cheater may has access to an auxiliary information which consists of the communications between the owner V and the probe $\widetilde{L}$, and the internal states of the probe. Let $\texttt{view}_L^{V,tag}(\mathbf{x}, k)$ be the random variable of those information. We also assume that the probe $\widetilde{L}$ has seen the communication during the setup phase. Thus the view contains all information of $tag(\mathbf{x}, k)$ and $\mathbf{x}$. Since `view` is polynomial-time, implicitly the size of its output is also in $poly(k)$.

By replacing the auxiliary function in Definition 2 by the view, similarly, we define:

DEFINITION 4 *A polynomial-time cheater $(L, \texttt{comp}, \widetilde{L})$ has a δ advantage over the owner $(V, tag, q(\cdot))$ if for infinitely many k,*

$$\mathsf{Pr}\left[\left\langle L\left(\texttt{comp}(X, \texttt{view}_{\widetilde{L}}^{V,tag}(X, U_k))\right), V(U_k)\right\rangle = accept\right] > \epsilon + \delta$$

where X is uniformly distributed among files in $\mathcal{X}_{q(k)}$ and ϵ is the predictability of the set $\mathcal{X}_{q(k)}$ with respect to `comp`, `view` *and k. The probability is taken over X, U_k, and random choices made by V and L,* `comp`, `view`.

Similar to Definition 3, an owner $(V, tag, q_1(\cdot))$ is γ-vigilant if there is no cheater $(L, \texttt{comp}, \widetilde{L})$ that has γ-advantage, and is strongly vigilant if there is no cheater with non-negligible advantage.

2.3 Remarks

1. The compression `comp` could be a one-way function. The definition does not exclude a cheater who does not has an advantage and yet unable to "decompress" the compressed data. That is, the cheater keeps all information of the file as promised, but no polynomial-time algorithm is able to retrieve it. By doing so, the cheater actually gains nothing, for example, no extra storage space is freed. This is more of a playful and trouble making cheater, and not a concern in most applications. Nevertheless, for the proposed Scheme 3 to be introduced later, there is no such playful cheater. This is because from sufficient number of replies by the server, we can derive the original $\mathbf{x}$ by solving a system of linear equations. Thus, its `comp` can not be one-way.

2. If we have an γ-vigilant owner where γ is a constant, by repeating the verification protocol, it is not necessary that we will get a more vigilant owner.

3 Two Unsatisfactory Schemes.

To illustrate the formulation, let us consider two simple schemes that are unsatisfactory. For the first scheme, there is a memoryless cheater achieving significant advantage. The second scheme uses a keyed hash and is vigilant over memoryless cheater, assuming the hash functions satisfy a property. When the cheater has access to previous interactions, it is not vigilant.

Scheme 1: Checking the bits

Given a file $\mathbf{x}$ of size m, the owner randomly picks $s \in \{1, \ldots, m\}$, and a bit e. The tag to be stored in the server is $t = e$ **xor** x_s. During verification, the owner sends s to the server, and the server is supposed to reply with x_s **xor** t. The owner checks whether the reply is indeed the secret bit e.

Now, consider a cheater who simply discards one bit, say the first bit, from $\mathbf{x}$. The predictability of $\mathcal{X}_m$ is $\frac{1}{2}$. For any challenge s sent by the owner, if $s \neq 1$, the cheater can send the correct reply. If $s = 1$, he will send the value 0. The probability that owner accept is $\frac{m-1}{m} + \frac{1}{2m} = \frac{1}{2} + \frac{m-1}{2m}$. Hence the cheater has a $\frac{m-1}{2m}$ advantage.

Clearly, if a cheater can remember previous interactions, this scheme will fail miserably. After the first interaction, the cheater just has to remember his reply, i.e. x_s and discard the rest. So, the predictability of $\mathcal{X}_m$ is $2^{-(m-1)}$, and the cheater always succeeds. Hence, the cheater has a $1 - 2^{-(m-1)}$ advantage.

We may enhance the security of this scheme using existing PIR schemes [2, 8]. During verification, PIR is carried out. After completion of the PIR protocol, the owner will obtain x_s. Next, the server sends t to the owner and the owner can verify. However, the above simple cheater still achieves $\frac{m-1}{2m}$ advantage over this enhanced owner.

Scheme 2: Using Keyed Hash

- **Setup**: The owner picks a k-bits prime p and a keyed hash, that is a family of hash $h_i : \{0,1\}^m \to \mathbb{Z}_p$. The owner also picks two secrets $s, e \in \mathbb{Z}_p^*$ and computes the tag,

$$t = (e + h_s(\mathbf{x})) \mod p \tag{1}$$

  The file $\mathbf{x}$, p, and t are sent to the server.

- **Verification**: Owner sends s to the server and server replies with $(t - h_s(\mathbf{x}))$ mod p. The owner then checks whether the reply that she received is indeed e.

The size function $q(\cdot)$ can be any polynomial.

The tag t is essentially a ciphertext from the shift-cipher where the plaintext is $h_s(\mathbf{x})$ and the key is e. Consider the following property of the keyed hash.

PROPERTY 1 *Consider a function $q : \mathbb{Z} \to \mathbb{Z}$ and a family of hash functions $h^k_{\mathbf{s}}$ whose message space is $\{0,1\}^{q(k)}$ and both the key space and digest space is $\{0,1\}^k$. This family of hash functions satisfies the following property: For any positive constant γ, for all sufficiently large k, let X be the random variable of randomly chosen element from $\mathcal{X}_{q(k)}$ following some distribution (not necessary uniform), S be the random variable of the uniformly chosen secret, and $H = h_S(X)$. Suppose*

$$\epsilon = \max_x \mathsf{Pr}(X = x).$$

We have

$$\mathbb{E}_{\mathbf{s} \leftarrow S}[\max_h \mathsf{Pr}(H = h | S = \mathbf{s})] < \epsilon + \gamma.$$

This scheme is γ-vigilant over memoryless cheater, for any positive constant γ if the keyed hash satisfy Property 1. However, as expected, this scheme fails miserably when encountering cheaters who can view previous interactions.

LEMMA 2 *Scheme 2 is γ-vigilant over memoryless cheater for any constant γ, if the keyed hash satisfies Property 1.*

A possible candidate of the hash functions is $h_{\mathbf{s}}(\mathbf{x}) = \sum \mathbf{s}^i x_i \mod p$ where p is a k-bit prime. However, further analysis is required.

4 Scheme 3: Secure Remote Execution

The owner in this scheme assigns a tag to each bit during setup. During verification, the owner requests the value of a keyed hash and a "proof" that the computation is done honestly.

– **Setup**: Given the file $\mathbf{x}$, the owner picks a composite $n = pq$, where p,q are strong primes, and a secret element g with order $\lambda(n) = lcm(p-1, q-1)$. Owner also picks secret $\mathbf{s} = (s_1, \ldots, s_m)$ uniformly from[1] $\{0, 1, \ldots, n^2\}$. For each i, the owner computes

$$t_i = g^{s_i + x_i} \mod n.$$

The owner sends to the server the following:

$$n \text{ and } (x_i, t_i)_{i=1,\ldots,m}$$

[1] For ease in the proof of Theorem 3, we need this set to be large. It is more reasonable to choose from $\mathbb{Z}_{\lambda(n)}$, but our proof can not handle that. In the proof, we require a simulator who does not know $\lambda(n)$ and yet able to generates the keys. If the set is large, the distribution of $s_1 \mod \lambda(n)$ is statistically close to uniform.

– **Verification**: The owner picks a random $r \in \mathbb{Z}_n^*$ and sends it to the server. Let $r_i = r^i \mod n$ and

$$h_r(\mathbf{x}) = \sum_{i=1}^{m} x_i r_i \mod n.$$

The server computes

$$B = \sum_{i=1}^{m} x_i r_i, \quad \text{and} \quad A = \prod_{i=1}^{m} t_i^{r_i} \mod n$$

and sends both A and B to the owner. (Note that B is computed over $\mathbb{Z}$). Owner verifies using the secrets g and s_i. That is, the owner computes $\widetilde{A} = g^{B+h_r(\mathbf{s})} \mod n$ and accepts iff $A = \widetilde{A}$.

Note that a cheater can evade detection in two ways: The cheater can try to predict the hash value, i.e. $B = h_r(\mathbf{x})$, which is unlikely if the keyed hash satisfies Property 1. Alternatively, he can try to forge a pair (A, B). To forge a pair, the cheater may first try to guess g. However, this is very unlikely as there are many unknowns $s_1, s_2, \ldots, s_m, g$ and the posterior probability of g given the tags is statistically close to uniform. This forces the cheater to factorize n. Detail of the proof is in Appendix B.

THEOREM 3 *Scheme 3 is γ-vigilant over memoryless cheater for any positive constant γ, assuming factorization is difficult and Property 1 is satisfied.*

The interactions with verifier is perfect zero-knowledge, in the sense that there is a simulator, with knowledge of $\mathbf{x}$ and other information held by the server, can generate a sequence that is statistically same as the interactions. Thus, it is suffice to study the memoryless cheater. From Theorem 3, we have:

COROLLARY 4 *Scheme 3 is γ-vigilant over cheater for any positive constant γ, assuming factorization is difficult and Property 1 is satisfied.*

4.1 Efficiency Issues

1. The size function $q(\cdot)$ is actually sub-linear. Most of the secret bits are for the sequence s_i. In practice, the owner wants to keep very small number of secret bits. One possible solution is to generate s_i by a cryptographic secure pseudo random number generator `CSPRNG` where $s_i = \texttt{CSPRNG}(s, i)$, and keep only the seed s. It seems that Scheme 3 is still vigilant.

2. The communication costs during verification is low. Note that B is computed over $\mathbb{Z}$. Since it is the sum of products, if n consists of k_0 bits, B is at most $(2k_0)\lceil \log_2 m \rceil$ bits.

3. To further speedup computation at the server during verification, the owner can indicate a subset of the file to be involved. For example, the owner sends an additional seed u which generates bits sequence $u_1, u_2, \ldots, u_m$. The hashed value is

$$\widetilde{h}_{r,u}(\mathbf{x}) = \sum_{i=1}^{m} u_i r_i x_i \mod n.$$

For i's such that $u_i = 0$, the server can ignore it in the computation of A and thus achieving speedup. However, this owner is less vigilant.

4. For ease in presentation, each x_i in Scheme 3 is one bit. Vigilance remains the same if we allow the x_i's from $\mathbb{Z}_n$. By doing so, the size of the tags is about the same as the file $\mathbf{x}$.

5 Scheme 4: Trap-door Compression

Scheme 4 follows the second approach mentioned in the introduction.

- **Setup**: The owner picks a composite $n = pq$ where p, q are strong primes, and a random $s \in \mathbb{Z}_n$. By treating the whole file $\mathbf{x}$ as a large integer, the verifier computes the tag:

  $$t = (s + c) \mod n \quad \text{where} \quad c = \mathbf{x} \mod \phi(n).$$

  The owner sends $t, \mathbf{x}$ to the server.
- **Verification**: The owner randomly picks $r \in Z_n^*$, and sends it to the server. The server computes:

  $$a = r^x \bmod n$$

  Server sends a, t to the owner.
  The owner computes

  $$\widetilde{c} = (t - s) \mod n,$$
  $$\widetilde{a} = r^{\widetilde{c}} \mod n$$

  The owner accepts iff $a = \widetilde{a}$.

The tag t is the ciphertext from the shift cipher where the key is s and the plaintext is c. A server may compute a honestly, but send $(ar \mod n)$ and $(t+1 \mod n)$ to the owner. In this case, the owner will accept. Although the server "forges" the reply, he still needs to compute a honestly and thus not considered to be cheating.

Similar to Scheme 3, the interactions are perfect zero knowledge since there is a simulator that generates output that is statistically same as the interactions.

Trap-door compression. Consider the function

$$h_{r,n}(\mathbf{x}) = r^{\mathbf{x}} \mod n.$$

If order of r is $lcm(p-1, q-1)$, then each h_r is a collision resistant hash function, assuming factorizing n is difficult [11].

We can view the value n as public information and the factors (p, q) as the corresponding private key. With the private key, the server is able to lossy-compress $\mathbf{x}$ using the following polynomial-time function that reduces the size of the file $\mathbf{x}$.

$$C_{p,q}(\mathbf{x}) = \mathbf{x} \mod \phi(n).$$

Now, there is a polynomial-time function $\mathcal{H}$ such that for any $\mathbf{x}$ and r,

$$h_{r,n}(\mathbf{x}) = \mathcal{H}(n, r, C_{p,q}(\mathbf{x})).$$

Hence, we say that the server is able to compress $\mathbf{x}$ with respect to the hash functions. However, without the private key, any cheater should not be able to find a polynomial-time function $\widetilde{C}_n(\cdot)$ and another polynomial-time function $\mathcal{H}(\cdot)$ such that.

$$h_{r,n}(\mathbf{x}) = \widetilde{\mathcal{H}}(n, r, \widetilde{C}_n(\mathbf{x})).$$

In other words, using the public information n, cheater must not able to compress $\mathbf{x}$ with respect to the hash functions. Since with the private key, $\mathbf{x}$ can be compressed easily, the function $C_{p,q}(\cdot)$ can be viewed as a *trap-door compression.*

Unfortunately, we are unable to prove the vigilance of Scheme 4. Nevertheless, it is easy to show that, there is no cheater who uses a compression function $\widetilde{C}_n(\cdot)$ that is not one-way. If $\widetilde{C}_n$ is not one-way, then with non-negligible probability, we can find $\mathbf{x}$ and $\mathbf{y}$ s.t. $\widetilde{C}_n(\mathbf{x}) = \widetilde{C}_n(\mathbf{y})$. This leads to:

$$h_{r,n}(\mathbf{x}) = h_{r,n}(\mathbf{y})$$

which contradicts the fact that the hash function is collision-resistant.

6 Conclusions and Discussions

We have studied the problem of remote integrity check of a file without having any information of the original with the verifier. A security model is proposed to capture the notion of a cheater who has an advantage in suppressing the ambiguity introduced by compression. Two simple schemes are analysed to illustrate our security model. We also proposed two more schemes, one is proven to be secure under the assumption that the underlying hash functions satisfy a property and factorizataion is hard, and the other scheme is supported by a partial proof of security.

The main gap to be filled in the proof for Scheme 3 is the property of the hash functions. From the first glance, it seems that Property 1 can be easily satisfied. However, it is rather frustrating that we are unable to find a proof.

The reduction of the number of secret bits using a CSPRNG is probably not a major issue, although the proof may be tedious. There are many more issues to be studied. From practical aspect, it is interesting to combine Scheme 3 and 4 so that we can inherit advantage from both: computation speedup is possible for scheme 3, and scheme 4 essentially does not need additional storage at the server. In addition, it would be interesting to find out whether trap-door compression is possible.

References

1. J. Algesheimer, C. Cachin, C. Jan, and G. Karjoth. Cryptographic security for mobile code. In *Proc. of IEEE Symposium on Security and Privacy*, pages 2–11, 2001.
2. Y. C. Chang. Single database private information retrieval with logarithmic communication. In *9th Australasian Conference on Information Security and Privacy*, pages 50–61, 2004.
3. D. Clarke, B. Gassend, G. E. Suh, M. v. Dijk, and S. Devadas. Offline integrity checking of untrusted storage. Technical Report MIT-LCS-TR-871, MIT, 2002.
4. Landon P. Cox, Christopher D. Murray, and Brian D. Noble. Making backup cheap and easy. In *5th symposium on Operating systems design and implementation*, 2002.
5. P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine. Authentic data publication over the internet. In *14th IFIP 11.3 Working Conference in Database Security: Data and Applications Security*, pages 101–113, 2000.
6. Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Eurocrypt'04*, volume 3027 of *LNCS*, pages 523–540. Springer-Verlag, 2004.
7. Robert G. Gallager. *Information Theory and Reliable Communication*. John Wiley & Sons, 1968.
8. C. Gentry and Z. Ramzan. Single-database private information retrieval with constant communication rate. In *32nd International Colloquium on Automata, Languages and Programming*, pages 803–815, 2005.
9. E-.J. Goh, H. Shacham, N. Modadugu, and D. Boneh. Sirius: Securing remote untrusted storage. In *Tenth Network and Distributed System Security*, pages 131–145, 2003.
10. J. Li, M. Krohn, D. Mazieres, and D. Shasha. Secure untrusted data repository (sundr). In *6th Symposium on Operating Systems Design and Implementation*, pages 121–136, 2004.
11. R. L. Rivest. Public communication. `http://www.mit.edu:8080/bloom-picayune/crypto/13190`, May 2003.
12. T. Sander and F. Tschudin. Protecting mobile agents against malicious hosts. In *Mobile Agents and Security*, pages 44–60, 1998.
13. G. E. Suh, D. Clarke, B. Gasend, M. van Dijk, and S. Devadas. Efficient memory integrity verification and encryption for secure processors. In *36th annual IEEE/ACM International Symposium on Microarchitectu*, pages 339–350, 2003.

A Proof of Lemma 2

LEMMA 2 *If the keyed hash satisfies Property 1, then Scheme 2 is γ-vigilant over memoryless cheater for any positive constant γ.*

Proof:

Suppose that there is a memoryless cheater $(\mathtt{comp}, L, \cdot)$ who has γ advantage, for some positive constant γ. From the definition of cheater (Definition 2), for infinitely many k's, we have

$$\mathsf{Pr}\left[\ \langle\ L\left(C(X,U_k)\right), V(U_k)\ \rangle = \mathit{accept}\ \right] > \epsilon + \gamma. \tag{2}$$

where $C(\mathbf{x},\mathbf{k}) = \mathtt{comp}(\mathbf{x}, tag(x,\mathbf{k}))$, and the probability is taken over X, U_k as defined in Definition 2, and the random choices made by L and V during the interaction, in particular, r chosen by the verifier V. The predictability (Definition 1) is

$$\epsilon = \mathbb{E}_{\mathbf{x}\leftarrow X,\mathbf{k}\leftarrow U_k}\left\{\max_{\mathbf{x}_0\in F}\ \mathsf{Pr}[X=\mathbf{x}_0\ |C(\mathbf{x},\mathbf{k})]\right\} \tag{3}$$

The owner accepts when the hashed value is correct, which implies that the cheater is able to predict the hashed value with probability more than $\epsilon+\gamma$. Since the cheater can not do better than the minimum error probability decoder, from (2), we have:

$$\mathbb{E}_{\mathbf{x}\leftarrow X,\mathbf{k}\rightarrow U_k, r\leftarrow R}\left\{\max_h \mathsf{Pr}\left[H=h\mid C(\mathbf{x},\mathbf{k}), r\right]\right\} > \epsilon+\gamma. \tag{4}$$

where $H = h_r(\mathbf{x})$ and $C(\mathbf{x},\mathbf{k}) = \mathtt{comp}(\mathbf{x}, tag(\mathbf{x},\mathbf{k}))$.

Now, we consider Property 1. For any compressed file c_0, we treat $X|C=c_0$ as the random variable X(the notation is "overloaded" here) in Property 1. Hence, by the property, for any c_0, we have

$$\mathbb{E}_{r\leftarrow R}[\max_h \mathsf{Pr}(H=h|C=c_0, R=k)] < \epsilon+\gamma$$

Thus, it is impossible to have (4). □

B Proof of Theorem 3

THEOREM 3 *Scheme 3 is vigilant over memoryless cheater, assuming factorization is difficult, and the hash functions satisfy Property 1.*

Proof: (Sketch)

Suppose there is a cheater $(\mathtt{comp}, L, \cdot)$ that has advantage over Scheme 3 with size function $q(\cdot)$, We want to construct an algorithm that factorizes a composite with non-negligible probability by simulating the cheater.

Given a randomly chosen k-bit composite $n = pq$, carry out the following:

- *Simulating the owner:* Randomly choose a file $\mathbf{x}$ of size $m = q(k)$, $g \in \mathbb{Z}_n$, $s_1, \ldots, s_m$ from $\mathbb{Z}_{n^2}$. Next construct the tag.

– *Simulating the cheater:* Construct the compressed file c from the tag. Eventually, the compressed data is computed from $\mathbf{x}$, n and $\mathbf{r}$ which are the random bits used by the owner to derive s_i and g. Let use write

$$c = f(\mathbf{x}, \mathbf{r}, n).$$

– *Simulating the interactions:* Choose a random challenge $r \in \mathbb{Z}_n$ which is to be the challenge sent by the owner. If $gcd(r, n) > 1$, factor n and halt. Let (A, B) be the response of the cheater computed from the compressed data c and r .
Recall that the owner will accept iff $A \equiv g^{h_r(\mathbf{s})+B} \pmod{n}$. Let

$$\widetilde{A} = g^{h_r(\mathbf{s})+h_r(\mathbf{x})} \mod n$$

– *First Failing point:* If the owner does not accept, then halt.
– *Second Failing point:* If $A = \widetilde{A}$, that is $B = h_r(\mathbf{x})$ then halt.
– *Third Failing point:* If $A \neq \widetilde{A}$, then we can obtain a pair (d, e) where $e \equiv g^d \pmod{n}$. These are obtained by:

$$d = B - h_r(\mathbf{x})$$
$$e = A\widetilde{A}^{-1} \mod n$$

If $e \notin \{-1, 1\}$, then halt.
– *Factorize n:* If $e \in \{-1, 1\}$, we can easily obtain the 2 factors of n from d.

We want to show that the probability that the execution will reach the last step is non-negligible. If there is a γ-cheater, then there is a probability of $\epsilon + \gamma$ passing the First Failing point. Now, let us consider the Second Failing point. Recall that the unsatisfactory Scheme 2 is vigilant over memoryless cheater. If the hash satisfies Property 1, then for sufficiently large k, the probability that $B = h_r(\mathbf{x})$ is smaller than $\epsilon + \gamma/2$. Therefore, there is a constant probability that the Second Failing point is passed.

For the Third Failing point, note that if we are able to get a pair (d, e) where $e \notin \{-1, 1\}$, then information theoretic-wise, we can restrict g to a very small set of candidates. On the other hand, any primite root in the multiplicative group generated by g can give the tags, $t_1, t_2, \ldots$ So, we are unable to get the pair with probablity more than a constant.

In sum, the Third Failing point is passed with constant probability. Hence, we can factorize n.

□