Remote Integrity Check with Dishonest Storage Server

Ee-Chien Chang Jia Xu

School of Computing
National University of Singapore
{changec,xujia}@comp.nus.edu.sg

Abstract. We are interested in this problem: a verifier, with a small and reliable storage, wants to periodically check whether a remote server is keeping a large file $\mathbf{x}$. A dishonest server, by adapting the challenges and responses, tries to discard partial information of $\mathbf{x}$ and yet evades detection. Besides the security requirements, there are considerations on communication, storage size and computation time. Juels et al. [10] gave a security model for *Proof of Retrievability* ($\mathcal{POR}$) system. The model imposes a requirement that the original $\mathbf{x}$ can be recovered from multiple challenges-responses. Such requirement is not necessary in our problem. Hence, we propose an alternative security model for *Remote Integrity Check* ($\mathcal{RIC}$). We study a few schemes and analyze their efficiency and security. In particular, we prove the security of a proposed scheme HENC. This scheme can be deployed as a $\mathcal{POR}$ system and it also serves as an example of an effective $\mathcal{POR}$ system whose "extraction" is not verifiable. We also propose a combination of the RSA-based scheme by Filho et al. [7] and the ECC-based authenticator by Naor et al. [12], which achieves good asymptotic performance. This scheme is not a $\mathcal{POR}$ system and seems to be a secure $\mathcal{RIC}$. In-so-far, all schemes that have been proven secure can also be adopted as $\mathcal{POR}$ systems. This brings out the question of whether there are fundamental differences between the two models. To highlight the differences, we introduce a notion, *trap-door compression*, that captures a property on compressibility.

1 Introduction

Recently, there is growing interests in remote verification of storage server. Consider the scenario where Alice has a large file $\mathbf{x}$ which she wants to store in a peer-to-peer backup system, and she does not want to keep it locally. Bob, a peer node, promises to keep the file for Alice. However, Bob might discard portion of $\mathbf{x}$ to save storage, or temporarily move the file to a slower storage, hoping that Alice may not need it during the period. To prevent cheating, periodically, Alice wants to remotely check that Bob indeed has $\mathbf{x}$ readily available for reading without retrieving the whole $\mathbf{x}$.

Besides the security requirements, the scheme has to be efficient. There are a few resources to be considered: (1) The amount of communication bits required per verification should be small. (2) The storage at the verifier should be small. We want to limit it to a constant factor of κ, where κ is a security parameter sufficiently large for cryptography, for e.g. $\kappa = 1024$. (3) The size of the additional storage at the server should be small. Although it is desired to have $O(\kappa)$ additional storage, sublinear (w.r.t. the original file size) is also acceptable. (4) Finally, computation per verification should be low. To quantify the amount of computation, we measure the number of bits accessed from the storage.

Security model. Let us call the problem considered in this paper *Remote Integrity Check* ($\mathcal{RIC}$). Formulating the security requirement is tricky since the server is free to transform the data. Juels et al. [10] proposed a security model for *Proof Of Retrievability* ($\mathcal{POR}$) system. Roughly, a scheme is secure if, there is a polynomial time extractor, s.t. for any server that is able to pass the verification, the extractor can recover the original file by carrying out multiple verifications[1]. There is a subtle but crucial difference between $\mathcal{POR}$ system and the $\mathcal{RIC}$. Under $\mathcal{POR}$, it is required that the original can be recovered by interacting with the server. This requirement on recovery is not necessary in $\mathcal{RIC}$, where verification and recovery can be carried out in two different phases. For example, in the previous application of peer-2-peer backup, when Alice decides to recover the file $\mathbf{x}$, she can retrieve the whole $\mathbf{x}$ and then checks the integrity of $\mathbf{x}$ using the usual message authentication code or signature, without carrying out the verification protocol.

Without the recovery requirement, we may be able to design schemes with better performance. One possible candidate is the simple and yet interesting RSA-based scheme by Filho et al. [7]. In this scheme, the server's response is $r^{\mathbf{x}} \bmod n$, where $\mathbf{x}$ is the file treated as a single integer, r a randomly chosen challenge and n a composite. Since the responses only contain information of $\mathbf{x} \bmod \phi(n)$, thus it is impossible to recover $\mathbf{x}$ from multiple challenges-responses. Nevertheless, the RSA-based scheme seems able to detect a dishonest server who has discarded partial information. Another example is a scheme by Ateniese et al. [2]. Similarly, it is impossible to recover the original by interacting with their server. Thus it is inappropriate to prove the security of these schemes using security model of $\mathcal{POR}$, and a "weaker" security model for $\mathcal{RIC}$ is needed.

We propose a variant of security requirement where the extractor, instead of interacting with the server, has complete access to the server's storage. Two forms based on the computing power of the extractor are considered. We first consider extractor which is a maximum likelihood decoder and does not impose constrain on its computing time. If a scheme is secure under this setting, we call it weakly-secure. We also consider extractors that are probabilistic polynomial time algorithms. The security of a scheme is parameterized by (β, γ). Intuitively,

[1] To be more precise, the original file can be extracted by interacting with the server who is carry out the verification protocol. Note during the extraction phase, the extractor may generate challenges in a way different from that of the verifier. Whereas the server (prover), who is unable to distinguish whether the challenges are from the extractor or the verifier, is using the same algorithm as in verification phase.

a scheme is (β, γ)-secure (or weakly secure) if, for any server that can pass the verification with probability at least β, then there is an extractor who can recover the original from the verifier's and server's storages with probability at least γ.

In-so-far, all schemes that have been proven secure under $\mathcal{RIC}$ can also be employed as a secure $\mathcal{POR}$. This indicates some intriguing differences between $\mathcal{POR}$ and $\mathcal{RIC}$. To highlight this issue, we reformulate the model to a simple form which we call *trap-door compression*: Consider a keyed-hash family. With a secret key, the owner can lossily (that is, some information has been discarded) compress the file $\mathbf{x}$ to $\widetilde{\mathbf{x}}$, s.t. for any r, the hashed value $H(r, \mathbf{x})$ can be computed from $\widetilde{\mathbf{x}}$, r and the secret key. However, without the secret key, any dishonest server is unable to discard partial information and yet able to compute the hash. In other words, without the secret key, compression w.r.t. the keyed-hash family is computationally difficult. Note that from a trap-door compression, it is easy to build a $\mathcal{RIC}$ which is not a $\mathcal{POR}$ system. This property on "compressibility" could be of independent interest.

Proposed schemes. The error correcting code (ECC) based authenticator [4, 12] can be directly employed as a $\mathcal{RIC}$ and $\mathcal{POR}$ (we call this scheme `AUTH`). The scheme `AUTH` introduces redundancy into the file to achieve tradeoff between additional storage and the number of bits read per verification. Such generic technique is effective and hence we want to design schemes whereby the technique can be incorporated. On the other hand, the main drawback of `AUTH` is the large server's storage required. For example, it requires at least 4 times more storage space if a single verification achieves less than 0.5 false acceptance rate. To lower the false acceptance rate, multiple verifications can be made but that will incur more communication bits.

To reduce the communication bits, we can use a simple homomorphic MAC and an almost universal hash family. This scheme is also proposed by Shacham et al. [14] and this paper is an independent work. Let us call this scheme `HTAG`. Essentially, `HTAG` hashes and aggregates multiple challenges-responses into one, and thus reducing the number of communication bits to a constant factor of κ. However, there is no reduction in the storage size.

We next give a simple scheme `HENC` which requires sub-linear (w.r.t. the file size) additional storage but more communication bits. `HENC` sends a sequence of the form $(g^{\alpha r} \mod p), (g^{\alpha r^2} \mod p), \ldots$ during verification, where α and r are secrets. Using a bilinear map, the amount of communication bits can be reduced by square-root of the size of the original file. We can show that `HENC` is a weakly-secure $\mathcal{RIC}$. By using the Paillier cryptosystem, we can obtain a variant that is a secure $\mathcal{RIC}$. `HENC` can be used as $\mathcal{POR}$ system: to recover the file, the owner uses another algorithm to generate the queries. However, the response from the server cannot be verified. Hence, `HENC` also serves as an example of a $\mathcal{POR}$ system whose extraction is not verifiable [5]. We also propose `HYB`, a hybrid of `HENC` with `HTAG`, that further reduces communication bits.

Along another direction in improving `AUTH`, we incorporate a redactable signature [9] scheme to reduce the additional storage down to a constant factor of

κ. Let us call this scheme `REDACT`. During setting up, the original file $\mathbf{x}$ is encoded and expanded to $\mathbf{y}$ as in `AUTH`, and a redactable signature of $\mathbf{y}$ is obtained. However, the server only need to store the original $\mathbf{x}$ and the signature. When the verifier wants to know the i-th object in $\mathbf{y}$, the server derives $\mathbf{y}$ from $\mathbf{x}$, and computes the redactable signature for the requested object. It is not difficult to show that `REDACT` is a secure $\mathcal{RIC}$ and $\mathcal{POR}$. Clearly, the main disadvantage is the computation time, and it is not clear how to aggregate multiple responses to reduce communication bits.

Filho et al. [7] proposed a scheme based on a collision resistant hash, which is a candidate of trap-door compression function. Let us call it `RSAb`. It seems to be secure but there is no rigorous proof. `RSAb` consumes the same resource as `REDACT` and require intensive computation. Fortunately, we can exploit its homomorphic properties to trade-off the number of bits read with the storage size, while keeping communication cost unchanged. Let us call this extension `RSAh`. Among the schemes studied in this paper, `RSAh` achieves the best asymptotic performance.

Performance. Table 1 gives a summary on performance. A reasonable choice of the parameters is: $c = 0.2, w = 500$, $\ell = 1000$ and $\kappa = 1000$. Thus, if the file is 1Gbits, then for `AUTH`-(c, w), the total storage required at the server is roughly 2.4Gbits, and the server has to send at least 1Mbits during verification. In contrast, `HTAG`-(c, w) requires only 4000 communication bits although it still requires 2.4Gbits total storage. `HYB`-(c, w) and `REDACT`-(c, w, ℓ) require only roughly 1.2Gbits total storage. `RSAh`-(c, w, ℓ) reduces the total storage size to roughly 1.2Gbits and communication cost to 5000 bits.

Contribution. We propose a security model for $\mathcal{RIC}$. Unlike the previously known model for $\mathcal{POR}$, the extractor can access the server's storage. We propose a few schemes: `HTAG`, `HENC`, `HYB`, `REDACT` and `RSAh`. Shacham et al. [14] gave a scheme same as `HTAG` and this paper is an independent work. The performance of these schemes is summarized in Table 1. The scheme `HTAG`, `HENC`, `HYB` and `REDACT` can be shown to be secure under reasonable cryptographic assumptions. Interestingly, schemes that have been proven secure under $\mathcal{RIC}$ can also be deployed as $\mathcal{POR}$ systems. To highlight the difference between the two models, we introduce the notion of trap-door compression.

2 Related Work

This paper is motivated by applications in remote-backup and peer-to-peer backup ([1, 3, 11]). Peer-to-peer backup system requires a mechanism to maintain the availability and integrity of data stored in peer nodes. Li et al. [11] proposed to choose neighboring nodes based on the social relationships and relies on the principle that people are more likely cooperative with friends.

Recently, there is a growing interest in the cryptographic aspects of the problem. Perhaps Filho et al. [7] first studied the scenario where the verifier does

Scheme	Additional Storage	Bits accessed	Communication	Refer to
AUTH-(c, w) [12]	$(1 + 2c)m\kappa + O(\kappa)$	$2w\kappa$	$(1 + 2w)\kappa$	Section 4.1
HTAG-(c, w) [14]	$(1 + 2c)m\kappa + O(\kappa)$	$2w\kappa$	$O(\kappa)$	Section 4.2
HYB-(c, w, ℓ)	$(c + \frac{1+c}{\ell})m\kappa + O(\kappa)$	$O(w\kappa\ell)$	$O(\kappa\sqrt{\ell})$	Section 4.4
REDACT-(c, w, ℓ)	$(c + \frac{1+c}{\ell})m\kappa + O(\kappa)$	$O(w\kappa\ell)$	$O(w\kappa)$	Section 4.5
RSAh-(c, w, ℓ)	$(c + \frac{1+c}{\ell})m\kappa + O(\kappa)$	$O(w\kappa\ell)$	$O(\kappa)$	Section 4.6

Table 1. The size of the original file is $w\kappa$ bits. The file is expanded to a factor of $(1 + c)$ using ECC, grouped into blocks where each block contains ℓ elements, and w "requests" are made during a single verification. When $w = \kappa$ and $c = 1$, they are either $(2^{-\kappa}, 1 - negl(\kappa))$-secure or weakly secure. For RSAh, there is no formal proof of its security.

not have the original. They described two potential applications: *uncheatable data transfer* and *demonstrating data possession*, and proposed the RSA-based scheme. Juels et al. formulated the $\mathcal{POR}$ system and gave a security model [10]. They also proposed a sentinel-based method. However, the sentinel-based method can only support constant number of verifications. A refined security formulation is given in a recent technical paper [5]. The main difference of $\mathcal{POR}$ and $\mathcal{RIC}$ is in the requirement on file recovery. Ateniese et al. [2] gave a model for *Proof of Data Procession* system, and proposed a few schemes that provide tradeoff of computing time and storage size. These schemes can be viewed as an extension of the RSA-based scheme. Our scheme RSAh exploits similar idea, but is enhanced with ECC and in a simple form. Ateniese et al. [2] adopted the security model of $\mathcal{POR}$ and showed the security of the proposed schemes. However, it is inappropriate to apply $\mathcal{POR}$ security model since the original file cannot be recovered by interacting with the proposed server. In a recent technical paper, Shacham et al. [14] proposed a scheme which is essentially the same as the scheme HTAG independently given in this paper. The security model of $\mathcal{RIC}$ in this paper is based on notations and insight provided by Juels et al. [10] and an earlier manuscript [6].

Remote integrity checking is closely related to memory integrity verification [15, 4]. The notion of authenticator proposed by Naor et al. [12] is formulated for memory integrity check. Nevertheless, an authenticator can also be deployed as a $\mathcal{POR}$ and $\mathcal{RIC}$ system. In particular, the idea of introducing redundancy to tradeoff resources is useful in our problem. Under the authenticator model, the queries sent must be requests of values at particular memory locations. Such restriction is not present in our problem, where the verifier may request for some computation to be done on the file.

The trap-door compression may be related to a notion by Harnik et al. [8] on compressibility of $\mathcal{NP}$ decision problems. Consider a $\mathcal{NP}$ decision problem, they studied compression that preserves the solution to an instance rather than preserving the instance itself.

3 Formulations and Definitions

Our formulation is based on the $\mathcal{POR}$ model proposed by Juels et al. [10] and the manuscript [6]. Roughly, a scheme is (β, γ) secure if, for any adversary who can pass verification with probability at least β, then there is an extractor who can recover the original with probability at least γ. The main difference of our model from $\mathcal{POR}$ is the type of information accessed by the extractor and its ability. Under $\mathcal{POR}$, the extractor is probabilistic polynomial time (*PPT*) and it can interact with the server, and has access of the verifier's storage. Under $\mathcal{RIC}$, the extractor has access to both the verifier's and the server's storages. We consider two settings. Firstly, the extractor is a maximum likelihood decoder, whose performance is an upper bound on all *PPT* extractors. For schemes that satisfy this setting, we say that they are weakly secure. Next, we consider extractor which is *PPT* . If a scheme is weakly secure, the adversary is unable to discard information and yet evade detection. However, there might not be an efficient algorithm in recovering the original. For instance, an adversary might apply an one-way function on the data and yet be able to carry out the verification. Although no information is lost, there is no efficient algorithm to transform it back to the original. Such weaker requirement is easier to handle.

3.1 Remote Integrity Check Model

A remote integrity check ($\mathcal{RIC}$) system consists of two entities, the owner and the server. The life cycle of $\mathcal{RIC}$ starts with a setup phase followed by a sequence of verification phases. An owner has a small private and reliable memory and is associated with three *PPT* (polynomial w.r.t. the security parameter κ) algorithms, the *key generator* $\mathcal{K}$, the *encoder* $\mathcal{E}$ and *verifier* $\mathcal{V}$. A server has a large storage and is associated with a *PPT* algorithm, the *prover* $\mathcal{P}$.

Setup Phase. An owner has a file $\mathbf{x} \in \{0,1\}^*$, and chooses a key k using the key generator: $k \leftarrow \mathcal{K}(\kappa)$. Given $\mathbf{x}$ and k, the encoder $\mathcal{E}$ outputs *public data* p_x and *private data* s_x, that is, $\mathcal{E}(\mathbf{x}, k) = (p_x, s_x)$. The public data p_x is then sent to the server and stored in server's storage. The private data s_x is stored in the owner's private memory.

Verification Phase. During this phase, $\mathcal{V}$ (on the owner side) interacts with $\mathcal{P}$ (on the server side). Let $\mathcal{M}_s$ denote the (possibly modified) storage content at the server. Let $\langle \mathcal{V}(s_x), \mathcal{P}(\mathcal{M}_s) \rangle$ be the output of $\mathcal{V}$ after the interactions, which is either 0 or 1. If it is 0, the owner rejects the server. A RIC system is *valid* if $\langle \mathcal{V}(s_x), \mathcal{P}(p_x) \rangle$ always outputs 1. In this phase, the owner plays the role of verifier, hence we use the terms "owner" and "verifier" interchangeably.

3.2 Security Model

Adversary. The adversary will go through two phases: learning phase and challenge phase. In the two phases, the adversary behaves in different modes accordingly. We denote the adversary with $\mathcal{A}$. During the learning and challenge phase, we write it as $\mathcal{A}_{\texttt{learn}}$ and $\mathcal{A}_{\texttt{chal}}$ respectively.

Learning Phase. The adversary chooses[2] a data file $\mathbf{x}$ and sends it to the owner. The setup phase in the previous section is then carried out. Next, polynomial number of verifications are carried out and the adversary (who plays the role of server) does not need to honestly follow the verification protocol. The adversary may modify the storage but it will be fixed at the end of the learning phase. Let us denote the storage content as $\mathcal{M}_s$.

Challenge Phase. During the challenge phase, the memory content $\mathcal{M}_c$ of the verifier and the $\mathcal{M}_s$ at the adversary will be fixed, i.e. both the verifier and adversary are stateless.

Table 2 gives a summary of the learning phase and challenge phase. In Table 2, $\mathcal{O}(\cdot)$ is the oracle carrying out the owner's algorithms.

Learning Phase $\mathsf{Exp}_{\mathcal{A}}^{\text{learn}}(\kappa)$	Challenge Phase $\mathsf{Exp}_{\mathcal{A}}^{\texttt{chal}}(\mathcal{M}_s, \mathcal{M}_c)$
1. $k \leftarrow \mathcal{K}(\kappa)$	1. $b \leftarrow \langle \mathcal{V}(\mathcal{M}_c), \mathcal{A}_{\texttt{chal}}(\mathcal{M}_s) \rangle$
2. $(\mathbf{x}, \mathcal{M}_s, \mathcal{M}_c) \leftarrow \mathcal{A}_{\texttt{setup}}^{\mathcal{O}(k)}$	2. Output b.
3. Output $(\mathbf{x}, \mathcal{M}_s, \mathcal{M}_c)$.	

Table 2. Summary of the learning and challenge phase.

Advantage of Adversary. The goal of an adversary is to discard some information, but yet evade detection. We will model the information loss in two different settings. Under the first setting, we consider the maximum likelihood decoder given $\mathcal{M}_s$ and $\mathcal{M}_c$. Let $\mathbf{Y}_0, \mathbf{Y}_1$ and $\mathbf{Y}_2$ denote the random variable for data $\mathbf{x}$, adversary's memory $\mathcal{M}_s$ and verifier's memory $\mathcal{M}_c$ respectively. We define $\mathsf{ML}(\mathcal{M}_s, \mathcal{M}_c)$ as follow.

$$\mathsf{ML}(\mathcal{M}_s, \mathcal{M}_c) = \max_{\mathbf{x}_0} \mathsf{Pr}\left(\mathbf{Y}_0 = \mathbf{x}_0 \mid \mathbf{Y}_1 = \mathcal{M}_s, \mathbf{Y}_2 = \mathcal{M}_c\right).$$

We define $\mathsf{Succ}_{\mathcal{A}}^{\texttt{chal}}(\mathcal{M}_s, \mathcal{M}_c)$ as the probability that adversary $\mathcal{A}$ passes verification, given that $\mathcal{A}$ has $\mathcal{M}_s$ and the verifier has $\mathcal{M}_c$.

$$\mathsf{Succ}_{\mathcal{A}}^{\texttt{chal}}(\mathcal{M}_s, \mathcal{M}_c) = \mathsf{Pr}\left[\mathsf{Exp}_{\mathcal{A}}^{\texttt{chal}}(\mathcal{M}_s, \mathcal{M}_c) = 1\right].$$

We define the advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{ML}}(\beta, \gamma; \kappa)$ of adversary w.r.t. security parameter κ as follow.

[2] In practice, the data file is chosen by the user. Here we assume a stronger adversary.

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{ML}}(\beta,\gamma;\kappa) = \mathsf{Pr}\begin{bmatrix} (\mathbf{x},\mathcal{M}_s,\mathcal{M}_c) \leftarrow \mathsf{Exp}_{\mathcal{A}}^{\mathtt{learn}}(\kappa)\ \wedge \\ \mathsf{Succ}_{\mathcal{A}}^{\mathtt{chal}}(\mathcal{M}_s,\mathcal{M}_c) \geq \beta\ \wedge\ \mathsf{ML}(\mathcal{M}_s,\mathcal{M}_c) < \gamma \end{bmatrix}.$$

Intuitively, the advantage of the adversary $\mathcal{A}$ is the probability that, $\mathcal{A}$ achieves false acceptance rate of at least β after learning, but the information loss of $\mathbf{x}$ is at least $1-\gamma$.

Definition 1 *A remote integrity check system is (β,γ)-weakly secure, if for any PPT adversary $\mathcal{A}$, the advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{ML}}(\beta,\gamma;\kappa)$ of $\mathcal{A}$ is negligible in κ, i.e. for any positive polynomail poly$(\cdot)$, for all sufficiently large κ,*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{ML}}(\beta,\gamma;\kappa) \leq \frac{1}{poly(\kappa)}.$$

Note that Definition 1 is equivalent with

$$\mathsf{Pr}\begin{bmatrix} (\mathbf{x},\mathcal{M}_s,\mathcal{M}_c) \leftarrow \mathsf{Exp}_{\mathcal{A}}^{\mathtt{learn}}(\kappa)\ \wedge\ \mathsf{Succ}_{\mathcal{A}}^{\mathtt{chal}}(\mathcal{M}_s,\mathcal{M}_c) \geq \beta \\ \Rightarrow \mathsf{ML}(\mathcal{M}_s,\mathcal{M}_c) \geq \gamma \end{bmatrix} = 1 - negl(\kappa).$$

That is, if $\mathcal{A}$ passes verification with high chance ($\geq \beta$), then the information loss is low ($< 1-\gamma$) with overwhelming high probability.

Similarly, we define the the security for *PPT* extractor. The success probability of an algorithm `extract`, which tries to extract $\mathbf{x}$ from $\mathcal{M}_c$ and $\mathcal{M}_s$, is defined as follow.

$$\mathsf{Succ}_{\mathcal{A}}^{\mathtt{extract}}(\mathcal{M}_s,\mathcal{M}_c;\mathbf{x}) = \mathsf{Pr}\left[\mathbf{x}^* \leftarrow \mathtt{extract}^{\mathcal{A}(\cdot)}(\mathcal{M}_s,\mathcal{M}_c)\ \wedge\ \mathbf{x}^* = \mathbf{x}\right].$$

We define the advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{extract}}(\beta,\gamma;\kappa)$ of adversary $\mathcal{A}$ w.r.t. algorithm `extract` and security parameter κ as,

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{extract}}(\beta,\gamma;\kappa) = \mathsf{Pr}\begin{bmatrix} (\mathbf{x},\mathcal{M}_s,\mathcal{M}_c) \leftarrow \mathsf{Exp}_{\mathcal{A}}^{\mathtt{learn}}(\kappa)\ \wedge \\ \mathsf{Succ}_{\mathcal{A}}^{\mathtt{chal}}(\mathcal{M}_s,\mathcal{M}_c) \geq \beta\ \wedge\ \mathsf{Succ}_{\mathcal{A}}^{\mathtt{extract}}(\mathcal{M}_s,\mathcal{M}_c;\mathbf{x}) < \gamma \end{bmatrix}.$$

Definition 2 *A remote integrity check system is (β,γ)-secure, if for any PPT adversary $\mathcal{A}$, there exists a PPT algorithm* `extract`*, the advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{extract}}(\beta,\gamma;\kappa)$ of $\mathcal{A}$ is negligible in κ, i.e. for any positive polynomail poly$(\cdot)$, for all sufficiently large κ,*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{extract}}(\beta,\gamma;\kappa) \leq \frac{1}{poly(\kappa)}.$$

In Definition 2, given an adversary, we require the existence of an extractor. Thus, for different adversaries, the extractor can be different. Note that we only require the existence of such extractor, and do not need to explicitly give an extractor in designing a $\mathcal{RIC}$ system. On the other hand, in the security formulation of $\mathcal{POR}$, the order of the two quantifiers are exchanged: there is an extractor that is able to handle any adversary. Furthermore, under the $\mathcal{POR}$ framework, the extractor is a component in the system and has to be explicitly given in designing a $\mathcal{POR}$ system.

Trap-door Compression. To highlight the difference between $\mathcal{POR}$ and $\mathcal{RIC}$, we introduce a notion of compressibility on hash function. From such functions, we can design a $\mathcal{RIC}$ system that is not $\mathcal{POR}$. We will give an informal description here. Consider a keyed hash function H. We say that it is a *trap-door compression*, if there is an efficient key generation algorithm that produces a public key e, and private key d, and two PPT algorithms, the compression $\mathcal{C}$ which is a many-to-one function that discards information, and $\mathcal{D}$ s.t. for any r,

$$\mathcal{D}(d, \mathcal{C}(d, x), r) = H(e, r, x).$$

However, without knowing d, for any PPT compression $\widetilde{\mathcal{C}}$ and PPT algorithm $\widetilde{\mathcal{D}}$, there exist many r's, s.t.

$$\widetilde{\mathcal{D}}(e, \widetilde{\mathcal{C}}(e, x), r) \neq H(e, r, x).$$

In other words, with the private key, it is possible to discard information and yet compute the hash value for any r. However, it is difficult to do so without the knowledge of the private key.

4 Schemes for Remote Integrity Check

4.1 `AUTH`: MAC and ECC

For completeness, we will describe the authenticator that uses message authentication code (MAC) and error-correcting code (ECC) by Naor et al. [12].

Let us illustrate the scheme using Reed-Solomon code and a MAC that produces a κ bits tag. The file is represented as $\mathbf{x} = x_1 x_2 \ldots x_m$ where each $x_i \in \mathbb{Z}_p$ and p is prime. The owner chooses a secret key s. Next, $\mathbf{x}$ is encoded using Reed-Solomon code, giving $\mathbf{y} = y_1 y_2 \ldots y_{2m}$. For each i, taking s as the key, the MAC t_i of (y_i, i) is computed. Finally, the sequence of tuples (y_i, t_i), for $1 \leq i \leq 2m$ are sent to the server. During verification, the verifier chooses a random $r, 1 \leq r \leq 2m$, and requests for the pair (y_i, t_i). The consistency of y_i and t_i is then verified using the key s.

Any modifications of a pair (y_i, t_i) can be detected using MAC. From the unmodified data, the original can be reconstructed using Reed-Solomon code as an erasure code. If the original is unable to be reconstructed, then at least $m+1$ pairs of $\{y_i, t_i\}$'s have been modified. Therefore, with probability more than $1/2$, the verifier rejects. To reduce the probability of false acceptance to below ϵ, the verification can be carried out $\Theta(\log \frac{1}{\epsilon})$ times, but incurring communication cost. The storage required is large, which is at least 4 times larger than the original.

This authenticator can be employed for remote integrity check, and we call it `AUTH`-(c, w), where the data of size $m\kappa$ is expanded to size $(c+1)m\kappa$ using ECC, and w randomly selected pairs (y_i, t_i)'s are accessed and checked during each verification.

4.2 HTAG: Homomorphic MAC

This scheme also appeared in an earlier technical paper by Shacham et al. [14]. The main idea is to reduce communication bits in AUTH using homomorphic MAC and an almost universal hash function. Each x_i is associated with an authentication tag t_i. During verification, the verifier chooses a key r and asks for the key-hashed value of $\mathbf{x}$ with r. Due to the homomorphic property of the tags, the server can also compute the tag of the hashed value from t_i's. Thus the verifier can check whether the server has carried out the computation honestly.

Setup. The owner chooses a κ bits prime p. The file is represented as $\mathbf{x} = x_1 x_2 \ldots x_m$ where $x_i \in \mathbb{Z}_p$ for each i. The owner randomly chooses s and α from $\mathbb{Z}_p^*$. Let $s_i = G(s, i)$ for $i = 1, 2, 3, \ldots, m$, where G is a secure pseudo random number generator. The owner computes a tag $t_i = \alpha x_i + s_i \mod p$, for each i, and sends $\mathbf{x}$, t_i's and p to the server. The value s and α are kept as secrets.

Verification. The verifier chooses a random r from $\mathbb{Z}_p^*$ and sends r to the server. The server computes A and B as follow and sends them to the verifier.

$$A = \mathcal{H}_p(r; \mathbf{x}) \triangleq \sum_{i=1}^{m} r^i x_i \mod p, \qquad B = \sum_{i=1}^{m} r^i t_i \mod p.$$

The verifier accepts if $B \equiv \alpha A + \mathcal{H}_p(r, \mathbf{s}) \pmod{p}$, where $\mathbf{s} = s_1 s_2 \ldots s_m$.

THEOREM 1 HTAG *is* $(0.5, 1 - negl(\kappa))$*-secure, assuming* $G(\cdot, \cdot)$ *is a cryptographic secure pseudo random number generator, where* $negl(\cdot)$ *is a negligible function and* κ *is the security parameter.*

Tradeoff. By adding redundancy, the variant HTAG-(c, w) can reduce the number of read accesses at the cost of storage size. The data $x_1 x_2 x_3 \ldots x_m$ is encoded as $y_1 y_2 \ldots y_{cm+m}$ using an ECC. Each y_i is associated with a tag t_i. During each verification, the verifier chooses w random indices $i_1, i_2, \ldots, i_w$ and sends these w indices together with the random key r to the server. The server computes and sends $A = \mathcal{H}_p(r; y_{i_1}, y_{i_2}, \ldots, y_{i_w})$ and $B = \mathcal{H}_p(r; t_{i_1}, t_{i_2}, \ldots, t_{i_w})$ to the verifier. To further reduce communication bits, the w indices are generated from a seed r_0, so that only two numbers r and r_0 are sent.

4.3 HENC: Homomorphic encryption

We give another simple scheme HENC. It also uses a homomorphic tag similar to HTAG but its goal is to reduce storage size instead of communication bits. In its basic form, it requires high communication cost but can be made efficient by incorporating other techniques.

Setup. The owner chooses a $p = 2q + 1$, where both p, q are primes, and a generator g of $\mathbb{Z}_p^*$. The file is organized as $x_1x_2x_3 \dots x_m$ where each $x_i \in \mathbb{Z}_{p-1}$. The file and p are then sent to the server. The owner also chooses r from $\mathbb{Z}_{p-1}^*$, and computes

$$h = \sum_{i=1}^{m} r^i x_i \mod (p-1). \tag{1}$$

Both r and h are kept as secrets. It is not necessary to keep g secret.

Verification. The verifier picks a random α, and sends the sequence

$$\langle\ b_i = g^{\alpha r^i} \mod p\ \rangle_{i=1,2,3,\dots,m}$$

The server is supposed to compute and return $A = \prod_{i=1}^{m} b_i^{x_i} \mod p$. The verifier accepts if $A \equiv g^{\alpha h} \pmod{p}$.

Remarks.
1. HENC is $(0.5, 1 - negl(\kappa))$-weakly secure, assuming that it is difficult to distinguish b_i's from random numbers.

Theorem 2 *The scheme* HENC *is* $(0.5, 1 - negl(\kappa))$*-weakly secure under* Assumption 3*, where* $negl(\cdot)$ *is a negligible function and* κ *is the security parameter.*

Assumption 3 *Let* $p = 2q+1$ *where* p, q *are primes, and* g *be a generator of* $\mathbb{Z}_p^*$*. The following two sequences, where* $r, r_1, r_2, \dots, r_m$ *are uniformly chosen from* $\mathbb{Z}_{p-1}^*$*, are computationally indistinguishable.*

$$\mathcal{R}_0 : \langle g^{r_1} \bmod p, g^{r_2} \bmod p, \dots, g^{r_m} \bmod p \rangle$$
$$\mathcal{R}_1 : \langle g^{r} \bmod p, g^{r^2} \bmod p, \dots, g^{r^m} \bmod p \rangle$$

The main idea of the proof is as follow. Suppose there is a successful adversary. Consider a tester who, on input of m number, $v_1, v_2, \dots, v_m$, generates instances for the learning phases, and simulates the adversary. Each learning instance is the sequence $v_i^a \mod p$ for $i = 1, \dots, m$, where a is randomly chosen. Next, the tester simulates an honest server, and the adversary during the challenge phase. If both produce the same response, the tester outputs 1, otherwise outputs 0. We can show that, if the input is from $\mathcal{R}_0$, the expected output is less than 0.3, whereas if the input is of the form $g^{r^i} \bmod p$, then the expected output is not less than 0.5. This contradicts Assumption 3.

2. To enhance the scheme from weakly-secure to secure, we require a *PPT* extractor. This can be achieved by the following modifications. Instead of choosing a prime p as modulus, we can incorporate the Paillier cryptosystem [13] and choose n^2 as modulus, where n is a composite, and an appropriate g. By

property of the Paillier cryptosystem, with the knowledge of the private key, the verifier can perform discrete-log. To recover the original from the storage, the extractor simulates sufficiently large number of verifications by sending the b_i's of the form

$$g^{\hat{r}} \bmod n^2, g^{\hat{r}^2} \bmod n^2, \ldots g^{\hat{r}^m} \bmod n^2$$

where $\hat{r}$ is randomly chosen for each verification. The correct response from the server gives the sample of a m-degree polynomial evaluated at $\hat{r}$. There may be "errors" in the server's responses, which can be corrected using list decoding. Thus, we have a $(0.5, 1 - negl(\kappa))$-secure scheme.

3. Using bilinear map, communication required can be reduced to $O(\sqrt{m})$ numbers. Note that it is not necessary to use the $r, r^2, r^3, \ldots, r^m$ as coefficients in (1). They can be a sequence of pseudo random numbers $r_1, r_2, \ldots, r_m$ chosen during setup phase. We can also construct each of the m coefficients using the product $r_i s_j$, for $i, j = 1, 2, \ldots, \sqrt{m}$, where the r_i's and s_j's are pseudo random. Now, using bilinear map, the number of values to be sent is reduced to $2\sqrt{m}$. That is, the verifier just need to send the sequences $g_1^{r_i}$'s and $g_1^{s_j}$'s, and the server can compute $e(g_1^{r_i}, g_1^{s_j})$ to obtain $g_2^{r_i s_j}$ for any i and j, where $e(\cdot, \cdot)$ is the bilinear map and g_1 and g_2 are generators of the two respective groups. We can show that this variant is weakly secure. However, it is not clear how to incorporate Paillier cryptosystem into this variant.

4.4 HYB-(c, w, ℓ): Hybrid of HENC with HTAG

For simplicity, we first explain the algorithm when $c = 0$ and $w = m\kappa/\ell$, where the file size is $m\kappa$. Thus, the only parameter is ℓ. The data $\mathbf{x}$ is grouped into blocks. The scheme HENC is applied on each block to obtain a sequence of hash values. Next, the homomorphic MAC is applied on the hash values.

Setup. The owner choose a κ bits $p = 2q + 1$ where both p, q are prime, and a generator g of $\mathbb{Z}_p^*$. Organized the file into u blocks and each block contains ℓ numbers from $\mathbb{Z}_{p-1}$. Let $x_{i,j}$ be the j-th number in the i-th block. The owner chooses a random r from $\mathbb{Z}_{p-1}^*$, and a seed s. Let $s_i = G(s, i)$ for $1 \le i \le u$ where G is a pseudo random number generator. Compute the tag t_i for the i-th block as follow:

$$h_i = \mathcal{H}_{p-1}(r; x_{i,1}, x_{i,2}, \ldots, x_{i,\ell}) = \sum_{j=1}^{\ell} x_{i,j} r^j \mod (p-1),$$
$$t_i = (h_i + s_i) \mod (p-1).$$

Send the data $x_{i,j}$'s, tags t_i's and p to the server. Keep s and r as secrets. It is not necessary to keep g secret.

Verification.

1. The verifier chooses random numbers a, α from $\mathbb{Z}_p^*$ and computes a sequence of numbers $b_1, b_2, \ldots, b_\ell$ where $b_i = g^{\alpha r^i} \mod p$. The verifier sends a and the b_i's to the server.
2. The server computes values A and B as follow and sends them to the verifier.

$$B = \prod_{i=1}^{u} \prod_{j=1}^{\ell} b_j^{a^i x_{i,j}} \mod p, \qquad A = \sum_{i=1}^{u} t_i a^i \mod (p-1).$$

3. The verifier accepts if $\quad Bg^{\alpha \mathcal{H}_{p-1}(a; s_1, s_2, \ldots, s_u)} \equiv g^{\alpha A} \pmod p$.

Security. We can show that HYB is $(0.5, 1 - negl(\kappa))$-weakly secure under Assumption 3 and Assumption 4 (defined below). Assumption 4 is required to ensure "forgery" of tags is difficult.

Assumption 4 *Let p be a κ bits prime, and g be a generator of $\mathbb{Z}_p^*$. Given a sequence $b_1, b_2, \ldots, b_m$, where $b_i = g^{\alpha r^i} \mod p$, and α, r are uniformly randomly distributed over $\mathbb{Z}_p$, it is infeasible to compute a pair of values (C, D) such that $C = D^\alpha$ and $D \neq 1$.*

Similarly, by using Paillier cryptosystem, we have a scheme that is $(0.5, 1 - negl(\kappa))$-secure.

Tradeoff. By adding redundancy, HYB-(c, w, ℓ) can tradeoff server's storage with the number of read accesses per verification, without incurring more communication bits. The $m\kappa$-bits file is first encoded using ECC and expanded to $(c+1)m\kappa$ bits. During each verification, only w blocks are selected. Therefore, the total number of bits accessed is reduced to $O(w\kappa\ell)$.

4.5 REDACT: Redactable Signature Scheme

Redactable signature scheme is a homomorphic signature scheme [9]. We consider such schemes for unordered sets where a message is a set $\mathbf{x} = \{x_1, x_2, \ldots, x_m\}$ of objects, and a set $\mathbf{x}'$ is a *redacted* message of $\mathbf{x}$ if $\mathbf{x}' \subset \mathbf{x}$. As usual, with the private key, the signer can compute a signature σ. A redactable signature scheme enables an entity, known as the redactor, to compute a valid signature for a redacted message $\mathbf{x}'$ from the message $\mathbf{x}$ and its signature σ, without knowing the private key. Hence, the scheme consists of three algorithms *Sign*, *Redact*, and $Verify$. There are known schemes [9] that produce short signature of length within a constant factor of the key size κ.

The main idea in REDACT is simple: The data $\mathbf{x} = x_1 x_2 \ldots x_m$ is first encoded using ECC to get $\mathbf{y} = y_1 y_2 \ldots y_{2m}$ (so the redundancy rate $c = 1$). The encoded data $\mathbf{y}$ is then represented as an unordered set $p_x = \{(1, y_1), (2, y_2), \ldots, (2m, y_{2m})\}$. The owner signs the unordered set p_x, and the server stores the original $\mathbf{x}$ and the signature σ. Thus, the addition storage size is $O(\kappa)$. During verification, the verifier sends an index, say i. The server is supposed to compute y_i from $\mathbf{x}$. By property of the redactable signature scheme, the signature σ_i for the singleton set $\{(i, y_i)\}$ can be computed by the server. The verifier then check whether σ_i is a valid signature of $\{(i, y_i)\}$. Since the verifier does not need to know the private key, this scheme can be employed in scenarios where the verifier is not the owner.

THEOREM 5 `REDACT` *is* $(0.5, 1-negl(\kappa))$*-secure, if the redactable signature scheme employed by* `REDACT` *is unforgeable, where* $negl(\cdot)$ *is a negligible function and* κ *is the security parameter.*

Tradeoff. Let us consider the extension `REDACT`-(c, w, ℓ) which reduces number of bits accessed at the cost of the server's storage. Using ECC, the data $\mathbf{x}$ is expanded by a factor of $(c+1)$. Next, the encoded data is divided into blocks, where each block contains ℓ elements and each elements is κ bits. Each block is signed independently. Thus there are $(c+1)m/\ell$ signatures. During the verification phase, the verifier sends w random indices $i_1, i_2, \ldots, i_w$ to the server. The server computes signatures σ_{i_j}'s for each y_{i_j} and sends them to the verifier.

4.6 `RSAb`: Trapdoor Compression

Filho et al. [7] proposed a RSA-based scheme. We will describe this scheme here and call it `RSAb`. Note that `RSAb` is not a $\mathcal{POR}$ system, and it seems to be a trap-door compression and secure $\mathcal{RIC}$ system. However, there is no proof yet.

Setup. Data $\mathbf{x}$ is represented as a single integer. Choose a κ bits RSA modulus n. The owner keeps $s = \mathbf{x} \mod \phi(n)$ as secret, and sends $\mathbf{x}$ and n to the server.

Verification. Verifier randomly chooses $r \in Z_n^*$ and sends r to the server. Server computes $z = r^{\mathbf{x}} \mod n$ and sends it back to the verifier. Verifier accepts if, $r^s \equiv z \pmod{n}$.

Note that the function $H(\mathbf{x}, r_0) = r_0^{\mathbf{x}} \mod n$, where r_0 is a fixed constant with maximum order in $\mathbb{Z}_n^*$, is collision resistant, assuming factorization is difficult. It is easy to show that, an adversary who can evade detection, must employ a one-way function to discard information. If not, the adversary can find a collision.

Tradeoff. We give an extension `RSAh`-(c, w, ℓ) by exploiting a homomorphic property and incorporating ECC. This hybrid achieves the best asymptotic performance among the schemes discussed here.

The data $\mathbf{x}$ of $m\kappa$ bits is encoded using ECC. The encoded data of size $(c+1)m\kappa$ is divided into blocks, where each block is represented as a single $\ell\kappa$-bits integer. Let b_i be the i-th block. Each b_i is associated with a tag

$$t_i = g^{b_i + s_i} \mod n,$$

where n is the κ-bits RSA modulus, g an element with large order, and the s_i's are secrets chosen by the owner. The data b_i's, tags t_i's, and n are sent to the server.

During verification, the verifier randomly chooses α and r from $\mathbb{Z}_n$, and computes $h = g^\alpha \mod n$. The verifier also chooses w random indices $i_1, i_2, \ldots, i_w$, and sends these w indices, h and r to the server. Let $\hat{b}_j = b_{i_j}$, $\hat{s}_j = s_{i_j}$ and $\hat{t}_j = t_{i_j}$ for each $1 \le j \le w$. The server is supposed to compute and send back H and T defined as follow:

$$H = h^u \mod n, \text{ where } u = \sum_{i=1}^{w} r^i \ \hat{b}_i, \qquad T = \prod_{i=1}^{w} (\ \hat{t}_i\ )^{r^i} \mod n.$$

The verifier accepts if $Hg^{\alpha v} \equiv T^\alpha \pmod{n}$ where $v = \sum_{i=1}^{w} r^i \hat{s}_i$.

The w indices can be generated from a short seed using pseudo random number generator, and s_i's can also be generated from another short seed. Although not obvious at first glance, `RSAh` can be treated as an extension of `RSAb`.

Efficiency. For `RSAb`, the communication cost, verifier's storage size, and additional server's storage size are $O(\kappa)$ bits, and the number of read access is $m\kappa$ bits which is exactly the size of the data. The variant scheme `RSAh`-(c, w, ℓ) reduces the number of read access to $(1+\ell)w\kappa$ bits at the cost of $(c + \frac{1+c}{\ell})m\kappa + O(\kappa)$ bits of additional storage. The communication cost are still in $O(\kappa)$ bits. Hence, in term of asymptotic performance, `RSAh`-(c, w, ℓ) is the most efficient among the proposed schemes.

5 Conclusion

The subtle difference between $\mathcal{RIC}$ and $\mathcal{POR}$ seems to be profound, and related to compressibility of hash functions. This is illustrated by the simple scheme `RSAb`. Although in a simple form, `RSAb` is not easy to analyze and new techniques seems to be required. In this paper, we focus on asymptotic performance. It is also interesting to investigate the performance of the proposed schemes in practical scenarios, and how to combine the underlying techniques for better tradeoff.

Acknowledgement. The authors wish to thank the anonymous reviewers for pointing out relevant references and their valuable suggestions, and Aldar Chan, National University of Singapore, for the insightful discussions.

References

1. S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004.
2. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. *ACM conf. on Computer and Communications Security*, pages 598–609, 2007.

3. C. Batten, K. Barr, A. Saraf, and S. Treptin. pStore: A secure peer-to-peer backup system. *LCS Technical Memo 632, MIT Laboratory for Computer Science*, 2001.
4. M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *IEEE Sym. on Foundations of Comp. Sci.*, pages 90–99, 1991.
5. K. D. Bowers, A. Juels, and A. Oprea. Proofs of retrievability: Theory and implementation. Cryptology ePrint Archive, Report 2008/175, 2008. http://eprint.iacr.org/.
6. E.-C. Chang, S. Mukhopadhyay, and J. Xu. Remote integrity check without the original. Manuscript submitted to CRYPTO'07, 2007. http://www.comp.nus.edu.sg/∼changec/publications/remote.pdf.
7. D. Filho and P. Barreto. Demonstrating data possession and uncheatable data transfer. Cryptology ePrint Archive, Report 2006/150, 2006.
8. D. Harnik and M. Naor. On the Compressibility of NP Instances and Cryptographic Applications. *IEEE Sym. on Foundations of Comp. Sci.*, pages 719–728, 2006.
9. R. Johnson, D. Molnar, D. X. Song, and D. Wagner. Homomorphic Signature Schemes. *CT-RSA*, pages 244–262, 2002.
10. A. Juels and Jr. B. S. Kaliski. Pors: proofs of retrievability for large files. *ACM conf. on Computer and Communications Security*, pages 584–597, 2007.
11. J. Li and F. Dabek. F2F: reliable storage in open networks. *Intern. Workshop on Peer-to-Peer Systems*, 2006.
12. M. Naor and G. N. Rothblum. The Complexity of Online Memory Checking. *IEEE Symp. on Foundations of Comp. Sci.*, pages 573–584, 2005.
13. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *EUROCRYPT*, pages 223–238, 1999.
14. H. Shacham and B. Waters. Compact proofs of retrievability. Cryptology ePrint Archive, Report 2008/073, 2008. http://eprint.iacr.org/.
15. G. E. Suh, D. Clarke, B. Gasend, M. van Dijk, and S. Devadas. Efficient memory integrity verification and encryption for secure processors. *IEEE/ACM Int. Sym. on Microarchitecture*, pages 339–350, 2003.

A Proof of Theorem 1

Let us introduce two related terms: *correct* and *valid*. We say a reply r_i to a query q_i is valid, if r_i passes the verification. We say r_i is correct, if r_i equals r_i^*, where r_i^* is the reply message for query q_i from the honest server. Correctness implies validity, but in general the converse is not true.

Theorem 1 `HTAG` *is* $(0.5, 1 - negl(\kappa))$*-secure, assuming* $G(\cdot,\cdot)$ *is a cryptographic secure pseudo random number generator, where* $negl(\cdot)$ *is a negligible function and* κ *is the security parameter.*

Proof: We first prove that if a reply from the adversary is valid, with overwhelming high probability it is correct, assuming $G(\cdot,\cdot)$ is a cryptographic secure pseudo random number generator. Let the data be $\mathbf{x}$. Given the query r, let (A, B) be the adversary's reply and (A_0, B_0) be the correct reply.

It is not difficult to show that, if there exists an adversary $\mathcal{A}$ which outputs (A, B) such that $A \neq A_0 \mod p$ and (A, B) passes $\mathcal{V}$, then there exists an algorithm which can compute the value of g. If s_i's are truely random sequence, g can be any element in $\mathbb{Z}_p^*$ with equal likelihood. However, our s_i's is pseudo random. Hence, if g can be found, this will contradict the assumption that s_i is from a cryptographic secure PRNG. Hence, with high probability, the valid reply is correct.

Now, let $\mathcal{A}$ be an adversary who can pass the verification with probability at least 0.5. We want to give an algorithm to recover the file $\mathbf{x}$ from the adversary's storage. Note that $\mathcal{H}_p(r; \mathbf{x})$ is a m-degree polynomial with variable r. For each query r_0, the value A is a sample of the polynomial at r_0. To recover the original, the algorithm simulates the verification for sufficient large enough rounds so that with high probability, more than $m+1$ of the replies are correct. From these correct replies, we can determine the polynomial whose coefficients are the file.

□

B Security analysis of HENC

Theorem 2. HENC *is* $(0.5, 1 - negl(\kappa))$*-weakly secure under* Assumption 3*, where* $negl(\cdot)$ *is a negligible function and* κ *is the security parameter.*

Proof: Suppose there exists an adversary $\mathcal{A}$, who can pass the verification with probability at least 0.5 and the information loss is larger than $\frac{1}{poly(\kappa)}$ for some polynomial $poly(\cdot)$. More precisely,

$$\mathsf{Succ}_{\mathcal{A}}^{\mathtt{chal}}(\mathcal{M}_s, \mathcal{M}_c) \geq 0.5, \quad \mathsf{ML}(\mathcal{M}_s, \mathcal{M}_c) < 1 - \frac{1}{poly(\kappa)}.$$

Let $\mathcal{R}_0$ denote the distributions of m uniformly and randomly chosen numbers from $\mathbb{Z}_p^*$, and $\mathcal{R}_1$ denote the distribution of the sequence stated in ASSUMPTION 3. We now construct a test $\mathcal{T}$, as shown in Table 3, which is able to distinguish $\mathcal{R}_1$ from $\mathcal{R}_0$.

We want to determine the difference:

$$\Delta = |\mathsf{Pr}[\mathcal{T}(\mathcal{R}_1) = 1]| - |\mathsf{Pr}[\mathcal{T}(\mathcal{R}_0) = 1]|.$$

Let us consider the case when the input is from $\mathcal{R}_1$. Note that the queries constructed follows the same distribution of the queries constructed by the owner. Thus, the adversary can pass the verification with probability at least 0.5 and so,

$$\mathsf{Pr}[\mathcal{D}(\mathcal{R}_1) = 1] \geq 0.5$$

When the input is from $\mathcal{R}_0$, we want to show by contradiction that the probability that $A = A^*$ is less than 0.4. Suppose that it is greater than 0.4, consider an extractor (without constrain on its running time) that simulates the adversary. For verification, the extractor sends in random numbers. By treating the x_i's as unknowns, and taking discrete log on A, each verification gives an equation in a linear system. By carrying out sufficiently large number of verifications, with high probability, more than 0.3 fraction of the verifications are correct. Although there are errors, the extractor can recover the correct solution in the overdetermined system with overwhelming high probability. This contradicts the fact that the maximum likelihood decoder is unable to recover the original with high probability. Thus, $|\mathsf{Pr}[\mathcal{T}(\mathcal{R}_0) = 1]| < 0.4$.

Therefore, Δ is greater than a constant, contradicting the assumption that $\mathcal{R}_1$ and $\mathcal{R}_0$ are computationally indistinguishable. So, such adversary $\mathcal{A}$ does not exists.

□

The input is a sequence $b_1, b_2, \ldots, b_m$ from $\mathbb{Z}_p^*$.

1. Simulates adversary $\mathcal{A}$ in the learning phase. For each verification requested by the adversary, constructs the query in this way: picks a random a, and takes the following as the query,

$$b_1^a \text{mod } p,\ b_2^a \text{mod } p,\ \ldots,\ b_m^a \text{mod } p$$

   Let $\mathbf{x} = x_1 x_2 \ldots x_m$ be the generated data file.
2. Simulates the adversary in the challenge phase and takes b_i^a's as the query, where a is randomly chosen. Let A be the adversary output.
3. Simulates the honest server and computes

$$A^* = \prod_{i=1}^{m} b_i^{x_i} \quad \text{mod } p.$$

4. Outputs 1 if $A = A^*$. Otherwise, outputs 0.

Table 3. Test $\mathcal{T}$: Distinguishing $\mathcal{R}_0$ from $\mathcal{R}_1$.