Leakage Resilient Proofs of Ownership in Cloud Storage, Revisited [⋆]

Jia Xu and Jianying Zhou

Infocomm Security Department, Institute for Infocomm Research, Singapore
{xuj, jyzhou}@i2r.a-star.edu.sg

Abstract. Client-side deduplication is a very effective mechanism to reduce both storage and communication cost in cloud storage service. Halevi *et al.* (CCS '11) discovered security vulnerability in existing implementation of client-side deduplication and proposed a cryptographic primitive called "proofs of ownership" (PoW) as a countermeasure. In a proof of ownership scheme, any owner of the same file can prove to the cloud storage server that he/she owns that file in an efficient and secure manner, even if a bounded amount of any efficiently extractable information of that file has been leaked. We revisit Halevi *et al.*'s formulation of PoW and significantly improve the understanding and construction of PoW. Our contribution is twofold:

- Firstly, we propose a generic and conceptually simple approach to construct *Privacy-Preserving* Proofs of Ownership scheme, by leveraging on well-known primitives (i.e. Randomness Extractor and Proofs of Retrievability) and technique (i.e. sample-then-extract). Our approach can be roughly described as Privacy-Preserving PoW = Randomness Extractor + Proofs of Retrievability.
- Secondly, in order to provide a better instantiation of Privacy-Preserving-PoW, we propose a novel design of randomness extractor with large output size, which improves the state of art by reducing both the random seed length and entropy loss (i.e. the difference between the entropy of input and output) simultaneously.

Keywords: Cloud Storage, Client-side Deduplication, Proofs of Ownership, Leakage Resilience, Privacy-Preserving, Proofs of Retrievability, Randomness Extractor, Sample-then-Extract

1 Introduction

Cloud storage service (e.g. Dropbox, Skydrive, Google Drive, iCloud, Amazon S3) is becoming more and more popular in recent years [1]. The volume of personal or business data stored in cloud storage keeps increasing [2,3,4]. In face to the challenge of rapidly growing volume of data in cloud, deduplication technique is highly demanded to save disk space by removing duplicated copies of the same file (Single Instance Storage). SNIA white paper [5] reported that the deduplication technique can save up to 90% storage, dependent on applications.

Traditional deduplication technique (i.e. server side deduplication [6,7,8,9]) in centralized storage system removes duplicated copies residing in the same server. Unlike server-side deduplication, client-side deduplication in cloud storage system will identify duplicated copies such that one copy resides in the cloud storage server and the other resides remotely in the cloud client, and saves the uploading bandwidth (time, respectively) for the duplicated file. In both server and client side deduplication, all owners of the deduplicated file will be provided a soft link to the unique copy of that file stored in the centralized storage or cloud storage respectively. In contrast to server-side deduplication which saves only storage on server

[⋆] This work is supported by Singapore A*STAR project SecDC-112172014. A short version of this work is accepted by ACNS '14.

side, client-side deduplication saves not only server storage but also network bandwidth and transmission time, and benefits both cloud server and client.

However, how to implement client-side deduplication *securely* in an untrusted environment, is far more challenging than it first appears [10,11]. Arguably, the root cause of the difference between security requirements of server-side and client-side deduplication, is that server-side deduplication is executed in the trusted server, while client-side deduplication is distributively executed between the trusted[1] cloud server and potentially untrusted cloud client. Here the cloud user is considered as potentially untrusted, since anyone from the untrusted Internet could become a cloud user and the cloud server is unable to distinguish honest users from malicious users (i.e adversaries) in general.

Server side deduplication may simply apply a collision resistant hash function (say SHA256) to identity duplicated files in the storage server, and remove the extra copies to achieve "single instance storage". An existing implementation of client-side deduplication (called as "hash-as-a-proof" method) is as below: Cloud storage server keeps a lookup table, which records hash value of each file in its storage. Cloud user Alice, who tries to upload file F to the cloud storage, will firstly send hash value $\mathsf{hash}(F)$ to the cloud server. If $\mathsf{hash}(F)$ is not found in the lookup table, then Alice should upload file F to the cloud storage and cloud server will update the lookup table by adding entry $\mathsf{hash}(F)$. Otherwise, cloud server has a copy of F already, which could be uploaded by other users. Consequently Alice's uploading process will be saved, and Alice is allowed to download F from cloud server on demand. In the above method, the knowledge of hash value $\mathsf{hash}(F)$ is treated as a "proof" that Alice owns file F. Previously, Dropbox[2] applied the above "hash-as-a-proof" method on block-level cross-users deduplication [11][12].

Halevi *et al.* [11] targets the critical security vulnerability in the above "hash-as-a-proof" method, where the leakage of a short hash value $\mathsf{hash}(F)$ would lead (or amplify) to leakage of entire file F to outside adversary. Their work proposes a cryptographic primitive called "proofs of ownership" (PoW) to address such leakage amplification vulnerability. The distinguishable feature of Halevi *et al.* [11] from all of previous study in security of deduplication (e.g. convergent encryption [6,7,13]), is that Halevi *et al.* [11] adopts a *bounded leakage model* to characterize the untrusted environment in which the client-side deduplication runs. Their formulation requires that, after a setup between one owner of file F and the cloud storage server, any owner of F can efficiently *prove* (in the sense of "interactive proof system" [14]) to the cloud storage server that he/she indeed owns file F without really transmitting F, even if a bounded amount of any efficiently extractable information of F has been leaked via some owner (considered as the accomplice or colluder) of F intentionally or unintentionally.

In this work, we revisit Halevi *et al.* [11]'s formulation, and extend it in two aspects: (1) We shift a significant amount of workload (precisely, the setup procedure) from cloud server to a cloud user, which reflects our understanding of real world setting—the average computation power allocated to each online user by cloud server is typically smaller than the computation power of an average cloud user. (2) We protect data privacy against verifier (e.g. the cloud storage server), during the interactive proof protocol. Halevi *et al.* [11]'s formulation does not address privacy protection of user data against the cloud storage server. Prudent

[1] The cloud server is trusted in data integrity and availability in this work.

[2] In Feb 2012, we noticed that Dropbox disabled the deduplication across different users, probably due to recent vulnerabilities discovered in their original cross-user client-side deduplication method. This also indicates the importance and urgency in the study of security in client-side deduplication.

users may have reasons to not trust the cloud server. For example, the cloud server may be hacked (e.g.[15]), making it a single point of failure of user data privacy. In addition, the cloud server may make careless technical mistakes [16,17], which may expose user data to unauthorized persons. In this work, we will trust cloud storage server in data availability and integrity (which is the research topic of proofs of storage [18,19]), but not trust it in data privacy.

1.1 Overview of our result

Under the framework of Halevi *et al.* [11], in a secure PoW scheme, if the input file F has k bits min-entropy to the view of adversary at the very beginning and at most T $(< k - \lambda)$ bits of message about F is leaked at adversary's (adaptive) choice, then the adversary should not be able to convince the cloud storage server that he/she owns file F with significant probability.

1.1.1 Generic Construction of Privacy-Preserving-PoW Intuitively, our generic construction of Privacy-Preserving-PoW is as below: At first, apply a *proper*[3] randomness extractor over file F to output $T + 2\lambda$ $(< k)$ bits almost-uniform random number Y_F. Next, apply a *proper* proofs of retrievability (POR [18]) scheme over Y_F. Since the output Y_F of the randomness extractor is statistically close to true uniform randomness, any adversary that learns at most T bits arbitrary information of F, cannot output the $T + 2\lambda$ bits long value Y_F entirely with significant probability, and thus cannot succeed in the verification of POR scheme. The difference $(k - T)$ is like the *entropy loss* in randomness extractor, thus the smaller the difference $(k-T)$ is , the better the PoW scheme is in aspect of leakage resilience.

Our result can be combined with convergent encryption or Message-Locked Encryption [6,7,20,9,21], in order to construct strong leakage-resilient client-side deduplication scheme for encrypted data in cloud storage and thus protect data privacy against both outside adversary and curious cloud server.

We remark that formulating and constructing privacy-preserving PoW scheme are very challenging. Previous work by Ng *et al.* [22] made the first attempt towards this goal, but gave an unsatisfactory solution: As pointed out by Xu *et al.* [20], Ng *et al.* [22] formulates the privacy property *locally* for each block and their scheme suffers from "divide and conquer" attack: If an input file with N blocks has 1 bit min-entropy in each block *independently*, then this file could be recovered by an outside adversary via brute force search in time $\mathcal{O}(N)$ instead of $\mathcal{O}(2^N)$.

1.1.2 Improved Randomness Extractor Unfortunately, the state of art [23,24] (with restriction of small seed size and practical computation cost) of randomness extractor only gives us a PoW with $k - T = \Omega(|F|)$ and requires relatively large random seed. We propose a new randomness extractor with shorter random seed and results in a PoW with $k - T = \mathcal{O}(|F|^{1-c})$ for any constant $c \in (0, 1)$.

1.2 Contributions

Our main contributions can be summarized as below:

[3] See Theorem 1 and Theorem 2 for the explanation of "proper" randomness extractor and "proper" POR.

Table 1. Compare our PoW scheme with existing works. Unsatisfactory items are highlighted in italic font and red color.

Scheme	Distribution of input	Randomness complexity	Computation complexity	Privacy-Preserving	Security Model
PoW1 [11]	Any	$\mathcal{O}(\lambda)$	*Expensive [11]*	*No (Leaking whole file F)*	Stand. Model
PoW2 [11]	Any	$\geq \boldsymbol{6T}$ †	*Prohibitively expensive [11]*	*No*	Stand. Model
PoW3 [11]	*Generalized block-fixing distribution*	$\mathcal{O}(\lambda)$	Practical	*Unclear*	*SHA256 is R.O. and assume their algorithm generates a "good" code‡*
This work	Any	$\mathcal{O}(\lambda)$	Practical	Yes	Stand. Model

† T may take value 64MB.

‡ Theorem 3 in [11] relies on an unproven assumption that the code generated by the third construction PoW3 is "good" and authors of [11] admits that it is very hard to analyze this unproven assumption. See text surrounding Theorem 3 in [11].

Table 2. Compare randomness extractors with output size $\ell\rho$, where ℓ could take value as large as $2^{21} \approx 2$ millions. The input is file F. Unsatisfactory items are highlighted in italic font and red color.

Scheme	Distribution of input	Randomness complexity	Computation complexity	Entropy Loss	Security Model
$\mathsf{HMAC}(s_1,F)\|\ldots\|\mathsf{HMAC}(s_\ell,F)$	Any	$\boldsymbol{\ell\lambda}$	$\boldsymbol{\ell\|F\|}$	small	Random Oracle
Inner Product Universal Hash [25]	Any	$\boldsymbol{2\|F\|}$	$\Omega(\|F\|\log(\ell\rho))$	$2\log(1/\epsilon)$	Stand. Model
[23]	Any	$\boldsymbol{\mathcal{O}(\ell\lambda)}$	$2\|F\|\log\ell$	$\boldsymbol{\Omega(\|F\|)}$	Stand. Model
This work	Any	$\mathcal{O}(\lambda)$	$2\|F\|\log\ell$	$\mathcal{O}(\|F\|^{1-c})$ †	Stand. Model

†$c \in (0,1)$

1. We propose a generic and conceptually simple paradigm to construct proof of ownership scheme: PoW=Randomness Extractor + Proofs of retrievability. To the best of our knowledge, this is the first work that bridges the proof of ownership and randomness extractor. Our result improves previous works on PoW in the following aspects: (1) Complete proof of security in standard model for *any* distribution of input file, while still being practical. (2) The first generic framework to construct PoW and benefited from the future advance in randomness extractor or proofs of retrievability. (3) Privacy-Preserving against verifier (e.g. cloud storage server). A detailed comparison between our work and existing PoW schemes is given in Table 1 (on page 4).
2. We propose a novel construction of randomness extractor with large output size, which improves existing work [23] by reducing both the seed length and entropy loss (i.e. the difference between entropy of input and output) *simultaneously*. This new randomness extractor may have independent interest. A detailed comparison between our work and existing randomness extractors is given in Table 2 (on page 4).

1.3 Organizations

We introduce preliminaries and background in Section 2 and formulation in Section 3. We present our overall solution in a modular approach in Section 4 and Section 5: At first in Section 4, we propose the construction of Privacy-Preserving-PoW and analyze its security, by treating an important component (i.e randomness extractor) as black-box. Next, Section 5 constructs the required randomness extractor with rigorous analysis and completes the description of the proposed solution. Section 6 describes applications of PoW in constructing secure client-side deduplication scheme. Section 7 reports the experiment data. Section 8 concludes this paper.

2 Preliminaries and Background

2.1 Notations and Definitions

Key notations in this paper are defined in Table 3 (on page 5).

Table 3. Key Notations.

Notation	Semantics
λ	The security parameter.
PPT	Probabilistic polynomial time (w.r.t. security parameter λ, if not explicitly stated otherwise).
$[n]$	The set of integers $1, 2, 3, 4, \ldots, n$.
$\mathsf{h}(\cdot)$	Full domain collision resistant hash function (e.g. SHA256).
$F[i]$	The projection of bit-string F onto i-th coordinate (i.e. the i-th bit of F, $1 \le i \le \lvert F\rvert$).
$F[\{i_1, \ldots, i_n\}]$	The projection of bit-string F onto the subset of coordinates (i.e $F[i_1]\Vert F[i_2]\Vert \ldots \Vert F[i_n]$, where $1 \le i_1 < i_2 < \ldots < i_n \le \lvert F\rvert$).
$\mathbf{H}_\infty(X)$	min-entropy of random variable X.
$\mathsf{SD}(X, Y)$	Statistical difference between random variables X and Y.
$X \approx_\epsilon Y$	$\mathsf{SD}(X, Y) \le \epsilon$; X is ϵ-close to Y.
$B\vert_{A=a}$	The conditional distribution of B given that $A = a$ for jointly distributed random variables (A, B).
$x \sim \mathcal{D}$	Sample x according to distribution $\mathcal{D}$.
$U_{\lvert n\rvert}$	Independent uniform random variable over $\{0, 1\}^n$.
$U_{\lvert n\rvert,1}$, $U_{\lvert n\rvert,2}, \cdots$	Independently and identically distributed uniform random variables over $\{0, 1\}^n$.

Definition 1 (Statistical Difference) *The* statistical difference *between two random variables* $\mathbf{X}$ *and* $\mathbf{Y}$ *on the same space* $\mathcal{U}$ *is defined as*

$$\mathsf{SD}(\mathbf{X}, \mathbf{Y}) = \frac{1}{2} \sum_{a \in \mathcal{U}} \Big| \mathsf{Pr}[\mathbf{X} = a] - \mathsf{Pr}[\mathbf{Y} = a] \Big| \tag{1}$$

Some useful background information about statistical difference is provided in Appendix A.

2.2 Proofs of Retrievability

We adopt the formulation of proofs of retrievability from existing works [26,27] and make some syntactical modifications according to our needs to construct proofs of ownership scheme.

Definition 2 (Proofs of Retrievability) *A proofs of retrievability (POR) scheme consists of PPT algorithms* KeyGen, Tag, GenChal, GenProof *and* Verify*, which are described as below*

- $\mathsf{KeyGen}(1^\lambda) \to (pk, sk)$. *The key generation algorithm takes a security parameter* λ *as input and outputs a pair of public-private key* (pk, sk).
- $\mathsf{Tag}(sk, \{F_i\}_{i=1}^n) \to \{\sigma_i\}_{i=1}^n$. *The tag generation algorithm computes an authentication tag* σ_i *for each file block* F_i.
- $\mathsf{GenChal}(pk, n, c) \to (C, \Psi_F, \Psi_\sigma)$. *The challenger generation algorithm takes as input the public key pk, erasure encoded file size n (in term of blocks), and the sample size c, and outputs a sample* $C \subset [n]$ *with* $|C| = c$ *and meta-data* (Ψ_F, Ψ_σ).

- $\mathsf{GenProof}(pk, \{(F_i, \sigma_i)\}_{i=1}^n, C, \Psi_F, \Psi_\sigma) \to (\bar{F}, \bar{\sigma})$, *where* $\bar{F} := \mathsf{GenProof}_{\mathsf{data}}(pk, \{F_i\}_{i=1}^n, C, \Psi_F)$ *and* $\bar{\sigma} := \mathsf{GenProof}_{\mathsf{tag}}(pk, \{\sigma_i\}_{i=1}^n, C, \Psi_\sigma)$*. The algorithm* $\mathsf{GenProof}_{\mathsf{data}}$ *takes as input the public key* pk*, file blocks* F_i*'s, a sample set* $C \subset [n]$*, and meta-data* Ψ_F*, and outputs an aggregated file block denoted as* $\bar{F}$*. The algorithm* $\mathsf{GenProof}_{\mathsf{tag}}$ *takes as input the public key* pk*, authentication tags* σ_i*'s, a sample set* $C \subset [n]$*, and meta-data* Ψ_σ*, and outputs an aggregated authentication tag denoted as* $\bar{\sigma}$*.*
- $\mathsf{Verify}(K, \bar{F}, \bar{\sigma}, \Psi_F, \Psi_\sigma, C) \to$ `Accept` *or* `Reject`*. If* K *is private key* sk*, then the POR scheme supports private key verifiability; if* K *is public key* pk*, then the POR scheme supports public key verifiability.*

We remark that the above formulation is syntactically different from original [26,27] in the sense that we explicitly decompose the algorithm $\mathsf{GenProof}$ into two sub-routines: $\mathsf{GenProof}_{\mathsf{data}}$ and $\mathsf{GenProof}_{\mathsf{tag}}$, where $\mathsf{GenProof}_{\mathsf{data}}$ processes selected data blocks F_i $(i \in C)$ and $\mathsf{GenProof}_{\mathsf{tag}}$ processes corresponding authentication tags σ_i's. Many existing works (e.g. [26,27] and Merkle Hash Tree based POR) support such decomposition, but a few works (e.g. [18]) do not.

For some POR schemes [26,27], meta-data Ψ_F and Ψ_σ are two seeds from which a list of coefficients $\{\alpha_i\}_{i\in C}$, $\{\beta_i\}_{i\in C}$ can be generated, and the aggregated values are $\bar{F} = \sum_{i\in C} \alpha_i F_i$ and $\bar{\sigma} = \sum_{i\in C} \beta_i \sigma_i$.

2.2.1 Merkle Hash Tree based POR

For completeness, we restate the Merkle Hash Tree based POR scheme from the literature (e.g [11]), which will be used to construct privacy-preserving PoW later. $\mathsf{MHT\text{-}POR}$ consists of the following algorithms:

- $\mathsf{KeyGen}(1^\lambda)$: Choose a real value constant $\alpha \in (0,1)$, and a collision-resistant full domain hash function $\mathsf{h} : \{0,1\}^* \to \{0,1\}^\lambda$. Let public key $pk = (\alpha, \mathsf{h})$, and private key $sk = \mathsf{null}$.
- $\mathsf{Tag}(pk, \{F_i\}_{i=1}^n)$, where $(F_1, \ldots, F_n)$ is the rate-α erasure encoded version of user file F, $F_i \in \{0,1\}^\lambda$. Construct a Merkle Hash Tree $\mathsf{MHT}_{F,\mathsf{h}}$ with hash function h over n leaf nodes $F_1, F_2, \ldots, F_n$ in left-to-right and bottom-up manner. Let π denote the hash value associated to the root of the constructed Merkle Hash Tree. Let $\sigma_i = \pi$ for all i, i.e. all σ_i's are equal.
- $\mathsf{GenChal}(pk, n, c)$. Choose a random sample C of size c from $[n]$. Let $\Psi_F = \Psi_\sigma = \mathsf{null}$. Output (C, Ψ_F, Ψ_σ).
- $\mathsf{GenProof}_{\mathsf{data}}(pk, \{F_i\}_{i=1}^n, C, \Psi_F)$ where the meta-data $\Psi_F = \mathsf{null}$.
  For each $i \in C$:
    Find the i-th leaf node F_i and all sibling nodes along the unique path from the i-th leaf to the root of the Merkle Hash Tree $\mathsf{MHT}_{F,\mathsf{h}}$. Let Sib_i denote the ordered collection of hash values associated to these sibling nodes.
  Output $\{(i, F_i, \mathsf{Sib}_i) : i \in C\}$.
- $\mathsf{GenProof}_{\mathsf{tag}}(pk, \pi, C, \Psi_\sigma)$ where the meta-data $\Psi_\sigma = \mathsf{null}$. This algorithm simply outputs the root hash value π.
- $\mathsf{Verify}(pk, \{(i, F_i, \mathsf{Sib}_i) : i \in C\}, \pi, \mathsf{null}, \mathsf{null}, C)$. For each $i \in C$: Reconstruct the hash value associated to the root of the Merkle Hash Tree from (i, F_i, Sib_i) using hash function h. If all reconstructed root hash values are equal to π, then output `Accept`; otherwise output `Reject`.

In the above $\mathsf{MHT\text{-}POR}$, the size of authentication tags (only a root hash value π) is constant—λ bits, proof size is $\lambda(\log n+1)|C|$ bits, and challenge size is $|C|\log n$, where the challenge size can be significantly reduced using hitter sampler [28,29]. The distinctive property of Merkle Hash Tree based POR scheme is that there is no private key, which makes it a good choice to build privacy-preserving PoW in the bounded leakage setting.

Definition 3 (Soundness of POR [18,26,27]) *Let $\epsilon \in (0,1)$. A POR scheme is ϵ-sound, if there exists a PPT extractor algorithm, such that for any prover which can convince the verifier to accept with probability $\geq \epsilon$, then the extractor can output the original file with overwhelming high probability (1 - negl) by executing POR proof protocol with the prover.*

Readers may find more details about POR in [18,26,29,27].

2.3 Randomness Extractor

Definition 4 (Strong Extractor) *We say $\mathsf{Ext} : \{0,1\}^{\ell_{\mathsf{in}}} \times \{0,1\}^{\ell_s} \to \{0,1\}^{\ell_{\mathsf{out}}}$ is a strong (k,ϵ)-extractor, if for any distribution X over $\{0,1\}^{\ell_{\mathsf{in}}}$ with at least k bits min-entropy, the following inequality holds*

$$\mathsf{SD}\Big((\mathsf{Ext}(X;s), s),\ (U_{\ell_{\mathsf{out}}}, s) \Big) \leq \epsilon \tag{2}$$

where the seed s is uniformly randomly chosen from $\{0,1\}^{\ell_s}$ and $U_{\ell_{\mathsf{out}}}$ is a uniform random variable over $\{0,1\}^{\ell_{\mathsf{out}}}$.

It is well known that the output size $\ell_{\mathtt{out}}$ of any randomness extractor can not exceed the min-entropy k of the input (i.e. $\ell_{\mathtt{out}} < k$), and the difference $(k - \ell_{\mathtt{out}})$ is called the "entropy loss" of the randomness extractor.

3 Formulation: Proofs of Ownership, Revisited

Halevi *et al.* [11] proposed the formulation of proofs of ownership. In this section, we revisit and revise their formulation and propose our definition for privacy-preserving proofs of ownership.

Definition 5 (Proofs of Ownership [11]) *A proof of ownership scheme (*PoW*) consists of a probabilistic algorithm* S *and a pair of probabilistic interactive algorithm* $\langle \mathsf{P}, \mathsf{V} \rangle$*, which are described as below:*

- $\mathsf{S}(F, 1^\lambda) \to \psi$*: The randomized summary function* S *takes a file* F *and the security parameter* λ *as input, and outputs a short summary value* ψ*, where the bit-length of* ψ *is short and independent on file size* $|F|$*.*
- $\langle \mathsf{P}(F), \mathsf{V}(\psi) \rangle \to$ `Accept` or `Reject`*: The prover algorithm* P *which takes as input a file* F*, interacts with the verifier algorithm* V *which takes as input a short summary value* ψ*, and outputs either* `Accept` *or* `Reject`*.*

We are only interested in efficient PoW *scheme, such that* V *is polynomial time algorithm w.r.t. security parameter* λ *and both* S *and* P *are polynomial algorithms in* $|F|$ *and* λ*.*

Definition 6 (Completeness of PoW [11]) *A PoW scheme* $(\mathsf{S}, \langle \mathsf{P}, \mathsf{V} \rangle)$ *is* complete*, if for all positive integer* λ *and for any file* $F \in \{0,1\}^{poly(\lambda)}$*, it holds that*

$$\langle \mathsf{P}(F), \mathsf{V}(\mathsf{S}(F, 1^\lambda)) \rangle \textit{ always outputs } \mathtt{Accept}.$$

3.1 Two Players Setting and Three Players Setting of PoW

In the original framework [11], PoW runs by two players: verifier and prover. In this paper, we will redefine this system model by introducing a third player, called summarizer, who is responsible to preprocess the data file F during the setup. The PoW scheme in three players setting executes in this way: Summarizer (e.g. data owner of F) runs summary function to obtain $\psi := \mathsf{S}(F, 1^\lambda)$ and sends ψ to verifier (e.g. the cloud storage server). Then prover (e.g. some cloud user claiming to own file F), who runs algorithm $\mathsf{P}(F)$, interacts with the verifier, who runs algorithm $\mathsf{V}(\psi)$. A dishonest prover (e.g. dishonest cloud user) may replace the prover algorithm P with any other PPT program of his/her choice.

Definition 7 (Two/Three Players setting of PoW) *For any PoW scheme* $(\mathsf{S}, \langle \mathsf{P}, \mathsf{V} \rangle)$, *the two players setting and three players setting are described as below:*

- *in a* **two players setting***, the summary algorithm* S *and verifier algorithm* V *are executed by the first player—*verifier *(e.g. cloud storage server), and the prover algorithm* P *is executed by the second player—*prover *(e.g. cloud user);*
- *in a* **three players setting***, the summary algorithm* S *is executed by the first player—*summarizer *(e.g. cloud user owning file F), the verifier algorithm* V *is executed by the second player—*verifier *(e.g. cloud storage server), and the prover algorithm* P *is executed by the third player—*prover *(e.g. another cloud user claiming to own F).*

The difference between the two players setting [11] and our three players setting is that, execution of the summary function S moves from the verifier (cloud storage server) to a new player—summarizer (i.e. some cloud user). As a result, the verifier (cloud storage server) only runs algorithm V. We remark that the summary function S, which is polynomial in file length $|F|$, is typically much more expensive than the verifier algorithm V, which is polynomial in the security parameter λ. Therefore, our three players setting will further relieve the computation burden of the cloud storage server, and might make our scheme easier to be adopted by cloud storage servers in real applications—This is exactly our initial motivation to introduce the new three players setting of PoW. We will experimentally show that the extra computation burden on a cloud user is affordable. We believe that, the average computation resource that a cloud storage server allocates to each online user, is typically less than the computation resource of an average cloud user. Additionally, the fact that many cloud storage servers (e.g. Dropbox, Skydrive, and Google Drive) provide free service to public users, further justifies our attempt to shift some computation burden from cloud server to cloud user.

The change from two players setting to three players setting also leads to the change of trust model and thus impact the security formulation. In the original two players setting of PoW [11], preserving privacy of input file F during the interactive proof $\langle \mathsf{P}, \mathsf{V} \rangle$ (like in zero-knowledge proof) is meaningless, since the verifier, who runs V, also runs the summary function $\mathsf{S}(F, 1^\lambda)$ and has direct access to file F. Therefore, the verifier has to be trusted in data confidentiality of input file F in this two players setting. In contrast, in our three players setting, preserving privacy of F during the interactive proof $\langle \mathsf{P}, \mathsf{V} \rangle$ (like in zero-knowledge proof) is an interesting problem, if the verifier (e.g. cloud storage server) is not trusted in data confidentiality.

3.2 Soundness of PoW

Intuitively, PoW aims to prevent leakage amplification in client-side deduplication: If an outside adversary *somehow* obtain a bounded amount ($\leq T$ bits) of messages about the target user file F via out-of-band leakage, then the adversary cannot obtain the whole file F by participating in the client-side deduplication with the cloud storage server.

The security game $\mathsf{G}_{\mathcal{A}}^{\mathsf{PoW}}(k, T)$ between a PPT adversary $\mathcal{A}$ and a challenger w.r.t. PoW scheme $(\mathsf{S}, \langle \mathsf{P}, \mathsf{V} \rangle)$ is defined as below. Here k is the lower bound of min-entropy of the distribution of the challenged file F at the beginning of the game, and the adversary is allowed to learn at most T bits message related to file F (possibly including random coins chosen when processing F) from the challenger via the leakage query.

Setup. The description of $(\mathsf{S}, \langle \mathsf{P}, \mathsf{V} \rangle)$ is made public. Let $\mathcal{D}$ be a distribution over $\{0,1\}^M$ with min-entropy $\geq k$, where $\mathcal{D}$ is chosen by the adversary $\mathcal{A}$ and M is any public positive integer constant. The challenger samples file F according to distribution $\mathcal{D}$ and runs the summary algorithm to obtain $\psi := \mathsf{S}(F, 1^\lambda)$.

Learning. The adversary $\mathcal{A}$ can adaptively make polynomially many queries to the challenger, where each query is in one of the following types and concurrent queries of different types are not allowed[4]. Furthermore, the total amount of messages output by all leakage queries should not be greater than the threshold T, i.e. $\mathcal{Y}_{\text{I}} + \mathcal{Y}_{\text{II}} \leq T$, where $\mathcal{Y}_{\text{I}}$ and $\mathcal{Y}_{\text{II}}$ will be defined below.

- Prove-Query: The challenger, running the verifier algorithm V with input ψ, interacts with the adversary $\mathcal{A}$ which replaces the prover algorithm P, to obtain $b := \langle \mathcal{A}, \mathsf{V}(\psi) \rangle$. The adversary $\mathcal{A}$ is given the value of b.
- Leak-Query-I($\mathcal{P}$): This query consists of a description of a PPT algorithm $\mathcal{P}$ (a variant version of prover algorithm). The challenger responses this query by computing the output y of $\mathcal{P}(F)$ after interacting with $\mathsf{V}(\psi)$ (i.e. $y := \mathcal{P}(F)^{\mathsf{V}(\psi)}$) and sending y to the adversary $\mathcal{A}$. Denote with $\mathcal{Y}_{\text{I}}$ the sum of bit-lengths of all responses y's for this type of queries.
- Leak-Query-II($\mathcal{L}$): This query consists of a description of a PPT algorithm $\mathcal{L}$. Let $\mathsf{transcript}_{\mathsf{S}}$ denote the transcript of all steps of operations in the execution of algorithm "$\psi := \mathsf{S}(F, 1^\lambda)$" in the above **Setup** phase. The challenger responses this query by computing the output $y := \mathcal{L}(\mathsf{transcript}_{\mathsf{S}})$ and sending y to the adversary $\mathcal{A}$. Denote with $\mathcal{Y}_{\text{II}}$ the sum of bit-lengths of all responses y's for this type of queries.

Challenge. The adversary $\mathcal{A}$ which replaces the prover algorithm P, interacts with the challenger, which runs the verifier algorithm V with input ψ, to obtain $b := \langle \mathcal{A}, \mathsf{V}(\psi) \rangle$. The adversary $\mathcal{A}$ wins the game, if $b = \mathtt{Accept}$.

Definition 8 (Soundness of PoW (Refining [11])) *A* PoW *scheme is* (k, T, ϵ)*-sound in* three players setting, *if for any PPT adversary* $\mathcal{A}$, $\mathcal{A}$ *wins the security game* $\mathsf{G}_{\mathcal{A}}^{\mathsf{PoW}}(k, T)$ *with probability not greater than* $\epsilon + negl(\lambda)$.

$$\mathsf{Pr}[\mathcal{A} \textit{ wins the security game } \mathsf{G}_{\mathcal{A}}^{\mathsf{PoW}}(k, T)] \leq \epsilon + negl(\lambda). \tag{3}$$

[4] Concurrent Prove-Query and Leak-Query would allow the adversary to replay messages back and forth between these two queries, and eliminate the possibility of any secure and efficient solution to PoW. Therefore, the framework of Halevi *et al.* [11] do not allow concurrent queries of different types in the security formulation. We clarify that, concurrent queries of the same type can be supported. Thus, in the real application, the cloud storage server (verifier) can safely interact with multiple cloud users (prover) w.r.t. the same file concurrently.

The (k, T, ϵ)-soundness definition in two players setting *is the same as the above, except that the adversary $\mathcal{A}$ is not allowed to make* Leak-Query-II *in the security game $\mathsf{G}_{\mathcal{A}}^{\mathsf{PoW}}(k, T)$ (i.e. $\mathcal{Y}_{\mathrm{II}} = 0$).*

We remark that (1) the (k, T, ϵ)-soundness definition in two players setting is essentially the same as the original formulation [11], and (2) soundness in three players setting implies soundness in two players setting, but not vice versa.

3.3 Privacy-Preserving PoW

Intuitively, we say a PoW scheme is privacy-preserving against the verifier, if everything about file F that the verifier can learn after participating the PoW scheme w.r.t. F, can be computed from the short summary value of F and some almost-perfect uniform random number.

Definition 9 (Privacy-Preserving) *A PoW scheme $(\mathsf{S}, \langle \mathsf{P}, \mathsf{V} \rangle)$ is (k, T, ϵ)-privacy-preserving against the verifier (in the three players setting), if for any distribution $\mathcal{D}$ over $\{0,1\}^M$ with at least k bits min-entropy, for every PPT interactive algorithm V^*, there exists a PPT algorithm Sim and a random variable Z over domain $\{0,1\}^{T+\lambda+\Omega(\lambda)}$, such that*

- $\mathsf{SD}(Z, U_{|Z|}) \leq \epsilon$*, where $U_{|Z|}$ is the uniform random variable over $\{0,1\}^{|Z|}$;*
- *for any function $f : \{0,1\}^M \to \{0,1\}$, and any (leakage) function $\mathcal{L} : \{0,1\}^M \to \{0,1\}^{\leq T}$, the following two probabilities (taken over file $F \sim \mathcal{D}$ and the random coins of related algorithms) are equal*

$$\mathsf{Pr}\Big[\mathsf{V}^*\big(\psi \| \mathcal{L}(F)\big)^{\mathsf{P}(F)} = f(F)\Big] = \mathsf{Pr}\Big[\mathsf{Sim}\big(\psi \| \mathcal{L}(F), Z\big) = f(F)\Big],$$

where $\psi := \mathsf{S}(F, 1^\lambda)$ and $\mathsf{V}^*(\mathsf{S}(F, 1^\lambda) \| \mathcal{L}(F))^{\mathsf{P}(F)}$ denotes the output of (dishonest) verifier V^* taking the summary value $\mathsf{S}(F, 1^\lambda)$ and leakage information $\mathcal{L}(F)$ as input and having interaction with interactive prover algorithm $\mathsf{P}(F)$.

As we discussed before, preserving privacy against the verifier for any PoW scheme in the two players setting, is impossible.

3.3.1 Why privacy-preserving property is necessary PoW scheme can be deployed together with convergent encryption (or message-locked encryption) in the below way [30] to achieve secure client side deduplication over encrypted data in cloud storage: The owner of file F who firstly uploads F to the cloud storage, runs the summary function $\mathsf{PoW.S}(F, 1^\lambda)$ on input file F to generate a summary value ψ, and encrypts F using convergent encryption[5] or message-locked encryption to generate a ciphertext C_F. Then the first uploader sends both ciphertext C_F and summary value ψ to the cloud storage server. Later, any other owner of F who tries to upload F, will be notified by the cloud storage server that F is already in the cloud. Then this subsequent uploader is supposed to run the prover algorithm $\mathsf{PoW.P}(F)$ and interacts with the cloud storage server, who runs the verifier algorithm $\mathsf{PoW.V}(\psi)$. If

[5] In convergent encryption or message-locked encryption, the encryption key is deterministically generated from the plaintext, and the encryption method is deterministic.

$\mathsf{PoW.V}(\psi)$ returns `Accept`, then the cloud storage server believes that this subsequent uploader is indeed an owner of file F and allows him/her to access the unique copy of F in the cloud storage. In this way, the client side deduplication achieves in bounded leakage environment. We remark that *target collision attack* [8] or *poison attack* [20] can be prevented in the above client side deduplication scheme, but details are saved, since our focus is PoW scheme instead of client side deduplication scheme.

Unfortunately, in the above application of PoW, the privacy preserving property of PoW scheme may not be necessary, since one can achieve the same security goal by applying the non-privacy-preserving PoW scheme over the ciphertext C_F instead of applying privacy-preserving PoW scheme over F.

A natural question is that, beyond the generic theoretic interest (like the research of zero-knowledge proof), is the concept of privacy-preserving PoW necessary in real world application? Here we give an affirmative answer to the above question by presenting an application scenario where privacy-preserving PoW is necessary.

At first, we describe a simple example of "server-aided message-locked encryption" given in [21]: Owner of file F sends hash value $\mathsf{hash}(F)$ to a key-server, and the key-server responses it with $k_F := \mathsf{PRF}_K(\mathsf{hash}(F))$ where K is the secret key of the key-server. Then the owner encrypts file F using a deterministic encryption method with k_F as encryption key, and sends the resulting ciphertext C_F to the cloud storage server. If C_F is already in the cloud storage server, the transmission of C_F to the cloud server could be saved due to client-side deduplication mechanism. Here, a new key-server with secret key K is introduced, in order to prevent *offline* brute force attack in ciphertext-only attack (COA) model on traditional convergent encryption or message-locked encryption.

To apply the above server-aided message-locked encryption to achieve client side deduplication over encrypted data in the bounded leakage setting, the owner of F has to convince the key-server that he/she indeed owns file F, in a privacy-preserving manner (i.e. apply privacy-preserving PoW), otherwise the key server should not return the encryption key k_F. Here non-privacy-preserving PoW over ciphertext C_F cannot work, because the file owner does not have the encryption key k_F and is not able to generate the ciphertext C_F when this file owner runs PoW with the key-server. We save the details in the full paper.

3.4 Clarification on Leakage of User ID and Password

We admit that, as the same as Halevi *et al.* [11], this work will consider leakage of user account (i.e. id and password) as out of scope. We assume the user account is associated to user's real identity (e.g. mobile phone number) and sibyl account is hard to create. Thus, leakage of user file stored in cloud storage by disclosure of user account could be traced back to the source and the corresponding account could be disabled without affecting honest users.

4 Generic Construction of Proofs of Ownership

4.1 Some Unsatisfactory Approaches

At first, putting privacy-preserving property aside, we review some straightforward approaches and existing works for PoW as below.

4.1.1 Compute fresh MACs online on both sides

In the summary phase, let the summary value ψ equal to file F. In the proof phase, both prover and verifier have access to the file F, and per each proof session compute a MAC (i.e. Message Authentication Code) value over F with a random nonce as key, where the random nonce is chosen by the verifier. This approach is secure, but rejected for two reasons: (1) in some applications of PoW, the verifier does not have access to the file F (e.g. the verifier who is the key server in the "server-aided message-locked encryption"); (2) it does not satisfy the stringent requirement on efficiency (including disk IO efficiency). The framework of Halevi *et al.* [11] only allows the verifier to access a summary (or aggregated) value during a proof session, where the summary value is generated from the file F in the setup phase and should be much shorter than the file itself. The reason behind is that, although cloud storage server has more computation resource than an average cloud user, the average computation resource allocated to each online user by the cloud server could be smaller than an average cloud user's computation resource.

4.1.2 Pre-compute MACs offline

In the summary phase, t number of keys $s_1, \ldots, s_t$ are randomly chosen and t number of MAC values $\mathsf{MAC}_{s_i}(F)$'s are computed correspondingly. The summary value of file F is $\{(i, s_i, \mathsf{MAC}_{s_i}(F)) : i \in [t]\}$. In the proof phase, the verifier keeps a counter state variable c, such that for each $i \geq \mathsf{c}$, the key s_i has not been sent to any prover. Once some prover initiates a new proof session, the verifier sends the unused key s_{c} to the prover as challenge and anticipate the correct response $\mathsf{MAC}_{s_{\mathsf{c}}}(F)$. No matter this prover passes the challenging or not, the verifier will increment the counter state c by one, in order to ensure that each key s_i will be used for at most once[6].

If t is relatively small (say $t < 1000$), then the above approach is efficient in both storage and computation. *Wishfully*, the above approach seems to be able to support t number of owners of F in cloud storage service. By estimating a proper upper bound on the number of owners of the same file, this approach might work well in most cases. However, this approach is actually not secure in the setting of PoW [11], since a single malicious adversary could consume up all of t pre-computed MACs easily by impersonating or colluding with t distinct cloud users.

4.1.3 Proofs of Retrievability

Some instance of POR (e.g. [26,31,29]) can serve as PoW. The first construction (i.e. PoW1 as in Table 1) of Halevi *et al.* [11] is just the Merkle Hash Tree based POR scheme (MHT-POR), which combines error erasure code and Merkle Hash Tree proof method[7]. The drawback of this approach is that, the relatively expensive error erasure code[8] is applied over the whole input file, while in our approach, error erasure code is applied over the output of the randomness extractor, which is much shorter than the whole input file.

[6] This is essential to achieve security in the bounded leakage model.

[7] Merkle Hash Tree proof method proves the correctness of a leaf value by presenting as a proof all sibling values along the path from the questioned leaf to the root of Merkle Hash Tree, and verification requires only the root value.

[8] In typical usage of error erasure code, block length is some small constant (say 223 bytes for (255, 223)-reed-solomon code). However, in the usage of POR, the block length has to be as large as the input file, which makes the coding much slower than typical case.

We notice that recent work by Zheng and Xu [32] attempts to equip proofs of storage (POR or PDP) with deduplication capability. However, their work is not in the leakage setting of Halevi *et al.* [11].

4.1.4 Pairwise-Independent Hash with Large Output Size

The second construction of PoW in Halevi *et al.* [11] is based on pairwise independent hash family (a.k.a 2-independent or 2-universal hash family). A large input file is hashed into a constant size (say about $3T = 3 \times 64\texttt{MB}$) hash value and then apply the merkle hash tree proof method over the hash value. This construction is secure, but very in-efficient in both computation and randomness complexity. Furthermore, large random seed also implies large communication cost required to share this seed among all owners of the same file. It is worth pointing out that Halevi *et al.* [11] overlooked the disadvantage in large randomness complexity (i.e. at least twice of hash output size, say about $2 \times 3T = 6 \times 64\texttt{MB}$), although they admitted that this construction is *prohibitively* expensive in computation for practical data size.

A quick thought to reduce the seed length is to apply pseudorandomness generated from a short true random seed. However, in the leakage setting of PoW, any short seed could be leaked to the adversary by some colluded owner of target file. Consequently, the standard computational indistinguishability argument of pseudorandom number generator (or pseudorandom functions) is not applicable. It is unclear whether this pseudorandomness approach works or not without new sophisticated proof (or disproof). Similar issue is discussed in the study of proofs of retrievability by Dodis *et al.* [29], which adopts sampling technique with public coin as seed to replace pseudorandomness.

We clarify that, in its appearance, the second construction of PoW in Halevi *et al.* [11] is a combination of universal hash and Merkle Hash Tree proof method and *sounds to* fit into our generic approach: `PoW = Randomness Extractor + Proofs of Retrievability`, since pairwise independent hash family is an instance of randomness extractor if in the proper use (see leftover hash lemma [25,33]) and MHT can be used to construct proofs of retrievability scheme. Unfortunately, their second construction cannot be considered as an (either explicit or implicit) instance of our generic approach for the below reasons: (1) they chooses a universal hash family with output size much larger (about 3 times larger) than k—the bound of min-entropy in the input file, which will render the hash output to deviate from uniform randomness. In other words, in their parameter setting, the universal hash family is no longer a (strong) randomness extractor; (2)Without error erasure encoding, Merkle Hash Tree proof method alone is not a POR scheme. By our understanding, the authors choose hash family with output size much larger than k on purpose, in order to compensate the omission of error erasure code. As a direct consequence of the first point (1) above, Halevi *et al.* [11] had to provide a particular customized (thus not general) proof instead of leveraging on the well-known leftover hash lemma [33], which characterizes the randomness extractor property of universal hash. Therefore, to the best of our knowledge, our work is the first to bridge PoW and randomness extractor.

4.1.5 PoW with respect to Particular Distribution

The third construction of PoW in Halevi *et al.* [11] is the most efficient one among all of three constructions proposed by Halevi *et al.* [11]. In the third construction, the size of random seed is dramatically reduced by treating hash function `SHA256` as a random oracle. However, their proof (in random oracle model) of this construction is incomplete: firstly, the distribution of input file is restricted as "generalized bit/block-fixing distribution"[9]; secondly, their proof assumes their algorithm will generate a "good linear code" and the authors admit that it is "very hard to analyze" this unproven assumption (See texts around Theorem 3 in [11]).

We emphasize that, information leakage of file F may have different forms. For example, some plain bits $F[i]$'s are leaked, or some aggregated information of file F (e.g. a hash value) is leaked. In the latter case, file F is hardly considered as fitting in (generalized) fixed-bit/block distribution.

Gabizon *et al.* [34] proposed a randomness extractor for input under bit-fixing distribution. Such extractor can be combined with our generic construction to obtain a secure PoW scheme for bit-fixing input file and with complete security proof in standard model.

Other works on deduplication/PoW include Pietro and Sorniotti [35], which treats a projection $(F[i_1], \ldots, F[i_\lambda])$ of file F onto λ randomly chosen bit-positions $(i_1, \ldots, i_\lambda)$ as the "proof" of ownership of file F. Similar to the "hash-as-a-proof" method, this work is extremely efficient but insecure in the bounded leakage setting [11]. Readers may find more related works in Xu *et al.* [20].

4.2 Our approach: $\mathsf{PoW} = \mathsf{Randomness\ Extractor} + \mathsf{POR}$

Intuitively, our generic construction extracts $(T + 2\lambda)$ bits message Y from the input file F and then apply a proofs of retrievability scheme over Y. It is worth noting that in our usage of proofs of retrievability scheme, algorithm $\mathsf{POR.GenProof_{data}}$ runs by prover and algorithm $\mathsf{POR.GenProof_{tag}}$ runs by verifier[10], while in the literature [18,26,27], both of these two algorithms run by prover. It is easy to see that, such modification will preserve the soundness of POR scheme.

The detailed construction is given in Figure 1 (on page 15). Before presenting a formal statement in Theorem 2 for the PoW scheme in Figure 1 which constructed from a generic randomness extractor algorithm and a generic POR scheme, we will prove a stronger result in Theorem 1 for the special case that the POR scheme is instantiated with MHT-POR scheme in the construction of PoW. The reason that MHT-POR can achieve a stronger result is that, the security of MHT-POR relies on the cryptographic one-way function without trapdoor (precisely the collision resistance hash function). In contrast, most other POR schemes rely on cryptographic trapdoor one-way function (e.g. factorization), and such *short* trapdoor (or private key) might be leaked via some colluded file owner in our stringent security model in three player setting. Once the short trapdoor is leaked to the adversary, the POR scheme can be easily broken.

Theorem 1 *Suppose* $\mathsf{Extractor} : \{0,1\}^M \times \{0,1\}^{\ell_s} \to \{0,1\}^{T+2\lambda}$ *is a strong* (k, ϵ)*-extractor, and the POR scheme is the Merkle Hash Tree based scheme* $\mathsf{MHT\text{-}POR}$ *(as described in*

[9] A M bits long file F with k bit entropy under "generalized bit-fixing distribution" is generated in this way: (1) Independently choosing k uniform random bits; (2) deriving all other $(M-k)$ bits from these k random bits (Halevi *et al.* [11] applies linear transformation); (3) the file F is a random permutation of these k random bits and $(M-k)$ derived bits. If in the above step (2), all $(M-k)$ bits are constant, then the resulting distribution is called "bit-fixing distribution" with entropy k.

[10] All tag values are stored with the verifier instead of the provers, in order to prevent any potential leakage of partial information of Y from its tag values to the (dishonest) provers.

Fig. 1. PoW = RE + POR: A Generic Construction of PoW using Randomness Extractor $\mathsf{Extractor}(\cdot\ ;\ \cdot)$ and POR scheme ($\mathsf{KeyGen}, \mathsf{Tag}$, $\mathsf{GenChal}$, $\mathsf{GenProof}_{\mathsf{data}}$, $\mathsf{GenProof}_{\mathsf{tag}}, \mathsf{Verify}$). The completeness of the constructed PoW scheme is straightforward.

$\mathbf{S}(\boldsymbol{F}, \mathbf{1}^{\boldsymbol{\lambda}})$ Summary function.

Input: An M-bit file $F \in \{0,1\}^M$ and security parameter λ in unary form.

Extract: Choose random seed s from domain $\{0,1\}^{\ell_s}$ and compute $Y := \mathsf{Extractor}(F; s)$.

Expand: Apply Erasure-Correcting-Code on Y to obtain $\hat{Y} = (\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_n)$ such that Y can be completely recovered from any αn blocks among $\{\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_n\}$, where constant $\alpha \in (0,1)$ is some system parameter. Generate POR-key pair $(pk, sk) := \mathsf{POR.KeyGen}(1^\lambda)$, and authentication tags $\{\sigma_i\}_{i=1}^n := \mathsf{POR.Tag}(sk, \{\hat{Y}_i\}_{i=1}^n)$. Let $\pi_F = (pk, sk, \{\sigma_i\}_{i=1}^n)$.

Note: As mentioned in [11], in the construction of PoW, the decoding algorithm of the above Erasure-Correcting-Code is not required to be practical, since the decoding algorithm will not be invoked in the legitimate application of PoW.

Output: The summary value of file F is $\psi = (s, \alpha, \pi_F)$. Output ψ.

$\langle \mathbf{P}(\boldsymbol{F}), \mathbf{V}(\boldsymbol{\psi}) \rangle$ Interactive proof system between verifier (cloud storage server) and prover (cloud storage client).

Input: The prover has file F as input and the verifier has a summary value $\psi = (s, \alpha, \pi_F)$ as input, where $\pi_F = (pk, sk, \{\sigma_i\}_{i=1}^n)$.

V1: Verifier finds $c = \lceil \log_{1-\alpha} \epsilon \rceil$ (i.e. c is the smallest integer such that $(1-\alpha)^c \leq \epsilon$) and computes $(C, \Psi_F, \Psi_\sigma) := \mathsf{POR.GenChal}(pk, n, c)$. Verifier sends $(C, s, \alpha, pk, \Psi_F)$ to the prover.

P1: Prover runs the extractor algorithm to obtain $Y := \mathsf{Extractor}(F; s)$, and re-generate the erasure code $\hat{Y}$ from Y using the same Erasure-Correcting-Code with the same parameter α. Prover divides $\hat{Y}$ into n blocks $\hat{Y}_1, \ldots, \hat{Y}_n$ and computes $\bar{F} := \mathsf{POR.GenProof}_{\mathsf{data}}(pk, \{\hat{Y}_i\}_{i=1}^n, C, \Psi_F)$. Prover sends $\bar{F}$ to verifier.

V2: Verifier computes $\bar{\sigma} := \mathsf{POR.GenProof}_{\mathsf{tag}}(pk, \{\sigma_i\}_{i=1}^n, C, \Psi_\sigma)$ and $b := \mathsf{POR.Verify}(K, \bar{F}, \bar{\sigma}, \Psi_F, \Psi_\sigma) \in \{\texttt{Accept}, \texttt{Reject}\}$, where K is pk if the POR scheme supports public key verification; otherwise K is sk.

Output: Output $b \in \{\texttt{Accept}, \texttt{Reject}\}$.

Note: The subset C requires $|C| \log n$ bits communication cost. We can reduce this communication cost by using Goldreich [28]'s (δ, γ)-hitter sampler[a] *to represent C compactly with only $\log n + 3\log(1/\gamma)$ bits of public random coins.*

[a] Goldreich [28]'s (δ, γ)-hitter guarantees that, for any subset $W \subset [1, n]$ with size $|W| \geq (1-\delta)n$, $\Pr[C \cap W \neq \emptyset] \geq 1 - \gamma$. Readers may refer to [28,29] for more details.

Section 2.2.1), which is ϵ-sound. Then the PoW scheme constructed in Figure 1 is (k, T, ϵ)-sound and (k, T, ϵ)-privacy-preserving in the three players setting.

Proof. This proof consists of two parts, one for soundness and the other for privacy-preserving.
Soundness part. The soundness of a PoW scheme is defined with a security game $\mathsf{G}_{\mathcal{A}}^{\mathsf{PoW}}$ in Definition 8. In the game, the adversary $\mathcal{A}$ is allowed to obtain at most T bits message about file F via Leak-Query-I and Leak-Query-II. The adversary $\mathcal{A}$, playing the role of prover, can only learn $(C, s, \alpha, pk, \Psi_F)$ in the Prove-Query and nothing else, where (s, α, pk) are public information, and (C, Ψ_F) are generated from public information (n, c, pk). Since $\mathsf{Extractor}$ is strong extractor, its output Y is ϵ-close to uniform randomness even if the seed s is made public. According to Lemma 13 in the Appendix A, conditional on adversary $\mathcal{A}$'s (at most) T bits message of F, Y has at least $(T + 2\lambda) - T - \lambda = \lambda$ bits min-entropy with overwhelming high probability $(1 - 2^{-\lambda})$. Therefore, the adversary $\mathcal{A}$ cannot output Y with probability larger than $2 \cdot 2^{-\lambda}$. On the other hand, the adversary $\mathcal{A}$ cannot obtain any short

trapdoor with which $\mathcal{A}$ can break the MHT-POR scheme, since such trapdoor does not exist in the MHT-POR scheme: MHT-POR relies on collision-resistant hash function (i.e SHA256) and has no any private key, i.e. $sk = \mathsf{null}$. Consequently, the subroutine MHT-POR over Y will reject the adversary with significantly high probability (i.e. $> 1 - \epsilon \geq 1 - (1-\alpha)^{|C|}$).

Privacy-Preserving part. Proof of this part is more straightforward. For each file F, set the random variable Z, as stated in Definition 9, to be equal to $Y = \mathsf{Extractor}(F; s)$. Since the variable file $F \sim \mathcal{D}$ has at least k bits min-entropy, by the property of (k, ϵ)-extractor Extractor, $\mathsf{SD}(Z, U_{|Z|}) \leq \epsilon$ where $U_{|Z|}$ is a uniform random variable over $\{0,1\}^{|Z|}$. We design the PPT simulator algorithm $\mathsf{Sim}(\psi \| \mathcal{L}(F), Z := Y)$ as below:

1. The input consists of the summary value ψ of file F and the extracted randomness Y.
2. Simulate the (dishonest) verifier by simply revoking algorithm V^* on input $\psi \| \mathcal{L}(F)$.
3. Simulate the (honest) prover P by carrying out Step **P1** in Figure 1 with information Y and without knowing F.
4. Let the simulated verifier to interact the simulated prover, while the simulated prover will honestly follow the protocol as in Figure 1.
5. Set the output of this simulator as the same as the output of the invoked algorithm V^*.

Therefore, given any input, the output of simulated verifier and prover are *identically* distributed as that of real verifier and prover. Consequently, for any function $f : \{0,1\}^{|F|} \to \{0,1\}$, and for any leakage function $\mathcal{L} : \{0,1\}^{|F|} \to \{0,1\}^{\leq T}$, we have the following equation as desired

$$\mathsf{Pr}\Big[\mathsf{V}^*\big(\mathsf{S}(F, 1^\lambda) \| \mathcal{L}(F)\big)^{\mathsf{P}(F)} = f(F)\Big] = \mathsf{Pr}\Big[\mathsf{Sim}\big(\mathsf{S}(F, 1^\lambda) \| \mathcal{L}(F), Z\big) = f(F)\Big]. \qquad (4)$$

At last, the size of Z is $|Z| = |Y| = T + 2\lambda = T + \lambda + \Omega(\lambda)$, as desired. □

One can see that, the above soundness proof only requires to detect possible data loss in file Y and does not require to actually recover the original file Y, thus other proof of storage methods like Provable Data Possession (PDP) scheme [19,36], can also be adopted. Most POR (PDP) schemes [26,27][19] require a short private key (e.g. the factorization of a RSA modulus, the secret key of some pseudorandom function) to work and thus cannot resist Type-II leak query Leak-Query-II, from which the adversary could learn the short private key and break the POR scheme. Therefore, for such POR schemes with private key, we have to disable Type-II leak query by switching to the two players setting as below.

Theorem 2 *Suppose* Extractor : $\{0,1\}^M \times \{0,1\}^{\ell_s} \to \{0,1\}^{T+2\lambda}$ *is a strong* (k, ϵ)*-extractor and* POR *is an* ϵ*-sound POR scheme. Then the PoW scheme constructed in Figure 1 is* (k, T, ϵ)*-sound in the two players setting. (Details of proof is saved)*

We compare two instantiations of our generic approaches in Table 4 (on page 17).

5 Randomness Extractor with Large Output Size

In this section, we propose in Figure 2 (on page 18) a novel randomness extractor with large output size using the well-known "sample-then-extract" approach: Repeatedly sample

Table 4. Two instantiations of PoW=RE+POR.

Choice of POR	Setting	Summary Value Size (bits)	Communication cost (bits)
MHT-POR	2P,3P	λ	$\lambda \cdot \log_{1-\alpha} \epsilon \cdot \log{(T/\alpha)}$
Brent-Waters-POR [26]	2P	$T/(\alpha s)$ †	$(s+3)\lambda + 440$

† : s is a system parameter of POR [26] and can take any positive integer value.

a subset of bits from a weak random source and then apply an existing extractor with small output size over the sample.

Intuitively, the sampling lemma [23,24] states that "if one samples a random subset of bits from a weak random source, the min-entropy rate (i.e. ratio of min-entropy to bit-length) of the source is nearly preserved". Precisely if $X \in \{0,1\}^n$ has δn min-entropy and $X[S] \in \{0,1\}^t$ is the projection of X onto a random set $S \subset [n]$ of t positions, then with high probability, $X[S]$ is statistically close to a random variable with $\delta' t$ min-entropy. We consider the difference $(\delta t - \delta' t)$ as the entropy loss in sampling t bits. Nisan and Zuckerman(Lemma 11 in [23]) gave a sampling algorithm where $\delta' = c\delta / \log(1/\delta)$ for some small positive constant c. Vadhan (Lemma 6.2 in [24]) improved their result and allows $\delta' = (\delta - 3\tau)$ for sufficiently small positive constant τ.

We brief the existing approach [23,37] as below: (1) Independently and randomly choose l number of seeds, in order to get l samples $X_1, \ldots, X_l$ from the input weak source F, which has min-entropy rate δ. (2) Show that $(X_1, \ldots, X_l)$ is a δ'-block-wise source with δ' close to δ, i.e. for each $i \in [l]$, conditional on $(X_1, \ldots, X_i)$, the random variable X_{i+1} has min-entropy rate at least δ'. (3) Apply existing randomness extractor on the *structured* weak random source $(X_1, \ldots, X_l)$ to generate almost-uniform random output $(y_1, \ldots, y_l)$.

Roughly speaking, in the analysis of the above approach in [23,37], to extract each block y_i, the remaining min-entropy of the input F reduces by $|X_i|$ bits—the bit-length of X_i. Unlike previous works [23,24,37], we do not generate block-wise source as intermediate product, and manage to show that the remaining min-entropy of the input F, after extracting each block y_i, reduces by $|y_i|$ bits—the bit-length of y_i which is much smaller than $|X_i|$. Readers may find definition and calculation of remaining (or conditional) min-entropy $\tilde{\mathbf{H}}_\infty(A|B)$ of variable A given variable B in Lemma 12 and Corollary 13 in Appendix A. In this jargon, we manage to switch the conditional variable B from X_i (as previous works) to y_i in the analysis of our new design.

Theorem 3 *Let $t = M^c$ and $\tau = M^{-c}$ for constant $c \in (0,1)$. Let* Ext *: $\{0,1\}^{t+256} \times \{0,1\}^{r_1} \to \{0,1\}^{\rho}$ be a strong (k_0, ϵ_0)-extractor. Let* Samp *be an (μ, θ, γ)-averaging sampler [24,37]. Then the algorithm* $\mathsf{Extractor}$ *: $\{0,1\}^M \times \{0,1\}^{\rho} \to \{0,1\}^{\rho\ell}$ constructed in Figure 2 is a (k_1, ϵ_1)-extractor, where $\rho = \lambda + \log(M/t) + \log(1/\gamma) \cdot poly(1/\theta)$, $\rho \cdot \ell = k_1 - (k_0 + 3)M^{1-c}$, and $\epsilon_1 = 5\ell(\epsilon_0 + \gamma + 2^{-\lambda} + 2^{-\Omega(\tau M)})$. Here $poly(1/\theta) = O(\theta^{-2.001})$ according to [24].*

We make the following remarks: (1) Our algorithm in Figure 2 requires about $1/\ell$ fraction of the amount of random bits required by [23], since [23] requires that all of sampling seeds $s_1, s_2, \ldots, s_\ell$ should be independent randomness. (2) The choice of value $t = M^c$ ensure that there will be sufficient remaining min-entropy in the last sample (worst case), and this value of sample size t would be much larger than required for the first few samples (good cases). One may use different sample size t_i for the i-th sample ($t_1 < t_2 < t_3 \ldots < t_\ell = M^c$), in

Fig. 2. A Novel Randomness Extractor with Large Output Size and Short Seed. Ext is some existing strong randomness extractor and Samp is some existing sampling algorithm.

```
Extractor(F; s, s') This extractor algorithm will serve as a subroutine to construct PoW scheme.
  Input: An $M$-bit file $F \in \{0,1\}^M$; $s \in \{0,1\}^{r_0}$ and $s' \in \{0,1\}^{r_1}$ are true random seeds,
      where $r_0 + r_1 = \rho$.
  Sample-then-Extract-Loop:
      Let $s_1 := s$ and $s'_1 := s'$. Let $h_F := \mathtt{SHA256}(F)$ with $|h_F| \le \rho$.
      For each $i$ from 1 to $\ell$:
      Sample: Independently and randomly sample $t$ distinct indices from the set $[M]$,
          using random seed $s_i$, to obtain $S_i := \mathsf{Samp}([M], t;\ s_i) \subset [M]$.
      Extract: Compute $y_i := \mathsf{Ext}(h_F \| \ F[S_i];\ s'_i) \in \{0,1\}^{\rho}$. Let $s_{i+1}$ be the prefix of bit-
          length $r_0$ of bit-string $y_i$, and $s'_{i+1}$ be the suffix of bit-length $r_1$ of bit-string $y_i$.
          Note: The hash value $h_F$ is added into the input of Ext, in order to ensure that any change in file
          $F$ will lead to significant change in the output of randomness extractor.
  Output: Let $Y := y_1 \| y_2 \| \ldots \| y_\ell \in \{0,1\}^{\rho\ell}$. The output is $Y$.
```

order to reduce the IO reading. (3) Alternatively, we may choose hitter-sampler [28] as in [23] instead of averaging sampler, in order to reduce the seed length ρ (only $\mathcal{O}(\lambda + \log M)$ bits) at the cost of larger value of t. (4) In practice, one may use Tabulation Hashing [38] or CBC-MAC or HMAC as the underlying extractor algorithm Ext (possibly in the companion with hitter sampler which allows small ρ), as analyzed by Dodis *et al.* [39].

To prove Theorem 3, we introduce Lemma 4 and Lemma 6.

Lemma 4 (Amplification) *Suppose the algorithm* $\overline{\mathsf{Ext}} : \{0,1\}^M \times \{0,1\}^\rho \to \{0,1\}^\rho$ *defined as*

$$\overline{\mathsf{Ext}}\big(X; (s, s')\big) \overset{\text{def}}{=} \mathsf{Ext}\Big(\mathtt{SHA256}(X) \ \| \ X[\mathsf{Samp}(s)];\ s'\Big) \tag{5}$$

is a strong (k_2, ϵ_2)*-extractor. Then* Extractor : $\{0,1\}^M \times \{0,1\}^\rho \to \{0,1\}^{\rho\ell}$ *constructed in Figure 2 is a* (k_1, ϵ_1)*-extractor, where* $k_1 \ge k_2 + \rho(\ell - 1) + \lambda$ *and* $\epsilon_1 = 5\ell(\epsilon_2 + 2^{-\lambda})$.

Our proof for Lemma 4 is an analog of *hybrid proof technique* for (computational) indistinguishability [40].

Proof (Proof of Lemma 4). Define a (deterministic) function Nest:

$$\mathsf{Nest}(F, s_0, n) \overset{\text{def}}{=} (s_1, s_2, \ldots, s_n),$$

where $s_i := \overline{\mathsf{Ext}}(F, s_{i-1})$ for each $i \in [n]$.

We have identity:

$$\mathsf{Nest}(F, s_0, n) \equiv \Big(\overline{\mathsf{Ext}}(F, s_0),\ \mathsf{Nest}(F, \overline{\mathsf{Ext}}(F, s_0), n - 1)\Big). \tag{6}$$

For each $i \in [\ell]$, define random variable

$$W_i \overset{\text{def}}{=} (w_1, \ldots, w_i, \mathsf{Nest}(F, w_i, \ell - i)) \in \{0,1\}^{\rho\ell},$$

where for each $j \in [i], w_j := \overline{\mathsf{Ext}}(F, U_{|\rho|,j})$ and $U_{|\rho|,j}$'s are independent and uniform random variables over $\{0,1\}^\rho$. Notice that W_1 represents the output of our construction of extractor with seed $U_{|\rho|,1} = s\|s'$ in Figure 2. Let $U_{|\rho\ell|}$ be uniform random variable over $\{0,1\}^{\rho\ell}$. Lemma 4 is equivalent to the following lemma:

Lemma 5 $\mathsf{SD}\big((U_{|\rho|,1}, W_1),\ (U_{|\rho|,1}, U_{|\rho\ell|})\big) \leq 5\ell(\epsilon_2 + 2^{-\lambda})$.

The above Lemma 5 can be derived directly from the following Claim 1 and Claim 2 using the triangle inequality of statistical difference (Lemma 8 in Appendix A).

Claim 1 $\mathsf{SD}\big((U_{|\rho|,1}, W_\ell),\ (U_{|\rho|,1}, U_{|\rho\ell|})\big) \leq \ell(\epsilon_2 + 2^{-\lambda})$. *(Proof is in Appendix B)*

Claim 2

- *For any* $i \in [\ell-1]$, $\mathsf{SD}\big((U_{|\rho|,1}, W_i),\ (U_{|\rho|,1}, W_{i+1})\big) \leq 4(\epsilon_2 + 2^{-\lambda})$.
- $\mathsf{SD}\big((U_{|\rho|,1}, W_1), (U_{|\rho|,1}, W_\ell)\big) \leq 4\ell(\epsilon_2 + 2^{-\lambda})$. *(Proof is in Appendix B)*

The proof of Lemma 4 completes. □

Lemma 6 (Theorem 6.3 [24], sample-then-extract) *Let* $1 \geq \overline{\delta} \geq 3\tau > 0$. *Suppose that* $\mathsf{Samp} : \{0,1\}^{r_0} \to [M]^t$ *is an* (μ, θ, γ) *averaging sampler with distinct samples for* $\mu = (\overline{\delta} - 2\tau)/\log(1/\tau)$ *and* $\theta = \tau/\log(1/\tau)$ *and that* $\mathsf{Ext} : \{0,1\}^{t+256} \times \{0,1\}^{r_1} \to \{0,1\}^\rho$ *is a strong* $(k_0 = (\overline{\delta} - 3\tau)t,\ \epsilon_0)$*-extractor. Let* $\rho = r_0 + r_1$ *and define* $\overline{\mathsf{Ext}} : \{0,1\}^M \times \{0,1\}^\rho \to \{0,1\}^\rho$ *by*

$$\overline{\mathsf{Ext}}(X; (s, s')) \overset{\text{def}}{=} \mathsf{Ext}\Big(\mathtt{SHA256}(X) \ \|\ X[\mathsf{Samp}(s)];\ s'\Big) \tag{7}$$

Then $\overline{\mathsf{Ext}}$ *is a strong* (k_2, ϵ_2)*-extractor with* $k_2 = \overline{\delta}M$ *and* $\epsilon_2 = \epsilon_0 + \gamma + 2^{-\Omega(\tau M)}$. *Note: As mentioned in [24],* τ *could be arbitrarily small and approaches 0. In this paper, we set* $\tau = M^{-c}$ *for some constant* $c \in (0,1)$.

Now with Lemma 4 and Lemma 6 available, we are ready to prove Theorem 3.

Proof (Proof of Theorem 3). This theorem is directly implied by Lemma 4 and Lemma 6, except only one missing part: the requirement $k_1 \geq k_2 + \rho(\ell - 1) + \lambda$ of Lemma 4 should be guaranteed.

This can be achieved by setting $\tau = M^{-c} = 1/t$ and $\rho \cdot \ell = k_1 - (k_0 + 3)M^{1-c}$. From $k_2 = \overline{\delta}M$ and $k_0 = (\overline{\delta} - 3\tau)t$, we derive $\overline{\delta} = k_0/t + 3\tau = (k_0 + 3)/t$ and $k_2 = (k_0 + 3)M/t = (k_0 + 3)M^{1-c}$ (Notice that $t = M^c$), Therefore, the requirement of Lemma 4 is satisfied as desired:

$$\begin{aligned} k_2 + \rho(\ell - 1) + \lambda &= \Big((k_0 + 3)M^{1-c}\Big) + \Big(k_1 - (k_0 + 3)M^{1-c}\Big) - \rho + \lambda \\ &= k_1 - \rho + \lambda \leq k_1. \end{aligned} \tag{8}$$

The size of random seed $\rho = r_0 + r_1$ where $r_1 = \lambda$ and $r_0 = \log(M/t) + \log(1/\gamma) \cdot O(\theta^{-2.001})$ as given in [24](Lemma 8.4). □

Computational Complexity Recall that, in order to reduce computation cost, we could choose different sample size t_j for iteration j, where $t_1 < t_2 < \ldots < t_\ell = t = M^c$. The computational complexity of our proposed randomness extractor can be measured by the total number of bits read (or sampled) from the file (double counting repeated bits), i.e. the sum of t_j for $j \in [\ell]$. We will give an upper bound on the sum of t_j.

Lemma 7 (Complexity) *Suppose $M^{1-c} \geq 2$. The total number of bits (i.e. $\sum_{j=1}^{\ell} t_j$) of input file F accessed by the randomness extractor in Figure 2 is in $\mathcal{O}(M \log \ell)$.*
Note: (1) If the underlying extractor Ext *is Tabulation Hashing, then the constant behind the big-O notation is very small—around 2. (2) Multiple access to the same bit will be counted with its frequency. (3) The proof of this lemma is in Appendix C. .*

We remark that the extractor algorithm in Figure 2 can be modified into m concurrent threads/processes, while increasing the seed size by m times.

6 Applications in Constructing Secure Client-side Deduplication

PoW scheme can be applied (together with some other techniques) to construct secure client-side deduplication scheme in cloud storage.

6.1 Honest Cloud Server

If the cloud server is trusted, the PoW scheme alone implies a secure client-side deduplication, which is leakage resilient against outside attack: When receiving a file F for the *first* time from some owner of F, the server will also receive a short meta-data $\psi = \mathsf{PoW.S}(F, 1^\lambda)$ generated by the same owner of F. This small meta-data ψ together with hash value $\mathsf{h}(F)$ will be stored in the primary memory (i.e RAM) of the server, and the potentially large file F will be stored in the slower secondary memory (e.g. hard disk). After this setup, if any cloud client claims to own file F to the cloud server by presenting the hash value $\mathsf{h}(F)$, this cloud client will be required to take part in the interactive proof $\mathsf{PoW}.\langle \mathsf{P}, \mathsf{V} \rangle$—The client runs prover algorithm and the server runs the verifier algorithm V with the meta-data ψ as input, where ψ is associated to hash value $\mathsf{h}(F)$ and fetched efficiently from the primary memory. If this client convinced the server in the interactive proof, then the server believes that this client owns F and allow it to access the copy of F in the cloud storage.

6.2 Confidentiality against Semi-Honest Cloud Storage Server

The above construction of client side deduplication will expose users' files to the cloud storage server. To protect confidentiality of users' files, we could combine the privacy-preserving PoW scheme with convergent encryption [11]: User file F will be encrypted using convergent encryption method and the resulting ciphertext will be stored in the cloud storage. The rest is the same as in the above construction. We remark that in this setting, we have to employ privacy-preserving PoW scheme to prevent information leakage to the cloud storage server during the execution of PoW scheme.

[11] Convergent encryption [6,7,20,9,21] method derives encryption key from the plaintext.

This scheme is leakage-resilient against outside attack, and protect data confidentiality against semi-honest cloud server who does not have out-of-band bounded leakage access to the target file. We remark that "pollution attack" should be addressed using the way in existing works [20,9].

6.3 Bounded Leakage Resilient against Semi-Honest Cloud Storage Server

It is easy to see that it is *impossible* to achieve leakage-resilient client-side deduplication against cloud storage server without additional assumption: Since the cloud storage server has the ciphertext of user file F, and can learn the short encryption key from the bounded leakage access to F, it can decrypt the ciphertext to obtain F.

Our strategy is to introduce a new assumption: There exist N cloud storage servers, out of which at least one server is honest. Apply threshold secret sharing over file F and then store the result to the N server distributively. Then apply POR scheme with each server.

Note that in this setting, secret sharing scheme replaces encryption method (e.g. AES) to protect F, and the above approach can reach unconditional security!

7 Experiments

We implement a prototype of our randomness extractor, PoW scheme and client side deduplication (CSD) scheme [12] in Sec 3.3.1, in C language, where $\mathsf{MHT\text{-}POR}$ is adopted for POR scheme, tabulation hashing is adopted as the underlying randomness extractor with small size, and the threshold $T = 64$MB. Our test machine is a laptop computer, which is equipped with a 2.5GHz Intel Core 2 Duo mobile CPU (model T9300 in Year 2008), a 3GB PC2700-800MHZ RAM and a 7200RPM hard disk. The test machine runs 32 bits version of Gentoo Linux OS with kernel 3.1.10. The file system is EXT4 with 4KB page size.

Our test files are generated randomly[13] and are of size 64MB, 128MB, 256MB, 512MB and 1024MB respectively. Each experiment repeats 10 times and we report the average data [14] in Figure 3 (on page 22).

In summary, our randomness extractor, PoW scheme and client side deduplication scheme can consume data at the speed upto 7.7 MB/s, 6.8MB/s, and 4.2MB/s, respectively, for large test files. In contrast, the highest national-wide residential Internet *download* speed [41] is 14.2Mbps=1.775MB/s, and the *uploading* speed is even slower. As long as the client side deduplication consumes the data at the speed faster than the cloud user's uploading speed, then Internet transmission time could be partially saved in case of duplication. The benefits of saving in server storage and network bandwidth always persist, independent on the speed of the client side deduplication scheme.

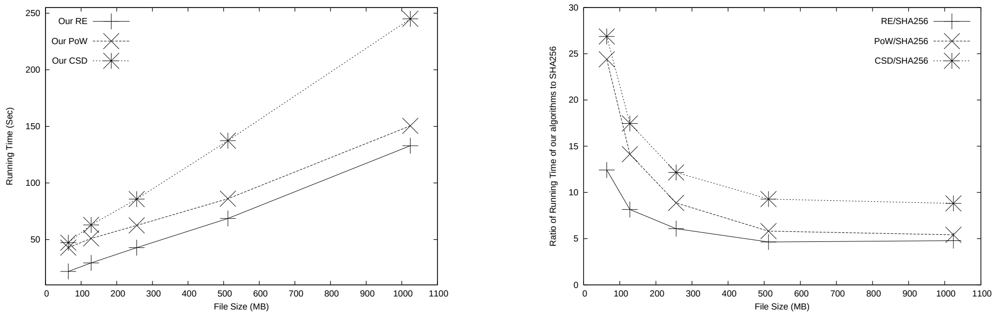
We remark that PoW scheme consists of three algorithms $(\mathsf{S}, \mathsf{P}, \mathsf{V})$, where Figure 3 just shows the experiment data for the most expensive algorithm S among them, the running time of P is very close to S, and the running time of V is within 2 seconds for all test files. Furthermore, the relatively expensive algorithms S and P are executed by some cloud users, and only the lightweight algorithm V is executed by the cloud server. Similar for client side deduplication scheme.

[12] Note that this CSD scheme in Section 6.2 consists of file encryption as part.

[13] Precisely, our test files are generated by encrypting some files with randomly chosen AES key using AES encryption method.

[14] Since the variance is very small, we save its details.

Fig. 3. Experiment Data. (a) Running time of our randomness extractor (PoW, Client side Deduplication, respectively) over test files of various sizes. The throughput of our extractor (PoW, Client side Deduplication, respectively) is upto 7.7 MB/s (6.8MB/s, 4.2MB/s, respectively) for large input files in our test machine. (b) Ratio of running time of our randomness extractor (PoW, Client side Deduplication, respectively) to SHA256 over test files of various sizes. Note that the output of our randomness extractor is 64MB long almost-perfect uniform random numbers extracted from input file. A straightforward implementation of such extractor using SHA256 should be about 2 millions times slower than SHA256.



(a) Running time

(b) Ratio of running time to SHA256

8 Conclusion and Open Problems

We were the first one to bridge construction of PoW with randomness extractor and proofs of retrievability. We also proposed a novel randomness extractor with large output size, which improves existing works in both seed length and entropy loss (i.e. the difference between entropy of input and output). Our proofs of ownership scheme can be applied in client-side deduplication of encrypted (unencrypted, too) data in cloud storage service, and the new randomness extractor may have independent interest.

Whether "partition-then-extract" approach works for *any* distribution of input file and how to apply pseudo-entropy extractor (e.g Yao-Entropy extractor) to construct proofs of ownership scheme, remain two open problems.

References

1. iHS iSuppli: Cloud Storage Services Now Have Over 375M Users, Could Reach 500M By Year-End `http://goo.gl/BO6zWy`.
2. Blog, A.: Amazon S3 goes exponential, now stores 2 trillion objects `http://goo.gl/NUIEny`, `http://gigaom.com/2013/04/18/amazon-s3-goes-exponential-now-stores-2-trillion-objects/`.
3. Blog, W.A.S.T.: Windows Azure Storage – 4 Trillion Objects and Counting `http://blogs.msdn.com/b/windowsazurestorage/archive/2012/07/20/windows-azure-storage-4-trillion-objects-and-counting.aspx`.
4. Blog, D.: Over 175 million people using Dropbox and more than a billion files synced each day `https://blog.dropbox.com/2013/07/dbx/`.
5. SNIA: Understanding Data De-duplication Ratios. white paper `http://www.snia.org/sites/default/files/Understanding_Data_Deduplication_Ratios-20080718.pdf`.
6. Douceur, J., Adya, A., Bolosky, W., Simon, D., , Theimer, M.: Reclaiming space from duplicate files in a serverless distributed file system. In: ICDCS '02: International Conference on Distributed Computing Systems. (2002)

7. Douceur, J., Bolosky, W., Theimer, M.: US Patent 7266689: Encryption systems and methods for identifying and coalescing identical objects encrypted with different keys (2007)
8. Storer, M., Greenan, K., Long, D., Miller, E.: Secure Data Deduplication. In: StorageSS '08: ACM international workshop on Storage security and survivability. (2008) 1–10
9. Bellare, M., Keelveedhi, S., Ristenpart, T.: Message-Locked Encryption and Secure Deduplication. In: EUROCRYPT '13: Advances in Cryptology. Volume 7881 of Lecture Notes in Computer Science. (2013) 296–312 `http://eprint.iacr.org/2012/631`.
10. Harnik D., Pinkas B., S.P.A.: Side Channels in Cloud Services: Deduplication in Cloud Storage. IEEE Security and Privacy Magazine, special issue of Cloud Security **8**(6) (2010)
11. Halevi, S., Harnik, D., Pinkas, B., Shulman-Peleg, A.: Proofs of ownership in remote storage systems. In: CCS '11: ACM conference on Computer and communications security. (2011) 491–500 `http://eprint.iacr.org/2011/207`.
12. Dropship: Dropbox api utilities (April 2011) `https://github.com/driverdan/dropship`.
13. Storer, M., Greenan, K., Long, D., Miller, E.: Secure data deduplication. In: Proceedings of the 4th ACM international workshop on Storage security and survivability. StorageSS '08 (2008) 1–10
14. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing **18**(1) (1989) 186–208
15. Wikipedia: PlayStation Network outage `http://en.wikipedia.org/wiki/PlayStation_Network_outage`.
16. wired.com: Dropbox Left User Accounts Unlocked for 4 Hours Sunday `http://www.wired.com/threatlevel/2011/06/dropbox/`; `http://blog.dropbox.com/?p=821`.
17. Twitter: Tweetdeck `http://money.cnn.com/2012/03/30/technology/tweetdeck-bug-twitter/`.
18. Juels, A., Kaliski, Jr., B.: Pors: proofs of retrievability for large files. In: CCS '07: ACM conference on Computer and communications security. (2007) 584–597
19. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: CCS '07: ACM conference on Computer and communications security. (2007) 598–609
20. Xu, J., Chang, E.C., Zhou, J.: Weak Leakage-Resilient Client side Deduplication of Encrypted Data in Cloud Storage. In: ASIACCS '13: Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security (Full Paper). (2013) 195–206 `http://eprint.iacr.org/2011/538`.
21. Bellare, M., Keelveedhi, S., Ristenpart, T.: DupLESS: Server-Aided Encryption for Deduplicated Storage (will appear in Usenix Security Symposium '13). Cryptology ePrint Archive, Report 2013/429 (2013) `http://eprint.iacr.org/2013/429`.
22. Ng, W.K., Wen, Y., Zhu, H.: Private data deduplication protocols in cloud storage. In: SAC '12: Proceedings of the 27th Annual ACM Symposium on Applied Computing. (2012) 441–446
23. Nisan, N., Zuckerman, D.: Randomness is linear in space. Journal of Computer and System Sciences **52**(Special issue on STOC 1993) (1996) 43–52
24. Vadhan, S.: Constructing Locally Computable Extractors and Cryptosystems in the Bounded-Storage Model. J. Cryptol. **17**(1) (2004) 43–77
25. Stinson, D.R.: Universal hash families and the leftover hash lemma, and applications to cryptography and computing. Journal of Combinatorial Mathematics and Combinatorial Computing **42** (2002) 3–31
26. Shacham, H., Waters, B.: Compact Proofs of Retrievability. In: ASIACRYPT '08. (2008) 90–107
27. Xu, J., Chang, E.C.: Towards efficient proof of retrievability. In: ASIACCS '12: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security (Full Paper). (2012) `http://eprint.iacr.org/2011/362`.
28. Goldreich, O.: A Sample of Samplers - A Computational Perspective on Sampling (survey). Electronic Colloquium on Computational Complexity (ECCC) **4**(20) (1997)
29. Dodis, Y., Vadhan, S., Wichs, D.: Proofs of Retrievability via Hardness Amplification. In: Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography. TCC '09 (2009) 109–127
30. Xu, J., Chang, E.C., Zhou, J.: Leakage-Resilient Client-side Deduplication of Encrypted Data in Cloud Storage. Cryptology ePrint Archive, Report 2011/538 (2011) `http://eprint.iacr.org/2011/538`.
31. Chang, E.C., Xu, J.: Remote Integrity Check with Dishonest Storage Server. In: ESORICS '08: European Symposium on Research in Computer Security: Computer Security. (2008) 223–237 `http://eprint.iacr.org/2008/346`.
32. Zheng, Q., Xu, S.: Secure and efficient proof of storage with deduplication. In: CODASPY '12: ACM conference on Data and Application Security and Privacy. (2012) 1–12
33. Barak, B., Dodis, Y., Krawczyk, H., Pereira, O., Pietrzak, K., Standaert, F.X., Yu, Y.: Leftover Hash Lemma, Revisited. In: CRYPTO. (2011) 1–20

34. Gabizon, A., Raz, R., Shaltiel, R.: Deterministic Extractors for Bit-Fixing Sources by Obtaining an Independent Seed. SIAM Journal on Computing **36**(4) (2006) 1072–1094
35. Pietro, R.D., Sorniotti, A.: Boosting Efficiency and Security in Proof of Ownership for Deduplication. In: ASIACCS '12: ACM Symposium on Information, Computer and Communications Security (Full Paper). (2012)
36. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Khan, O., Kissner, L., Peterson, Z., Song, D.: Remote data checking using provable data possession. ACM Transactions on Information and System Security **14** (2011) 12:1–12:34
37. Vadhan, S.: Pseudorandomness. Foundations and Trends in Theoretical Computer Science **7**(1-3) (2012) 1–336
38. Patrascu, M., Thorup, M.: The power of simple tabulation hashing. In: STOC '11: ACM symposium on Theory of computing. (2011) 1–10
39. Dodis, Y., Gennaro, R., Håstad, J., Krawczyk, H., Rabin, T.: Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes. In: CRYPTO '04. (2004) 494–510
40. Goldreich, O.: Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press (2004)
41. Akamai: The State of the Internet (2013 1st Quarter Report) `http://www.scribd.com/document_downloads/155497912?extension=pdf&from=embed&source=embed`.
42. Sahai, A., Vadhan, S.: A complete problem for statistical zero knowledge. J. ACM **50**(2) (2003) 196–249
43. Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.: Fuzzy Extractors: How to generate strong keys from biometrics and other noisy data. SIAM Journal on Computing **38**(1) (2008) 97–139

A Background knowledge of statistical difference

Lemma 8 (Fact 2.2 in Sahai and Vadhan [42]; Triangle Inequality) *For any probability distributions X, Y and Z,*

$$\mathsf{SD}(X,Y) \le \mathsf{SD}(X,Z) + \mathsf{SD}(Z,Y).$$

Lemma 9 (Fact 2.3 in Sahai and Vadhan [42]) *Suppose X_1 and X_2 are independent random variables on one probability space, and Y_1 and Y_2 are independent random variables on another probability space. Then*

$$\mathsf{SD}((X_1,X_2),\ (Y_1,Y_2)) \le \mathsf{SD}(X_1,Y_1) + \mathsf{SD}(X_2,Y_2).$$

Lemma 10 (Fact 2.4 in Sahai and Vadhan [42]) *If X and Y are random variables and A is any randomized (or deterministic) procedure, then*

$$\mathsf{SD}(A(X),\ A(Y)) \le \mathsf{SD}(X,\ Y).$$

Note: Statistical difference cannot be created out of nothing.

Lemma 11 (Fact 2.5 in Sahai and Vadhan [42]) *Suppose $X = (X_1, X_2)$ and $Y = (Y_1, Y_2)$ are probability distributions on a set $D \times E$ such that*

1. *X_1 and Y_1 are identically distributed, and*
2. *With probability greater than $(1-\epsilon)$ over $x \leftarrow X_1$ (equivalently, $x \leftarrow Y_1$), $\mathsf{SD}(X_2|_{X_1=x},\ Y_2|_{Y_1=x}) < \delta$, where $B|_{A=a}$ denotes the conditional distribution of B given that $A = a$ for jointly distributed random variables A and B.*

Then $\mathsf{SD}(X,Y) < \epsilon + \delta$.

Lemma 12 (Dodis *et al.* [43]) *Define* average min-entropy *of random variable A given random variable B as below*

$$\tilde{\mathbf{H}}_\infty(A \mid B) \overset{\text{def}}{=} -\log\left(\mathbb{E}_{b\leftarrow B}\left[\max_a \mathsf{Pr}[A = a \mid B = b]\right]\right). \tag{9}$$

Let A, B be random variables and B has at most 2^{ℓ_B} possible values. Then $\tilde{\mathbf{H}}_\infty(A \mid B) \ge \mathbf{H}_\infty((A,B)) - \ell_B \ge \mathbf{H}_\infty(A) - \ell_B$.

Corollary 13 *Let A and B be random variables where the domain of B is $\{0,1\}^{\ell_B}$. Then for any positive integer λ, for all but $2^{-\lambda}$ fraction of possible value $b \in \{0,1\}^{\ell_B}$, we have* $\mathbf{H}_\infty(A|_{B=b}) \geq \mathbf{H}_\infty(A) - \ell_B - \lambda$.

Proof. Applying Lemma 12, we have

$$\tilde{\mathbf{H}}_\infty(A \mid B) = -\log\Big(\mathbb{E}_{b\leftarrow B}\Big[\max_a \mathsf{Pr}[A=a \mid B=b]\Big]\Big) \geq \mathbf{H}_\infty(A) - \ell_B \tag{10}$$

$$\Rightarrow \mathbb{E}_{b\leftarrow B}\Big[\max_a \mathsf{Pr}[A=a \mid B=b]\Big] \leq 2^{-\mathbf{H}_\infty(A)+\ell_B} \tag{11}$$

$$\Rightarrow \mathsf{Pr}_{b\leftarrow B}\Big[\max_a \mathsf{Pr}[A=a \mid B=b] \geq v\Big] \tag{12}$$

$$\leq \frac{1}{v}\cdot \mathbb{E}_{b\leftarrow B}\Big[\max_a \mathsf{Pr}[A=a \mid B=b]\Big]$$

$$\leq v^{-1}\cdot 2^{-\mathbf{H}_\infty(A)+\ell_B}\text{(Markov's Inequality)} \tag{13}$$

Let $v = 2^{-\mathbf{H}_\infty(A)+\ell_B+\lambda}$. Thus

$$\mathsf{Pr}_{b\leftarrow B}\Big[\mathbf{H}_\infty(A|_{B=b}) \leq \mathbf{H}_\infty(A) - \ell_B - \lambda\Big] = \mathsf{Pr}_{b\leftarrow B}\Big[\max_a \mathsf{Pr}[A=a \mid B=b] \geq 2^{-\mathbf{H}_\infty(A)+\ell_B+\lambda}\Big] \leq 2^{-\lambda}.$$

□

B Proof for Randomness Extractor

Claim 1 $\mathsf{SD}(W_\ell,\ U_{|\rho\ell|}) \leq \ell(\epsilon_2 + 2^{-\lambda})$.

Proof (Proof of Claim 1). Recall that $U_{|n|}$ denotes the (independent) uniform random variable over $\{0,1\}^n$. Treat W_ℓ as a vector of ℓ elements $(\overline{\mathsf{Ext}}(F, U_{|\rho|,1}), \ldots, \overline{\mathsf{Ext}}(F, U_{|\rho|,\ell}))$. Let $W_\ell[1,i]$ denote the first i components of W_ℓ. We will prove the following statement using mathematical induction from $i = 1$ upto $i = \ell$:

$$\mathsf{SD}\big((U_{|\rho|,1}, W_\ell[1,i]),\ (U_{|\rho|,1}, U_{|i\rho|})\big) \leq i(\epsilon_2 + 2^{-\lambda}), i \in [\ell] \tag{14}$$

Notice that the case of $i = \ell$ is just Claim 1. The basic case of $i = 1$ is simply derived from the assumption that $\overline{\mathsf{Ext}}$ is a strong (k_2, ϵ_2)-extractor. Now we prove the induction step. Assuming Eq (14) holds for $i = j \in [\ell - 1]$, we try to prove that it also holds for $i = j + 1$.

Recall that $W_\ell = (w_1, \ldots, w_\ell)$ where $w_i = \overline{\mathsf{Ext}}(F, U_{|\rho|,i})$ and $U_{|\rho|,i}$'s are independent random variables over $\{0,1\}^\rho$. According to Corollary 13, conditional on all but $2^{-\lambda}$ fraction of possible values $(u_1, w_1, \ldots, w_j)$, the min-entropy F is at least $\mathbf{H}_\infty(F) - j\rho - \lambda \geq k_1 - j\rho - \lambda \geq k_2$ (Note that $U_{|\rho|,1}$ is independent on F). Applying the property of extractor $\overline{\mathsf{Ext}}$, $w_{j+1} = \overline{\mathsf{Ext}}(F, U_{|\rho|,j+1})$ is ϵ_2-close to uniform randomness $U_{|\rho|}$ (conditional on $(w_1, \ldots, w_j)$). Applying Lemma 11, we have

$$\mathsf{SD}\big((U_{|\rho|,1}, w_1, \ldots, w_j, w_{j+1}),\ \ (U_{|\rho|,1}, w_1, \ldots, w_j, U_{|\rho|})\big) \leq \epsilon_2 + 2^{-\lambda}. \tag{15}$$

By induction hypothesis, i.e. Eq (14) holds for $i = j$,

$$\mathsf{SD}\big((U_{|\rho|,1}, w_1, \ldots, w_j),\ \ (U_{|\rho|,1}, U_{|j\rho|})\big) \leq j(\epsilon_2 + 2^{-\lambda}).$$

$$\Rightarrow \mathsf{SD}\big((U_{|\rho|,1}, w_1, \ldots, w_j, U_{|\rho|}),\ \ (U_{|\rho|,1}, U_{|j\rho|}, U_{|\rho|})\big)$$

$$\leq j(\epsilon_2 + 2^{-\lambda}) \tag{16}$$

The last derivation is because that $U_{|\rho|}$ is independent uniform randomness. Combining Eq (15) and Eq (16) using triangle inequality (Lemma 8), we have

$$\mathsf{SD}\big((U_{|\rho|,1}, w_1, \ldots, w_j, w_{j+1}),\ \ (U_{|\rho|,1}, U_{|j\rho|}, U_{|\rho|})\big) \leq (j+1)(\epsilon_2 + 2^{-\lambda}). \tag{17}$$

The induction step finishes and thus the original Claim 1 is proved. □

Claim 2 *For any $i \in [\ell - 1]$,* $\mathsf{SD}(W_i,\ W_{i+1}) \leq 4(\epsilon_2 + 2^{-\lambda})$. $\mathsf{SD}(W_1, W_\ell) \leq 4\ell(\epsilon_2 + 2^{-\lambda})$.

Proof (Proof of Claim 2). Let $\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4$ be independent and uniform random variables over $\{0,1\}^\rho$.

According to Corollary 13, conditional on $(U_{|\rho|,1}, w_1, \ldots, w_{i-1})$: F has at least[15] $k_1 - \rho(i-1) - \lambda \geq k_2$ bits min-entropy with o.h.p $(1 - 2^{-\lambda})$. So $\mathsf{SD}(w_i = \overline{\mathsf{Ext}}(F, U_{|\rho|,i}),\ \mathcal{U}_1) < \epsilon_2 + 2^{-\lambda}$. Applying Lemma 10,

$$\mathsf{SD}((w_i, \overline{\mathsf{Ext}}(F, w_i)),\ (\mathcal{U}_1, \overline{\mathsf{Ext}}(F, \mathcal{U}_1))) \leq \mathsf{SD}(w_i, \mathcal{U}_1) \leq \epsilon_2 + 2^{-\lambda}. \tag{18}$$

Since $\overline{\mathsf{Ext}}$ is an extractor, $\mathsf{SD}((\mathcal{U}_1, \overline{\mathsf{Ext}}(F, \mathcal{U}_1)),\ (\mathcal{U}_1, \mathcal{U}_2)) \leq \epsilon_2 + 2^{-\lambda}$

Conditional on $(U_{|\rho|,1}, w_1, \ldots, w_{i-1}, w_i)$, $\mathsf{SD}(\overline{\mathsf{Ext}}(F, \mathcal{U}_3,\ \mathcal{U}_4) \leq \epsilon_2$, since $\overline{\mathsf{Ext}}$ is an extractor.

Conditional on $(U_{|\rho|,1}, w_1, \ldots, w_{i-1})$: $(w_i, \overline{\mathsf{Ext}}(F, \mathcal{U}_3)) \approx_{\epsilon_2} (w_i, \mathcal{U}_4) \approx_{\epsilon_2} (\mathcal{U}_1, \mathcal{U}_4) \approx_0 (\mathcal{U}_1, \mathcal{U}_2)$.

Therefore, by apply triangle inequality multiple times, we have[16] $(w_i, \overline{\mathsf{Ext}}(F, \mathcal{U}_3)) \approx_{4(\epsilon_2 + 2^{-\lambda})} (w_i, \overline{\mathsf{Ext}}(F, w_i))$. Applying Lemma 10 over the last equation, we have

$$(w_i, \overline{\mathsf{Ext}}(F, \mathcal{U}_3), \mathsf{Nest}(F, \overline{\mathsf{Ext}}(F, \mathcal{U}_3), \ell - i - 1)) \approx_{4(\epsilon_2 + 2^{-\lambda})} (w_i, , \overline{\mathsf{Ext}}(F, w_i), \mathsf{Nest}(F, \overline{\mathsf{Ext}}(F, w_i), \ell - i - 1)) \tag{19}$$

Therefore,

$$\begin{aligned} &\Big(U_{|\rho|,1}, w_1, \ldots, w_i, \overline{\mathsf{Ext}}(F, \mathcal{U}_3), \mathsf{Nest}(F, \overline{\mathsf{Ext}}(F, \mathcal{U}_3), \ell - i - 1)\Big) \\ \approx_{4(\epsilon_2 + 2^{-\lambda})} &\Big(U_{|\rho|,1}, w_1, \ldots, w_i, , \overline{\mathsf{Ext}}(F, w_i), \mathsf{Nest}(F, \overline{\mathsf{Ext}}(F, w_i), \ell - i - 1)\Big) \end{aligned} \tag{20}$$

By the definition of Nest, we have $\big(\overline{\mathsf{Ext}}(F, w_i), \mathsf{Nest}(F, \overline{\mathsf{Ext}}(F, w_i), \ell - i - 1)\big) = \mathsf{Nest}(F, w_i, \ell - i)$. By the definition of random variable W_i, the left hand side of the above Eq (20) is identically distributed as W_{i+1} and the right hand side of Eq (20) is identically distributed as W_i. Thus $W_{i+1} \approx_{4(\epsilon_2 + 2^{-\lambda})} W_i$. Therefore, applying the triangle inequality for $i = 1, 2, \ldots, \ell$, we have $W_1 \approx_{4\ell(\epsilon_2 + 2^{-\lambda})} W_\ell$. □

C Proof of Complexity

Lemma 7 *Suppose $M^{1-c} \geq 2$. $\sum_{j=1}^{\ell} t_j = \mathcal{O}(M \log \ell)$.*

Proof. In each iteration j, the sampled t_j bits should have sufficient min-entropy to feed in the underlying extractor Ext, that is, $t_j(k_1 - j\rho - \lambda)/M - 3\tau t \geq k_0$. Thus the minimal possible value for t_j is

$$t_j^* = \frac{k_0}{\frac{k_1 - j\rho - \lambda}{M} - 3\tau} = \frac{k_0}{a - j\rho M^{-1}}$$

Here a is defined as: $a = \frac{k_1 - \lambda}{M} - 3\tau = \frac{1}{M}\left(k_1 - \lambda - 3M^{1-c}\right)$. Since $\rho \cdot \ell = k_1 - (k_0 + 3)M^{1-c}$, we have

$$a = \frac{1}{M}\left(\rho\ell + k_0 M^{1-c} - \lambda\right) \geq \frac{1}{M}\left(\rho\ell + \rho M^{1-c} - \lambda\right) \geq \frac{1}{M}\left(\rho\ell + \rho\right) = \frac{\rho(\ell+1)}{M} \quad \text{(requires } \ M^{1-c} \geq 2) \tag{21}$$

Therefore,

$$\begin{aligned} \sum_{j=1}^{\ell} t_j^* &\leq k_0 \rho^{-1} M \int_{a - \rho\ell M^{-1}}^{a - \rho M^{-1}} \frac{1}{x} = k_0 \rho^{-1} M \ln x \Bigg|_{a - \rho\ell M^{-1}}^{a - \rho M^{-1}} && (22) \\ &= k_0 \rho^{-1} M \left(\ln(a - \rho M^{-1}) - \ln(a - \rho\ell M^{-1})\right) && (23) \\ &= k_0 \rho^{-1} M \ln \frac{a - \rho M^{-1}}{a - \rho\ell M^{-1}} \ \left(\substack{\text{This function is decreasing} \\ \text{w.r.t. variable } a}\right) && (24) \\ &\leq k_0 \rho^{-1} M \ln \frac{\rho(\ell+1)M^{-1} - \rho M^{-1}}{\rho(\ell+1)M^{-1} - \rho\ell M^{-1}} && (25) \\ &= k_0 \rho^{-1} M \ln \ell = \mathcal{O}(M \ln \ell). && (26) \end{aligned}$$

In case of Tabulation Hashing [38] or HMAC or CBC-MAC [39] as the underlying extractor Ext, $k_0 \rho^{-1} \approx 2$ is a small constant.

□

[15] Note that $U_{|\rho|,1}$ is independent on F

[16] $X \approx_{4\epsilon_2 + 2 \cdot 2^{-\lambda}} Y \quad \Rightarrow \quad X \approx_{4(\epsilon_2 + 2^{-\lambda})} Y$.