DeCredit Decentralized Storage

A Privacy-Preserving Provable Data Procession Scheme

1. Summarize

Our contribution can be summarized as the following aspects:

1) We motivate the public auditing system of data storage security in decentralized network and provide a privacy-preserving auditing protocol, i.e., our scheme enables auditors to audit user's data without learning the data content.

2) Our scheme supports scalable and efficient public auditing. Specifically, our scheme achieves batch auditing where multiple auditing tasks can be performed simultaneously by the auditor.

2. Problem Statement

2.1 The System and Threat Model

We consider a decentralized data storage network involving three different entities, as illustrated in Fig. 1: the User (U), who has data files to be stored in the network; the Storage Node (SN), which is managed by the DeCredit consensus to provide data storage service and has significant storage space and computation resources; the Auditor (A), who is chosen to access the storage service and verify its correctness.

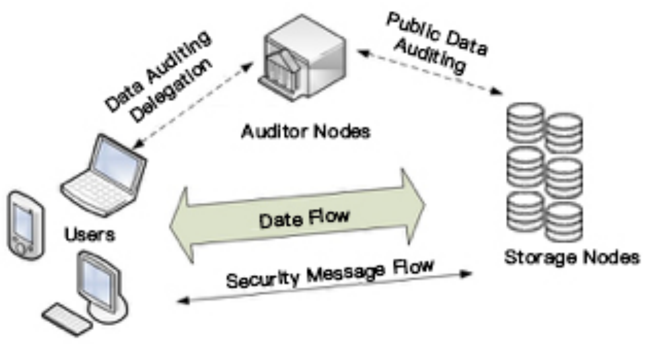


Fig 1. The architecture of decentralized data storage service

Users rely on the SN for data storage and maintenance. To save the computation resource as well as the network burden, users may resort to Auditors for ensuring the storage integrity of their outsourced data, while still hoping to keep their data private from the Auditors.

We assume storage nodes are rational. Namely, in most of time they behave properly according to the prescribed protocol execution and earn the incentives. However, for their own benefits the SN might neglect to keep or deliberately delete rarely accessed data files which belong to ordinary users. Moreover, the SN may decide to hide the data corruptions caused by hacks or Byzantine failures to maintain reputation. We assume the Auditors, who is chosen by randomness, is reliable and independent, and thus has no incentive to collude with either the SN or the Users during the auditing process. However, it harms the user if the Auditor could learn the outsourced data after the audit.

Basically, the user can sign a certificate granting audit rights to the Auditor public key, and all audits from the Auditor are authenticated against such a certificate.

2.2 Design Goals

Our protocol design should achieve the following security and performance guarantees.
1）Public auditability: to allow Auditor to verify the correctness of the data on demand without retrieving a copy of the whole data or introducing additional online burden.
2）Storage correctness: to ensure that there exists no cheating storage node that can pass the audit without indeed storing users' data intact.
3）Privacy-preserving: to ensure that the Auditor cannot derive users' data content from the information collected during the auditing process.
4）Batch auditing: to enable Auditor with secure and efficient auditing capability to cope with multiple auditing delegations from possibly large number of different users simultaneously.
5) Lightweight: to allow Auditor to perform auditing with minimum communication and computation overhead.

3. Proposed Schemes

This section presents our scheme which provides a decentralized storage solution. We start from discussing two straightforward schemes and their demerits. Then we present our scheme and show how to extent our scheme to support batch auditing.

3.1 Definitions and Framework

A public auditing scheme consists of four algorithms (KeyGen, SigGen, GenProof, VerifyProof). KeyGen is a key generation algorithm that is run by the User to setup the scheme. SigGen is used by the User to generate verification metadata, which may consist of MAC, signatures, or other related information that will be used for auditing. GenProof is run by the Storage Node to generate a proof of data storage correctness, while VerifyProof is run by the Auditor to audit the proof from SN.

Running a public auditing system consists of two phases, Setup and Audit:

- Setup: The User initializes the public and secret parameters by executing KeyGen, and pre-processes the data file F by using SigGen to generate the verification metadata. The User then stores the data file F and the verification metadata to the Storage Node.

- Audit: The Auditor issues an audit message or challenge to the Storage Node to make sure that the node has retained the data file F properly at the time of the audit. The Storage Node will derive a response message from a function of the stored data file F and its verification metadata by executing GenProof. The Auditor then verifies the response via VerifyProof.

Our decentralized storage design does not assume any additional property on the data file. If the User wants to have more error-resiliency, he/she can always first redundantly encodes the data file and then uses our system with the data file that has error-correcting codes integrated (e.g. Erasure Coding).

3.2 Notation and Preliminaries

- F – the data file to be outsourced, denoted as a sequence of n blocks $m_1, \ldots, m_n \in Z_p$.
- $MAC_{(\cdot)}(\cdot)$ – message authentication code (MAC) function, defined as: $K \times \{0, 1\}^* \rightarrow \{0, 1\}^l$ where K denotes the key space.
- $H(\cdot), h(\cdot)$ – cryptographic hash functions.

We now introduce some necessary cryptographic background for our proposed scheme.

Bilinear Map. Let G_1, G_2 and G_T be multiplicative cyclic groups of prime order p. Let g_1 and g_2 be generators of G_1 and G_2, respectively. A bilinear map is a map $e : G_1 \times G_2 \rightarrow G_T$ such that for all $u \in G_1$, $v \in G_2$ and $a, b \in Z_p$, $e(u^a, v^b) = e(u,v)^{ab}$. This bilinearity implies that for any $u_1, u_2 \in G_1, v \in G_2, e(u_1 \cdot u_2, v) = e(u_1, v) \cdot e(u_2, v)$. Of course, there exists an efficiently computable algorithm for computing e and the map should be non-trivial, i.e., e is non-degenerate: $e(g_1, g_2) != 1$.

3.3 Basic Schemes

MAC-based solution suffers from undesirable systematic demerits - bounded usage and stateful verification, which may pose additional online burden to users, in a public auditing setting. This somehow also shows that the auditing problem is still not easy to solve even we have introduced the Audior.

Homomorphic linear authenticators (HLA) based solution covers many recent proof of storage systems. However, all existing HLA-based systems are not real privacy-preserving because the linear combination of blocks may potentially reveal user data information to the Auditor, and violates the privacy preserving guarantee. Specifically, if an enough number of the linear combinations of the same data blocks are collected, the Auditor can simply derive the user's data content by solving a system of linear equations.

The analysis of these basic schemes leads to our main scheme, which overcomes all these drawbacks. Our main scheme to be presented is based on a specific HLA scheme.

3.4 Privacy-Preserving Public Auditing Scheme

Overview. To achieve privacy-preserving public auditing, we propose to uniquely integrate the homomorphic linear authenticator with random masking technique. In our protocol, the linear combination of sampled blocks in the node's response is masked with randomness. With random masking, the Auditor no longer has all the necessary information to build up a correct group of linear equations and therefore cannot derive the user's data content, no matter how many linear combinations of the same set of file blocks can be collected. On the other hand, the correctness validation of the block authenticator pairs can still be carried out in a new way which will be shown below, even with the presence of the randomness. Our design makes use of a public key based HLA, to equip the auditing protocol with public auditability. Specifically, we use the HLA proposed in "Compact proofs of retrievability"[1], which is based on "Short signatures from Weil pairing[2].

Scheme Details. Let G_1, G_2 and G_T be multiplicative cyclic groups of prime order p, and $e : G_1 \times G_2 \rightarrow G_T$ be a bilinear map as introduced in preliminaries. Let g be a generator of G_2. $H(\cdot)$ is a secure map-to-point hash function: $\{0, 1\}^* \rightarrow G_1$, which maps strings uniformly to G_1. Another hash function $h(\cdot) : G_T \rightarrow Z_p$ maps group element of G_T uniformly to Z_p. The proposed scheme is as follows:

Setup Phase: The cloud user runs KeyGen to generate the public and secret parameters. Specifically, the user chooses a random signing key pair (spk, ssk), a random $x \leftarrow Z_p$, a random element $u \leftarrow G_1$, and computes $v \leftarrow g^x$. The secret parameter is $sk = (x, ssk)$ and the public parameters are $pk = (spk, v, g, u, e(u, v))$.

Given a data file $F = (m_1, . . . , m_n)$, the user runs SigGen to compute authenticator σ_i for each block m_i: $\sigma_i \leftarrow (H(W_i) \cdot u^{m_i})x \in G_1$. Here $W_i = name||i$ and $name$ is chosen by the user uniformly at random from Z_p as the identifier of file F. Denote the set of authenticators by $\Phi = \{\sigma_i\}_{1 \leq i \leq n}$.

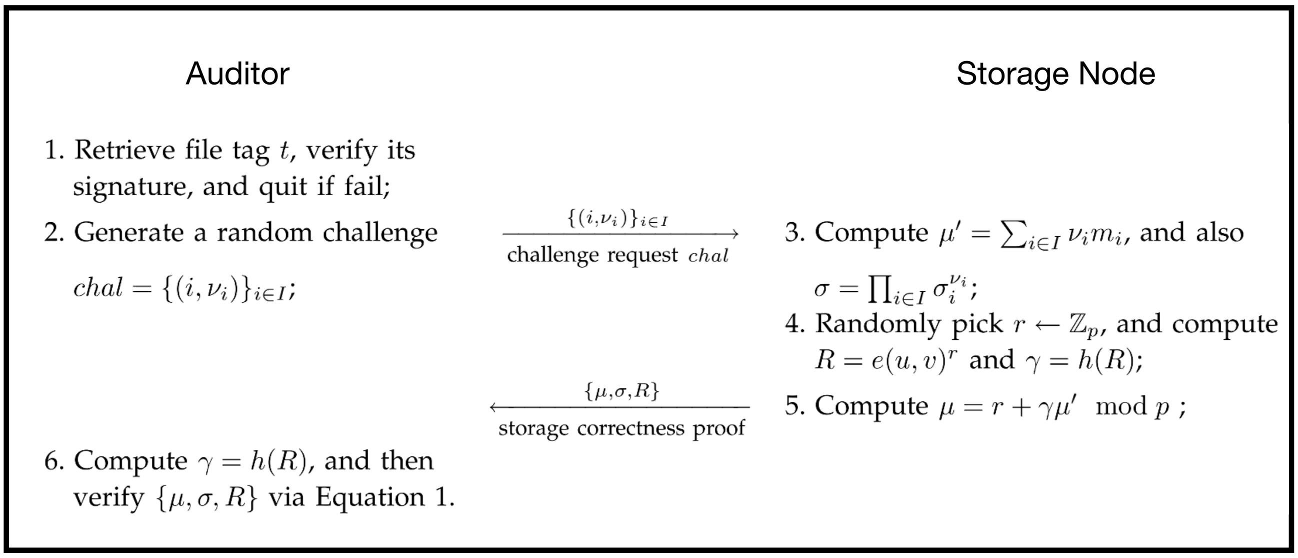
The last part of SigGen is for ensuring the integrity of the unique file identifier $name$. One simple way to do this is to compute $t = name||SSig_{ssk}(name)$ as the file tag for F, where $SSig_{ssk}(name)$ is the signature on $name$ under the private key ssk. For simplicity, we assume the the Auditor knows the number of blocks n. The User then sends F along with the verification metadata (Φ, t) to the network.

Audit Phase: The Auditor first retrieves the file tag t. With respect to the mechanism we describe in the Setup phase, the Auditor verifies the signature $SSig_{ssk}(name)$ via spk, and quits by emitting FALSE if the verification fails. Otherwise, the Auditor recovers $name$.

Now it comes to the "core" part of the auditing process. To generate the challenge message for the audit "$chal$", the Auditor picks a random c-element subset $I = \{s_1, . . . , s_c\}$ of set [1, n]. For each element $i \in I$, the Auditor also chooses a random value v_i (of bit length that can be shorter than $|p|$, as explained in Compact proofs of retrievability[1]). The message “*chal*” specifies the positions of the blocks that are required to be checked. The Auditor sends $chal = \{(i, v_i)\}_{i \in I}$ to the Storage Node.

Upon receiving challenge $chal = \{(i, v_i)\}_{i \in I}$, the Storage Node runs GenProof to generate a response proof of data storage correctness. Specifically, the node chooses a random element $r \leftarrow Z_p$, and calculates $R = e(u, v)^r \in G_T$. Let μ' denote the linear combination of sampled blocks specified in $chal$: $\mu' = \Sigma_{i \in I} v_i m_i$. To blind μ' with r, the Storage Node computes: $\mu = r + \gamma\mu' \bmod p$, where $\gamma = h(R) \in Zp$. Meanwhile, the node also calculates an aggregated authenticator $\sigma = \Pi_{i \in I} \sigma_i^{v_i} \in G_1$. It then sends $\{\mu, \sigma, R\}$ as the response proof of storage correctness to the Auditor. With the response from the Storage Node, the Auditor runs VerifyProof to validate the response by first computing $\gamma = h(R)$ and then checking the verification equation. The protocol is illustrated in Fig. 2.

$$R \cdot e(\sigma^\gamma, g) \overset{?}{=} e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^\gamma \cdot u^\mu, v) \qquad (1)$$



Auditor

Storage Node

1. Retrieve file tag t, verify its signature, and quit if fail;
2. Generate a random challenge $chal = \{(i, \nu_i)\}_{i \in I}$;

$\xrightarrow[\text{challenge request } chal]{\{(i,\nu_i)\}_{i \in I}}$

3. Compute $\mu' = \sum_{i \in I} \nu_i m_i$, and also $\sigma = \prod_{i \in I} \sigma_i^{\nu_i}$;
4. Randomly pick $r \leftarrow \mathbb{Z}_p$, and compute $R = e(u, v)^r$ and $\gamma = h(R)$;
5. Compute $\mu = r + \gamma\mu' \mod p$;

$\xleftarrow[\text{storage correctness proof}]{\{\mu,\sigma,R\}}$

6. Compute $\gamma = h(R)$, and then verify $\{\mu, \sigma, R\}$ via Equation 1.

Fig. 2: The privacy-preserving public auditing protocol

The correctness of the above verification equation can be elaborated as follows:

$$\begin{aligned}
R \cdot e(\sigma^\gamma, g) &= e(u, v)^r \cdot e((\prod_{i=s_1}^{s_c} (H(W_i) \cdot u^{m_i})^{x \cdot \nu_i})^\gamma, g) \\
&= e(u^r, v) \cdot e((\prod_{i=s_1}^{s_c} (H(W_i)^{\nu_i} \cdot u^{\nu_i m_i})^\gamma, g)^x \\
&= e(u^r, v) \cdot e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^\gamma \cdot u^{\mu'\gamma}, v) \\
&= e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^\gamma \cdot u^{\mu'\gamma + r}, v) \\
&= e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^\gamma \cdot u^\mu, v)
\end{aligned}$$

Properties of Our Protocol. It is easy to see that our protocol achieves public auditability. There is no secret keying material or states for the Auditor to keep or maintain, and the auditing protocol does not pose any potential online burden. This approach ensures the privacy of user data content during the auditing process by employing a random masking r to hide μ, a linear combination of the data blocks. Note that the value R in our protocol, which enables the privacy-preserving guarantee, will not affect the validity of the equation, due to the circular relationship between R and γ in $\gamma = h(R)$ and the verification equation. Storage correctness thus follows from that of the underlying protocol - "Compact proofs of retrievability[1]". Besides, the HLA helps achieve the constant communication overhead for node's response during the audit: the size of $\{\sigma, \mu, R\}$ is independent of the number of sampled blocks c.

Previous work showed that if the Storage Node is missing a fraction of the data, then the number of blocks that needs to be checked in order to detect node misbehavior with high probability is in the order of $O(1)$. For examples, if the node is missing 1% of the data F, to detect this misbehavior with probability larger than 95%, the Auditor only needs to audit for $c = 300$ *(up to* $c = 460$ *for 99%)* randomly chosen blocks of F. Given the huge volume of data stored in the network, checking a portion of the data file is more affordable and practical for both the Auditor and the Storage Node than checking all the data, as long as the sampling strategies provides high probability assurance.

3.5 Support for Batch Auditing

With the establishment of privacy-preserving public auditing, the Auditor may concurrently handle multiple auditing upon the assignments. The individual auditing of these tasks for the Auditor can be tedious and very inefficient. Given K auditing assignments on K distinct data files from K different users, it is more advantageous for the Auditor to batch these multiple tasks together and audit at one time. Keeping this natural demand in mind, we slightly modify the protocol in a single user case, and achieves the aggregation of K verification equations (for K auditing tasks) into a single one, as shown in Equation 2. As a result, a secure batch auditing protocol for simultaneous auditing of multiple tasks is obtained. The details are described as follows.

Setup Phase: Basically, the Users just perform Setup independently. Suppose there are K users in the system, and each user k has a data file $F_k = (m_{k,1}, \dots, m_{k,n})$ to be stored to the Storage Node in the network, where $k \in \{1, \dots, K\}$. For simplicity, we assume each file Fk has the same number of n blocks. For a particular user k, denote his/her secret key as (x_k, ssk_k), and the corresponding public parameter as $(spk_k, v_k, g, u_k, e(u_k, v_k))$ where $v_k = g^{x_k}$. Similar to the single user case, each user k has already randomly chosen a different (with overwhelming probability) name $name_k \in Z_p$ for his/her file F_k, and has correctly generated the corresponding file tag $t_k = name_k || SSig_{ssk_k}(name_k)$. Then, each user k runs SigGen and computes $\sigma_{k,i}$ for block $m_{k,i}$: $\sigma_{k,i} \leftarrow (H(name_k||i) \cdot u_k^{m_{k,i}})^{x_k} = (H(W_{k,i}) \cdot u_k^{m_{k,i}})^{x_k} \in G_1$ $(i \in \{1, \dots, n\})$, where $W_{k,i} = name_k||i$. Finally, each user k sends file F_k, set of authenticators Φ_k, and tag t_k to the Storage Node.

Audit Phase: The Auditor first retrieves and verifies file tag t_k for each user k for later auditing. If the verification fails, the Auditor quits by emitting FALSE; otherwise, the

Auditor recovers $name_k$. Then the Auditor sends the audit challenge $chal = \{(i, v_i)\}i \in I$ to the Storage Node for auditing data files of all K users.

Upon receiving $chal$, for each user $k \in \{1, . . . , K\}$, the Storage Node randomly picks $r_k \in Z_p$ and computes $R_k = e(u_k, v_k) r_k$. $Denote\ R = R_1 \cdot R_2 \cdot \cdot \cdot R_K,\ and\ \xi = vk_1||vk_2|| \cdot \cdot \cdot ||vk_K$, our protocol further requires the node to compute $\gamma k = h(R||v_k||\xi)$. Then, the randomly masked responses can be generated as follows:

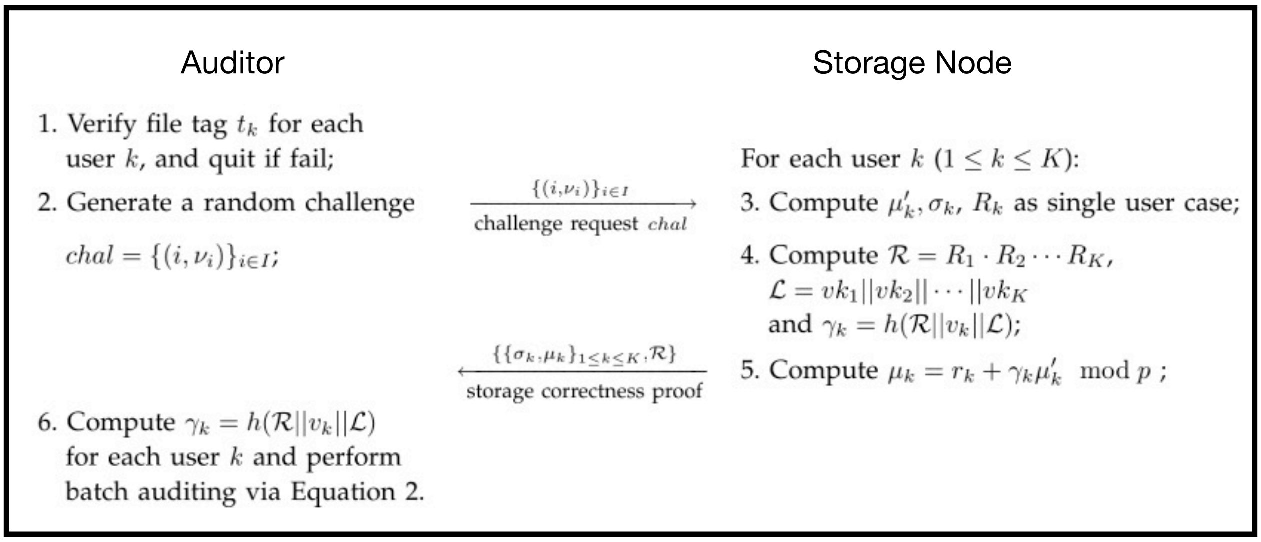
$$\mu_k = \gamma_k \sum_{i=s_1}^{s_c} \nu_i m_{k,i} + r_k \mod p \quad \text{and} \quad \sigma_k = \prod_{i=s_1}^{s_c} \sigma_{k,i}^{\nu_i}.$$

The Storage Node then responses the Auditor with $\{\{\sigma_k, \mu_k\}_{1 \le k \le K}, R\}$.

To verify the response, the Auditor can first compute $\gamma k = h(R||vk||\xi)$ for $1 \le k \le K$. Next, the Auditor checks if the following equation holds:

$$\mathcal{R} \cdot e(\prod_{k=1}^{K} \sigma_k{}^{\gamma_k}, g) \overset{?}{=} \prod_{k=1}^{K} e((\prod_{i=s_1}^{s_c} H(W_{k,i})^{\nu_i})^{\gamma_k} \cdot u_k^{\mu_k}, v_k) \qquad (2)$$

The batch protocol is illustrated in Fig. 3.


Auditor

1. Verify file tag t_k for each user k, and quit if fail;
2. Generate a random challenge $chal = \{(i, \nu_i)\}_{i \in I}$;

$\xrightarrow{\{(i,\nu_i)\}_{i\in I}}$ challenge request $chal$

Storage Node

For each user k $(1 \le k \le K)$:

3. Compute μ'_k, σ_k, R_k as single user case;
4. Compute $\mathcal{R} = R_1 \cdot R_2 \cdots R_K$, $\mathcal{L} = vk_1||vk_2|| \cdots ||vk_K$ and $\gamma_k = h(\mathcal{R}||v_k||\mathcal{L})$;
5. Compute $\mu_k = r_k + \gamma_k \mu'_k \mod p$;

$\xleftarrow{\{\{\sigma_k,\mu_k\}_{1\le k\le K},\mathcal{R}\}}$ storage correctness proof

6. Compute $\gamma_k = h(\mathcal{R}||v_k||\mathcal{L})$ for each user k and perform batch auditing via Equation 2.

Fig. 3: The batch auditing protocol

Here the left-hand side (LHS) of Equation 2 expands as:

$$\begin{aligned} \text{LHS} &= R_1 \cdot R_2 \cdots R_K \cdot \prod_{k=1}^{K} e(\sigma_k{}^{\gamma_k}, g) \\ &= \prod_{k=1}^{K} R_k \cdot e(\sigma_k{}^{\gamma_k}, g) \\ &= \prod_{k=1}^{K} e((\prod_{i=s_1}^{s_c} H(W_{k,i})^{\nu_i})^{\gamma_k} \cdot u_k^{\mu_k}, v_k) \end{aligned}$$

which is the right hand side, as required. Note that the last equality follows from Equation 1.
Efficiency Improvement. As shown in Equation 2, batch auditing not only allows the Auditor to perform the multiple auditing tasks simultaneously, but also greatly reduces the computation cost on the Auditor side. This is because aggregating K verification equations into one helps reduce the number of relatively expensive pairing operations from $2K$, as required in the individual auditing, to $K + 1$. Thus, a considerable amount of auditing time is expected to be eliminated.

Identification of Invalid Responses. The verification equation (Equation 2) only holds when all the responses are valid, and fails with high probability when there is even one single invalid response in the batch auditing. In some situations, a response collection may contain invalid responses, especially $\{\mu_k\}_{1 \le k \le K}$, caused by accidental data corruption, or possibly malicious activity by a Storage Node. The ratio of invalid responses to the valid could be quite small, but yet a standard batch auditor will reject the entire collection. To further sort out these invalid responses in the batch auditing, we can utilize a recursive divide-and-conquer approach (binary search), as suggested by "Practical short signature batch verification"[3]. Specifically, if the batch auditing fails, we can simply divide the collection of responses into two halves, and recurse the auditing on halves via Equation 2. The Auditor may now require the Storage node to send back all the $\{R_k\}_{1 \le k \le K}$, as in individual auditing. Using this recursive binary search approach, even if up to 18% of responses are invalid, batch auditing still performs faster than individual verification.

To get a complete view of batching efficiency, a timed batch auditing test was conducted, where the number of auditing tasks is increased from 1 to approximately 200 with intervals of 8. The performance of the corresponding non-batched (individual) auditing is provided as a baseline for the measurement. Following the same experimental settings c = 300 and c = 460, the average per task auditing time, which is computed
by dividing total auditing time by the number of tasks, is given in Fig. 4 for both batch and individual auditing. It can be shown that compared to individual auditing, batch auditing indeed helps reducing the Auditor's computation cost, as more than 11% and 14% of per-task auditing time is eliminated, when c is set to be 460 and 300, respectively.

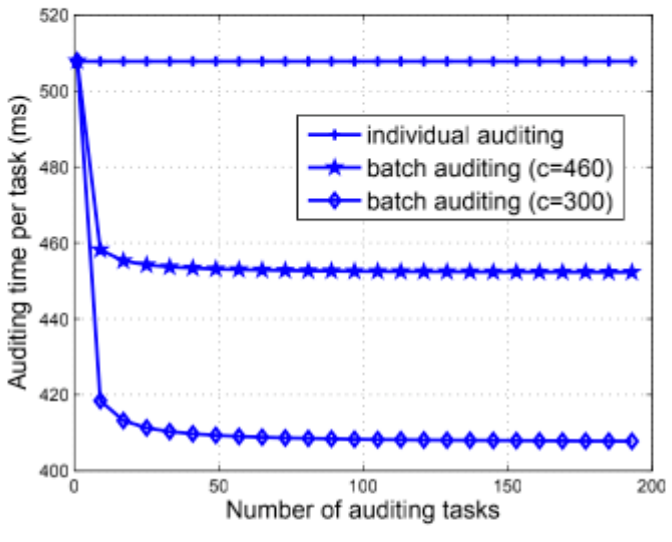


Fig. 4: Comparison on auditing time between batch and individual auditing.

4. Future Work

4.1 Zero-Knowledge Proof

Though our scheme prevents the Auditor from directly deriving $\mu'$from μ, it does not rule out the possibility of offline guessing attack from the Auditor using valid σ from the response. Specifically, the Auditor can always guess whether the stored data contains certain message $\tilde{m}$, by checking $e(\sigma, g) ?= e((\prod^{sc}_{i=s1} H(W_i)^{vi}) \cdot u^{\tilde{\mu}'}, v)$.Thus, our main scheme is not semantically secure yet. However, we must note that $\tilde{\mu}'$ is chosen from Z_p and $|p|$ is usually larger than 160 bits in practical security settings. Therefore, the Auditor has to test basically all the values of Z_p in order to make a successful guess. Given no background information, the success probability of such all-or-nothing guess launched by the Auditor can be negligible. Nonetheless, for completeness, we want to give a provably zero-knowledge based public auditing scheme, which further eliminates the possibilities of above offline guessing attack.

4.2 Support for Data Dynamics

Outsourced data might not only be accessed but also updated frequently by Users for various application purposes. Hence, supporting data dynamics for privacy-preserving public auditing is also of paramount importance.

Ideally, it can be achieved by replacing the index information i with m_i in the computation of block authenticators and using Merkle hash tree (MHT) for the underlying block sequence enforcement.

4.3 Deterministic Deletion

Deterministic Deletion becomes a difficult problem because the User no longer has data ownership once he/she outsourcing the data to the network. A solution based on trusted computing[4] could be used to to achieve the data deterministic destruction.

5. Blockchain and Substrate Implementation

5.1 Off-chain Worker

The off-chain worker subsystem allows execution of long-running and possibly non-deterministic tasks (e.g. web requests, encryption/decryption and signing of data, random number generation, CPU-intensive computations, enumeration/aggregation of on-chain data, etc.) which could otherwise require longer than the block execution time. The code for the off-chain workers is stored on-chain, and has access to the on-chain environment, but is never executed as part of block-processing. Off-chain workers make it easy to run the correct code and allow longer running tasks to be performed without holding up the blockchain. Our privacy-preserving PDP scheme introduced in this article is run by Off-chain Worker.

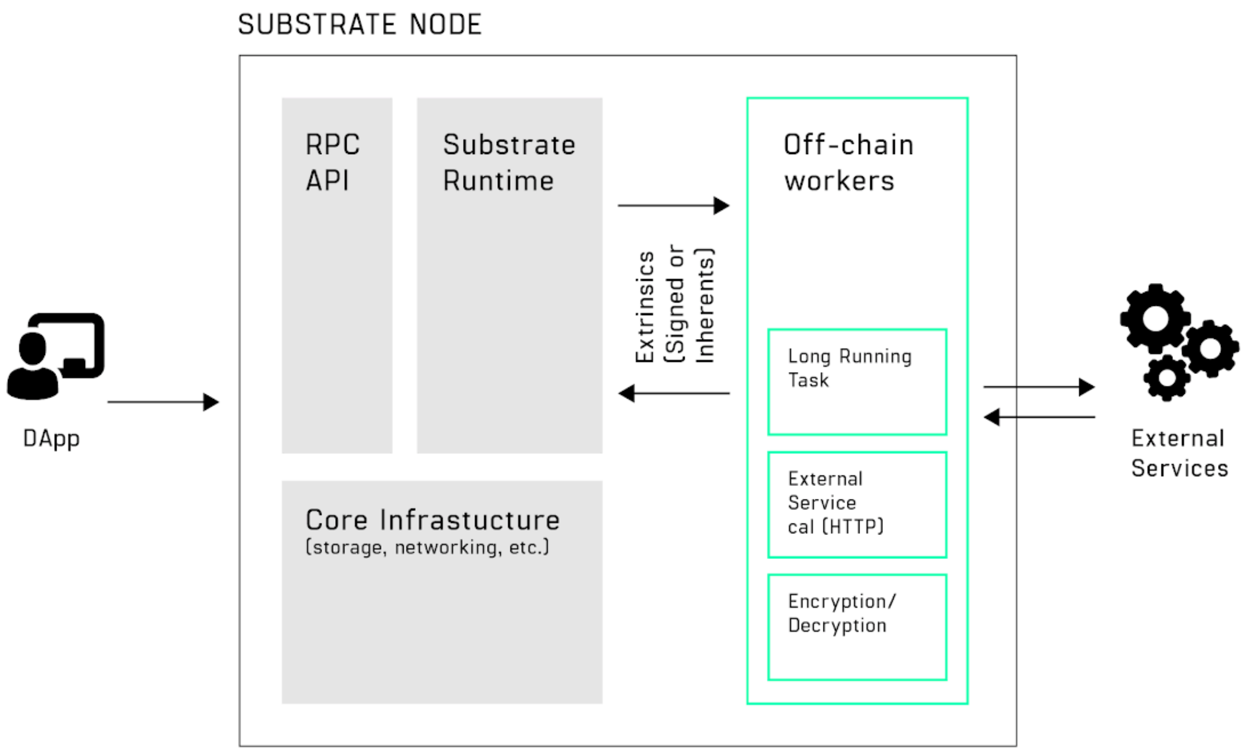


Fig. 5 Substrate node architecture

5.2 Substrate Runtime

The runtime of a blockchain is the business logic that defines its behavior. In Substrate-based chains, the runtime is referred to as the "state transition function"; it is where Substrate developers define the storage items that are used to represent the blockchain's state as well as the functions that allow blockchain users to make changes to this state.

Staking. The Storage Node must stake tokens to make sure the User data is correctly kept on the node. It will be slashed if it fails in the auditing. The Auditor Node must stake tokens as well to prove he/she is an honest auditor.

Randomness. Randomness is used to select an Auditor in the network for the auditing of each round.

Assest. We are going to issue a token ‘CDTC’ in the network for the incentives of the rational Storage Nodes and honest Auditor Nodes.

Reference

[1] H. Shacham and B. Waters, “Compact proofs of retrievability,” in Proc. of Asia crypt 2008, vol. 5350, Dec 2008, pp. 90–107.
[2] D. Boneh, B. Lynn, and H. Shacham, “Short signatures from the Weil pairing,” J. Cryptology, vol. 17, no. 4, pp. 297–319, 2004.
[3] A. L. Ferrara, M. Greeny, S. Hohenberger, and M. Pedersen, “Practical short signature batch verification,” in Proceedings of CT-RSA, volume 5473 of LNCS. Springer-Verlag, 2009, pp. 309–324.
[4] Zhang Fengzhe, Chen Jin, Chen Haibo, et al. Lifetime privacy and self-destruction of data in cloud [J]. Journal of Computer Research and Development, 2011, 48(7): 1155-1167.