2022 IEEE 9th International Conference on Cyber Security and Cloud Computing (CSCloud)/2022 IEEE 8th International Conference on Edge Computing and Scalable Cloud (EdgeCom)



# A Secure and Decentralized Auditing Scheme for Cloud Ensuring Data Integrity and Fairness in Auditing

Tariqul Islam[1], Kamrul Hasan[2], Saheb Singh[3], Joon S. Park[4],
[1,3,4]Syracuse University, Syracuse, NY, USA
[2] Tennessee State University, Nashville, TN, USA
Email: {*mtislam@syr.edu, mhasan1@tnstate.edu, ssingh55@syr.edu, jspark@syr.edu*}

***Abstract*—With the advent of cloud storage services many users tend to store their data in the cloud to save storage cost. However, this has lead to many security concerns, and one of the most important ones is ensuring data integrity. Public verification schemes are able to employ a third party auditor to perform data auditing on behalf of the user. But most public verification schemes are vulnerable to procrastinating auditors who may not perform auditing on time. These schemes do not have fair arbitration also, i.e. they lack a way to punish the malicious Cloud Service Provider (CSP) and compensate user whose data has been corrupted. On the other hand, CSP might be storing redundant data that could increase the storage cost for the CSP and computational cost of data auditing for the user. In this paper, we propose a Blockchain-based public auditing and deduplication scheme with a fair arbitration system against procrastinating auditors. The key idea requires auditors to record each verification using smart contract and store the result into a Blockchain as a transaction. Our scheme can detect and punish the procrastinating auditors and compensate users in the case of any data loss. Additionally, our scheme can detect and delete duplicate data that improve storage utilization and reduce the computational cost of data verification. Experimental evaluation demonstrates that our scheme is provably secure and does not incur overhead compared to the existing public auditing techniques while offering an additional feature of verifying the auditor's performance.**

***Keywords*:** Convergent Encryption, Confidentiality, Deduplication, Blockchain, Public Data Auditing, Procrastinating Auditor.

## I. Introduction

In the era of internet, it has become important that we are able to preserve the data not only for future references but also for historical records [1]. Due to the fast growing data capacity of devices, days of data deletion are gone [2]. Nowadays, most companies keep their historic data separate from their data warehouse. The most common way of storing data is in the cloud which provides a cheap way to store large amount of data without worrying about being able to handle the data storage, and data is easily available anywhere in the world with only internet connection [3]. But storing data in the cloud comes with its caveats such as losing control over your data, data privacy, and data integrity issues. The integrity of outsourced data is being put at risk in practice. For instance, CSPs may not feel comfortable to reveal the incidents of data corruption for maintaining their good reputation, or may delete a portion of data that is never accessed to reduce the storage costs. Furthermore, an external adversary may tamper with the outsourced data for financial or political reasons [4]. One of the ways we can ensure that our data stored in the cloud is not corrupted is auditing the CSP periodically.

The simplest way for a user to audit data is by downloading a random subset of data and comparing it with the metadata of the original data subset. While this is theoretically possible, it is not feasible for the user as it is computationally expensive when data size is huge. An alternate way is to authorize an auditor to perform auditing on behalf of the user. Auditor usually has a contractual agreement with the user to audit the CSP after every pre-defined period of time. Auditor takes a certain amount of fee for this task and the user does not need to worry about the integrity of the data. In a public verification scheme, the auditor is assumed to be honest and reliable. However, user also needs to be vigilant so that auditor and CSP do not collude with each other to hide information about corrupted data blocks [4]. To ensure security in the case that the auditor is compromised, the user is required to audit the auditor's behaviors– after each verification, the auditor is expected to record the information used to verify the data integrity, which can enable the user to verify the auditor's performance.

Most of the auditing schemes focus on auditing the CSP without thinking about data size. As data size is growing exponentially, it is becoming increasingly challenging to remove redundant data and maximize storage savings. A technique called $Data\ Deduplication$ [5] is being adopted by CSPs to improve storage utilization by storing only one copy of a file in the system. Using data deduplication user can not only save on storage cost but it is also convenient for auditor to audit only the unique data [6]. Existing auditing schemes also require data validation to be performed periodically so that any data corruption can be detected as early as possible. Any delay in detecting corrupt data might lead to a huge financial loss for the user. However, an irresponsible auditor may procrastinate on the scheduled verification due to network failures, system errors, or request from the CSP. We call this auditor a $Procrastinating\ Auditor$, it deviates from the main objective of the public verification schemes, i.e., detecting data corruption as soon as possible [1]. It might

978-1-6654-8066-6/22/$31.00 ©2022 IEEE
DOI 10.1109/CSCloud-EdgeCom54986.2022.00022

be too late to recover the data loss or damage if the auditor procrastinates on the verification. In fact, the procrastinating auditor cannot be detected in most of the current public verification schemes, even though malicious auditors can be detected [7]. Most of the existing public auditing schemes do not have fair arbitration system– in the case of data corruption, the user has to bear the financial loss. Therefore, it is essential to have a system that can not only punish malicious CSPs but also compensate user for the data loss. Moreover, existing public auditing schemes are either focused on the problem of procrastinating auditor or on the deduplication. Very few Blockchain-based auditing schemes exist that looked at fair arbitration for the user. And those who looked at fair arbitration are unable to punish the auditor for procrastination or colluding with the CSP for corrupting data.

**Objectives.** The purpose of our work is to propose a public auditing scheme that, i) resists malicious and procrastinating auditors, ii) provides users a fair arbitration system in case data integrity is lost, iii) guarantees secure access control through a robust key management technique, and iv) achieves storage efficiency by employing deduplication.

**Contributions.** Following are the main contributions of our work:

- We propose a cloud based public auditing scheme that blends public auditing with a fair arbitration system to achieve data integrity and fairness in auditing.
- Our scheme uses a Blockchain-based solution to protect users from the procrastinating auditor and guarantees that the auditor would not be able to collude with the CSP.
- The use of HCE2 (Hash–and–Convergent Encryption-2) algorithm ensures secure and efficient access control to cloud storage system and also makes our scheme resilient to several well known attacks.
- The use of smart contract (which are immutable after being deployed) in our scheme ensures that CSP is unable to collude with the auditor to hinder fair arbitration to the user.
- Our scheme supports deduplication which not only saves storage cost but also reduces the cost of auditing for the user. We use Ramp Secret Sharing scheme for key management, which requires less computational cost for sharing and recovering the secret key.

The rest of the paper is organized as follows: In Section II, we review some preliminaries and cryptographic primitives. In Section III, we describe our system model and the proposed scheme. In Section IV, we present implementation and evaluation of our scheme. We present some related work in Section V. Finally, we conclude the paper in Section VI.

## II. Preliminary Approaches and Related Algorithms

In this section, we provide formal definitions of the cryptographic primitives that form the foundation for our storage scheme. We also present the algorithms and protocols that we use in our proposed scheme.

**Fair Arbitration.** CSP receives storage fees for storing user data in the cloud, whereas auditor receives auditing fees for auditing the CSP. If CSP is not able to store data correctly which leads to corruption of stored data, user solely has to bear the loss incurred from this corruption. However, ideally, in case of data corruption, CSP should pay a certain amount to user as a compensation for mishandling the data. In a fair arbitration system, CSP bears some responsibilities for the loss incurred due to the corruption of the data file [8]. In this work, we integrate a fair arbitration system with our public auditing scheme using the concept of smart contract [4]. Since smart contracts are immutable, we are able to perform data verification in the smart contract and user pays the CSP for storage if auditing is successful. Similarly, CSP pays the user certain amount as compensation when user data get corrupted. This not only helps the user monetarily but also provides an extra motivation to the CSP to maintain data correctly.

**Deduplication.** Deduplication is a technique being adopted by many CSPs to improve storage utilization by storing only a single copy of a file or file block in a system [6]. Research [9] has shown that data deduplications can save up to nearly 75% in business applications storage and bandwidth costs [4]. Deduplication is achieved by using *convergent encryption*, which produces identical ciphertexts from the same plaintext, regardless of the users and the number of times it is encrypted. In *convergent encryption*, a user derives a *convergent key* by computing the hash of the file content and then encrypts the file with this key.

**Hash–and–Convergent Encryption-2 (HCE2).** Hash-and-CE2 (HCE2) is one of the message locked encryption techniques [10]. Using HCE2, a user generates convergent key by hashing the plaintext [4]. And then using this convergent key the data is encrypted. Thus, users with the same data will generate the same convergent key and using this key on the same plain-text, users will always get the same cipher-text. Therefore, CSP can realize data deduplication on the cipher-text. In our scheme, we use HCE2. The use of Hash-and-CE2 (HCE2) instead of Hash-and-CE (HCE) has some advantages. HCE2 uses guarded decryption which enables it to achieve "Tag Consistency" security. And, unlike HCE, HCE2 is resilient to erasure attacks.

**Ramp Secret Sharing Technique.** Secret sharing enables a group of $n$ shareholders to share a secret between them so that any $k$ participants can construct the secret, and *($k$ - 1)* or fewer participants cannot generate the secret [6], [11]. We utilize ramp secret sharing in our scheme to share the convergent encryption key between users having the same data. We use ramp secret sharing instead of Shamir Secret Sharing [12] as Shamir's has two limitations: i) a heavy computational cost is involved to create shares and recover the secret, and ii) a large storage capacity is needed to reconstruct all the shares [3]. Shamir secret sharing has a constraint of one share per shareholder, which makes computational cost really high in case of large number of shares for each shareholder. Contrarily, ramp secret sharing removes this constraint and reduces the load on individual stakeholder.

**Homomorphic Linear Authenticator.** Homomorphic Linear Authenticator (HLA) is a signature scheme widely used in cloud computing and storage server systems, which allows client who stored data in an untrusted server to verify that the server possesses the original data without retrieving it [13]. HLA allows CSPs to respond with an aggregated authenticator to the Auditor's challenge, hence reducing the communication complexity [14].

## III. Proposed Scheme

We consider a practical scenario where an organization uses a cloud storage service that facilitates staff members in the same department or group to read, write and share files. Our proposed scheme consists of the following four entities and the workflow is divided into seven main tasks as illustrated in Fig 1. At first, user prepares data to be outsourced to the CSP (task-1). Then, user delegates the auditing task to the auditor (task-2). Auditor generates challenge messages by utilizing the blockhashes of the Ethereum Blockchain (task-3) and sends those to the CSP (task-4). CSP then computes the corresponding proof messages and responds to the auditor (task-5). Auditor checks the validity of the proof messages to verify data integrity and records the audit log in the Blockchain (task-6). Finally, through task-7, user can audit the quality of the auditor by investigating the audit logs stored in the Blockchain.

### A. Essential Components of Our Proposed Scheme

**User**. *User* (i.e., *data owner*) is the owner of data and outsources data to the CSP. User is responsible for auditing the auditor using Blockchain log. User is also the owner of the smart contract and therefore is the only one who can delete smart contract. User deposits an initial amount to the smart contract which could be used later on to penalize the CSP if any corrupted data is found within a given time frame.

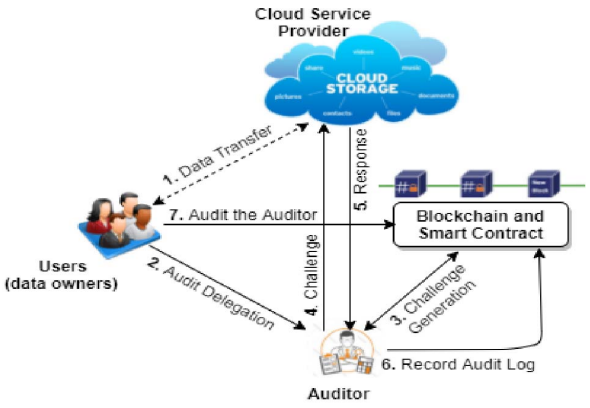


Fig. 1. Blockchain-based Cloud Auditing Model

**Auditor.** *Auditor* is responsible for auditing the CSP by sending the challenge blocks to the CSP using Blockchain hashes in regular intervals. Once auditor gets the response from the CSP, it forwards the challenge blocks from the CSP to the smart contract for auditing.

**Cloud Storage Provider (CSP).** *CSP* stores user data and responds to the challenge sent by the auditor. CSP also stores a deposit to the smart contract for facilitating fair arbitration to the user in case data get corrupted.

**Smart Contract.** *Smart Contract* is responsible for auditing the CSP using the challenge blocks it received from the auditor and the response provided by the CSP. If no data corruption is found, smart contract creates an empty transaction and records the auditing log. If data corruption is discovered, smart contract creates a transaction– sends an amount to the user as a compensation and records the log of this transaction for auditing purposes.

### B. Hash-and-Convergent Encryption-2 Operations (HCE2)

Our scheme uses HCE2 encryption which is a type of convergent encryption. We define HCE2 with the following four fundamental operations:

1) $KeyGenHCE2(P_i, F_i) \rightarrow K_i$: This HCE2 key generation algorithm takes a file block $F_i$ and public parameter $P_i$ as its input and outputs a convergent key $K_i$.
2) $EncryptHCE2(K_i, F_i) \rightarrow Cipher_i$: This encryption algorithm takes as input a file block $F_i$, the convergent key $K_i$, and produces a ciphertext $Cipher_i$, such that only users having $K_i$ will be able to decrypt $F_i$.
3) $DecryptHCE2(Cipher_i, K_i) \rightarrow F_i$: This decryption algorithm takes the ciphertext and the convergent key as its inputs. It produces the original file block $F_i$.
4) $TagGenHCE2(Cipher_i) \rightarrow Tag_i$: This tag generation algorithm takes as input a ciphertext $Cipher_i$ and outputs a tag $Tag_i$.

### C. Bilinear Groups

Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_\mathbb{T}$ are three different multiplicative cyclic groups of the order $p$. We use $g_1$ and $g_2$ to denote the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$. Following are the properties it holds.

1) $Computational$: There exists an efficient computational algorithm which is capable of computing map $e$.
2) $Bilinearity$: For all a $\in \mathbb{G}_1$, b $\in \mathbb{G}_2$, x and y in $\mathbb{Z}_p$, e($a^x$, $b^y$) is equal to e($a$, $b$)$^{xy}$.
3) $Non-degeneracy$: e($g_1$, $g_2$) is not equal to 1.

TABLE I
Notations.

| Notation | Description |
|---|---|
| $U_i$ | Cloud User $i$ |
| $F$ | Plain text file of the User |
| $F_i$ | Blocks of plain text files of the User |
| $K_i$ | Convergent key for block $F_i$ |
| $Tag_i$ | Authentication tag of block $F_i$ |
| $Cipher_i$ | Encrypted Ciphertext of block $F_i$ |
| $Sig(.)$ | Signature generating algorithm |
| $H(.)$ | Secure hash function |
| $h(.)$ | Secure hash function |
| $\mathbb{G}_1$ | Multiplicative cyclic group |
| $\mathbb{G}_2$ | Multiplicative cyclic group |
| $\mathbb{G}_T$ | Multiplicative cyclic group |
| $f$ | Pseudorandom function: $\{0, 1\}^* \rightarrow n$ |

*D. Our Implementation*

In this section, we describe the overall structure and the algorithms implemented in our scheme. We use the notations shown in TABLE I to present our system.

**Algorithm 1** Data Encryption

**Input:** Files F
**Output:** ($Key_i$, $Cipher_i$, $Tag_i$)
*Initialization* : Public Parameter $P$
1: $\{F_i\} \leftarrow$ Split$(F, n)$ {split file $F$ into $n$ blocks}
2: **for** each $F_i \in F$ **do**
3: $Key_i \leftarrow$ KeyGenHCE2$(P_i, F_i)$
4: $Cipher_i \leftarrow$ EncryptHCE2$(Key_i, F_i)$
5: $Tag_i \leftarrow$ TagGenHCE2$(Cipher_i)$
6: **end for**

**Data Encryption.** This algorithm (Algorithm 1) consists of the following steps. The user delegates the auditing to the third party auditor. The user and the auditor comes to an agreement about the frequency of the verification to be performed by the auditor. The third party auditor is considered to be honest but curious. Using encryption algorithm (Algorithm 1), user encrypts the data and sends ($Key_i$, $Cipher_i$, $Tag_i$) for each file block to the CSP. Due to the intrinsic property of convergent encryption, the users with the same data always generate the same encryption key and thus facilitates deduplication operation. The user first randomly chooses a key pair (Public K, Private K) as the public key ($p_k$) and private key ($s_k$) for signing. In line 1 of Algorithm 1, user splits the file F into a set of chunks ($\{F_1\}$, $\{F_2\}$, ... ,$\{F_n\}$). The user takes each file block and generates keys using the KeyGenHCE2 method (lines 2-3). Once key is generated user encrypts the data using the EncryptHCE2 method (line 4). Finally, the user generates the tag of the encrypted data using TagGenHCE2 method (line 5) and stores the tag for future verification. The user then computes $\sigma \leftarrow (H(W_i).u^{C_i})^x$ and $t \leftarrow name \mid\mid Sig_{sk}(name)$ and sends the ciphtertext and $\sigma$ to the CSP. Once the data has been uploaded, CSP makes a comparison between the newly uploaded data and stored data. If the same data is already stored, CSP no longer stores the new data.

**Algorithm 2** Auditor Challenges the CSP

**Input:** Blockhashes for $c$ blocks at time $t$ in $\{B_t, B_{t-1},\ldots, B_{t-c}\}$
**Output:** Chal $\leftarrow \{(a_c, n_{ac}), \{B_t, B_{t-1},\ldots, B_{t-c}\}\}$
1: **for** i $\leftarrow 0$ to $N-1$ **do**
2: **for** $y \leftarrow 0$ to $c-1$ **do**
3: $a_i \leftarrow$ f$(B_{t-y}||$i$)$ {f is a pseudorandom function}
4: $n_{ai} \leftarrow$ H$(B_{t-y}||a_y)$ {H is a hash function}
5: **end for**
6: **end for**

**Auditor Challenge Algorithm.** The auditor is required to audit the CSP after a pre-defined time interval. This algorithm (Algorithm 2) consists of the following steps. Using the current blockhash in the Blockchain, auditor generates $a_i$, a c-element random subset (lines 1-3), which represents the different files selected at random to be audited. In line 4, auditor generates a random number $n_{ai}$ using current blockhash in the Blockchain which represents the blocks in the files selected to be audited. The auditor finally generates the challenge and sends it to the CSP.

**Algorithm 3** CSP's Response to Challenge

**Input:** $Chal \leftarrow \{(a_c, n_{ac}), \{B_t, B_{t-1},\ldots, B_{t-c}\}\}$
**Output:** $\{\mu, \sigma,$ R$\}$
*Initialization*: Cyclic Group $G_1$, $G_2$, $G_t$
$G_t \leftarrow G_1$x $G_2$ {Bilinear Mapping}
1: $u \leftarrow G_1$ {Random element}
2: $x \leftarrow G_t$ {Random element}
3: $v \leftarrow g^x$ {g is the generator of bilinear mapping}
4: $R \leftarrow$ e$(u, v) \in G_t$
5: $\omega \leftarrow \sum_{a_i} n_{ai} C_{ai}$
6: $\mu \leftarrow$ h$(R)\,\omega$ {h is a secure hash function}
7: $\sigma \leftarrow \prod_{a_i \in i} \sigma_{a_i}^{n_{ai}}$ {Generating aggregated authenticator}

**CSP's Response to Challenge Algorithm.** After receiving the challenge from the auditor, CSP computes the response for the different blocks in different files by generating a linear combination of sampled blocks (lines 1-5 of Algorithm 3). In lines 6-7, CSP generates an aggregated authenticator. CSP then sends the aggregated authenticator along with the linear combination of the blocks to the smart contract for verification.

**Smart Contract Algorithm.** Smart contract receives the challenge-response (of the CSP) from the auditor and tag $t$ from the user; where, $t$ is the tag of the file generated by the user before sending data to the CSP. Smart contract first verifies the signature (line 2 of Algorithm 4) of the files sent by the CSP by comparing it with the user file tag using $p_k$. If the verification fails, smart contract sets $Y = 0$ (line 3), verifies that the auditing has failed as CSP has not sent the correct data required for auditing and stops the auditing. If verification is passed, smart contract then computes $a$ and $b$ (lines 6-8). In line 9, it checks if both $a$ and $b$ are similar to each other. If $a$ and $b$ are not similar then it executes lines 10-12, sets $Y = 0$ (line 10), and stops the auditing. If both $a$ and $b$ are equal to each other, then it executes lines 14-16 and sets $Y = 1$ (line 14). After the auditing is done, if $Y$ is set as 0, it means that auditing has failed and data has been found corrupted. In this case, CSP's deposit is sent to the user as compensation for the data loss. If $Y = 1$, it means that the auditing has passed successfully. In this case, user's deposit is sent to the CSP for maintaining data integrity. The smart contract sets the current blockhash used for auditing and the proof of verification as a log entry, stores the entry to the log file, and creates a transaction that transfers 0 deposit from its account to the user's account. This transaction can be used by the user to check for procrastinating auditor by matching the blockhash of the Blockchain at the time of the auditing

and checking the transaction records in the Blockchain.

**Algorithm 4** Smart Contract

**Input:** tag t, $\{\mu, \sigma, \mathrm{R}\}$ of all challenged files.
**Output:** Result of Integrity Y

**for** each $F_i \in F$ **do**
  **if** (`Verification(t)` != $t$) **then**
    $Y = 0$
    break
  **end if**
  $z \leftarrow$ `h(R)` {h(.) is a hash function}
  $a \leftarrow$ `e(`$\sigma^z$`, g)`
  $b \leftarrow$ `e(`$(\prod_{a_i \in i} H(W_{ac})^n_{ac})^z$`, v)`
  **if** $(a! = b)$ **then**
    $Y = 0$
    Send deposit to User as Compensation
  **end if**
  **if** $(a == b)$ **then**
    $Y = 1$
    Send deposit to Cloud Service Provider as fee
  **end if**
**end for**

## IV. Implementation and Evaluation

In this section, we provide experimental evaluation of our scheme. We implemented our scheme in Python programming language by using Crypto library [15] and Solidity v0.5.13 [16]. The test environment is Intel Core i9 (2.30 GHz and 32GB RAM) and Macintosh Big Sur v11.2.1. We test our program in the Remix-IDE [17].

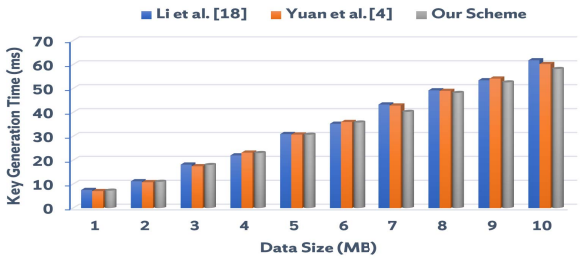


Fig. 2. Key Generation Time

### A. Key Generation Time

To verify the performance of key generation, we compared our scheme with Yuan et al.'s [4] and Li et al.'s scheme [18]. We considered file size ranging from 1MB to 10MB in this case. We observed almost identical performances (1MB to 6MB files) for all three schemes. Fig. 2 shows that for files larger than 6MB, our scheme performed a little better.

### B. Encryption vs Decryption Time

Our scheme uses homomorphic linear authentication of the outsourced data on the cipher text which is different to that used in existing data auditing schemes [13], [18]. To test encryption and decryption time, we use AES-256 algorithm, where the user data size varies from 1MB to 8MB. The encryption and decryption time are plotted in Fig. 3 and Fig. 4 respectively. Our scheme performed better in both categories.

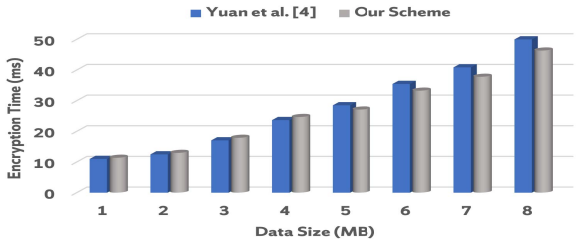


Fig. 3. Encryption Time

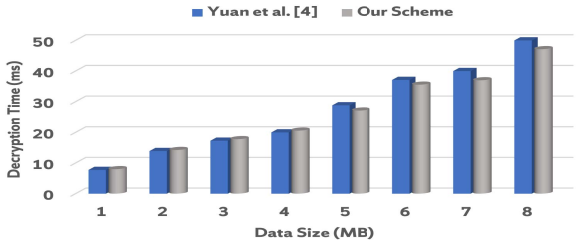


Fig. 4. Decryption Time

### C. Total Upload Time

We have taken the combination of key generation, data encryption, tag generation, and data upload time as the "Total Upload Time". Comparing our scheme with the Yuan et al.'s [4] and Wang et al.'s scheme [13], we achieved slightly improved performance. Comparison is shown in Fig. 5.

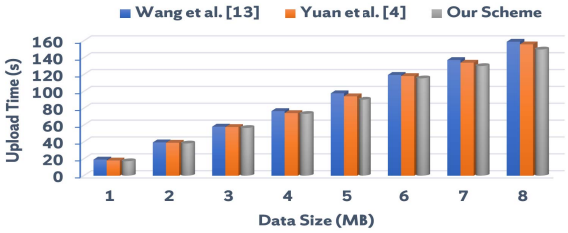


Fig. 5. Total Upload Time

### D. Overall Gas Cost for Integrity Verification and Checking Fairness in Auditing

To achieve fair arbitration, we wrote a smart contract using Solidity to verify the audit logs of our data with the CSP. Since our scheme also has the additional benefit of auditing the auditor (for procrastinating) along with data integrity verification, we needed a little more gas as compared to Yuan et al's [4] and Wang et al's [13] scheme. For the smart contract, we have modified Yuan et al's scheme [4] to add the feature of "auditing the procrastinating auditor" along with data integrity verification. The term $Gas$ here refers to the cost of compiling the smart contract and performing a transaction. Gas prices are denoted in $Gwei$ where each $Gwei$ is equal to 0.000000001 ETH ($10^{-9}$ ETH). ETH is the native cryptocurrency of the

Etherium platform. Our scheme uses 2 x $10^5$ more $Gwei$ as compared to Wang et al's [13] and Yuan el al's [4] scheme for 400 file blocks. This is only an increase of $1\%$ in $Gas$ cost and is needed for storing the balance of the deposit and performing transactions which the user will use later on to verify if the auditor is procrastinating or not. For a total of 400 blocks with different data sizes, we observed the gas cost in $Gwei$ to be 1.8 x $10^7$ (0.018 Ether) approximately. With the increase in file size, this is expected to be increased as larger file will require more computational $Gas$ to compute the smart contract. We compared the $Gas$ cost in Fig. 6.

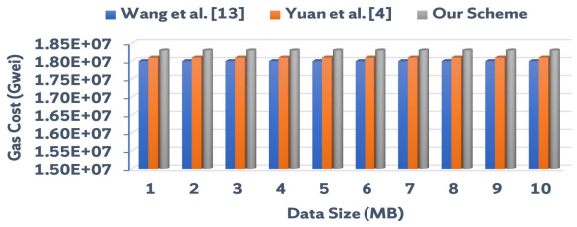


Fig. 6. Overall Gas Cost

## V. Comparison With Related Work

Most of the existing public auditing schemes [1], [4], [13], [19] are focused on helping either the third party auditor or the CSP. Wang et al. [13] presented a scheme for auditing the CSP. Their scheme is efficient in detecting corrupted data; however, it is vulnerable to procrastinating auditor, and does not provide user with any fair compensation. It does not give any financial incentive to the CSP for maintaining data integrity also. Yuan et al. [4] and Jin et al. [19] proposed auditing schemes that provide user compensation if data integrity is not maintained by the CSP. While these schemes focus on maintaining data integrity, they are vulnerable to procrastinating auditor. If auditing is not performed regularly, by the time data corruption is detected by the auditor, user might have suffered more loss than she would be compensated. Unlike their schemes, our scheme simultaneously achieves data integrity and fairness in auditing.

Zhang et al. [1] proposed a public auditing scheme for protecting users against procrastinating auditor. If the auditor is procrastinating and some data blocks are found to be corrupted, only the user has to bear the cost of data loss. Contrarily, our scheme takes care of the user if the data is corrupted by the CSP, by making CSP liable to compensate for the data loss. This not only helps the User but also provides an extra financial incentive to the CSP for maintaining data integrity.

## VI. Conclusion

In this paper, we proposed a secure and efficient public auditing scheme that employs a fair arbitration system to tackle the problems caused by the procrastinating auditor. Our scheme uses HCE2 algorithm that enables deduplication, and thus saves extra cost of storing redundant data and performing auditing. Moreover, the use of ramp secret sharing ensures secure authentication and access control of outsourced data. Experimental results show that our scheme performs well in terms major cryptographic operations. Our prototype implementation of fair arbitration with protection against procrastinating auditor demonstrates that it incurs only $1\%$ overhead in terms of $Gas$ cost compared to the existing techniques while providing an additional important feature of auditing the auditor.

## References


[1] Y. Zhang, C. Xu, X. Lin, and X. Shen, "Blockchain-Based Public Integrity Verification for Cloud Storage against Procrastinating Auditors," *IEEE Transactions on Cloud Computing*, March 2019.

[2] S. Keelveedhi, M. Bellare, and T. Ristenpart, "Secure Deduplication with Efficient and Reliable Convergent Key Management," *IEEE Transactions on Parallel and Distributed Systems*, vol. w5, no. 6, pp. 1615–1625, 2014.

[3] T. Islam, H. Mistareehi, and D. Manivannan, "SecReS: A Secure and Reliable Storage Scheme for Cloud with Client-Side Data Deduplication ," in *Proc. of the 2019 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2019, pp. 1–6.

[4] H. Yuan, X. Chen, J. Wang, J. Yuan, H. Yan, and W. Susilo, "Blockchain-based public auditing and secure deduplication with fair arbitration," *Information Sciences*, vol. 541, no. 4, pp. 409–424, Dec. 2020.

[5] J. Douceur, A. Adya, W. Bolosky, D. Simon, and M. Theimer, "Reclaiming Space from Duplicate Files in a Serverless Distributed File System," in *Proc. of the 22$^{nd}$ International Conference on Distributed Computing Systems*, July 2002, pp. 617–624.

[6] T. Islam, K. Lim, and D. Manivannan, "Blending Convergent Encryption and Access Control Scheme for Achieving A Secure and Storage Efficient Cloud ," in *Proc. of the 2020 IEEE 17$^{th}$ Annual Consumer Communications Networking Conference (CCNC)*, Jan. 2020, pp. 1–6.

[7] J. Li, Y. Li, X. Chen, P. Lee, and W. Lou, "A Hybrid Cloud Approach for Secure Authorized Deduplication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1206–1216, May 2015.

[8] Y. Zhang, C. Xu, N. Cheng, H. Li, H. Yang, and X. Shen, "Chronos$^{+}$+: An Accurate Blockchain-Based Time-Stamping Scheme for Cloud Storage," *IEEE Transactions on Services Computing*, vol. 13, no. 2, pp. 216–229, Oct. 2020.

[9] M. Dutch, "Understanding data deduplication ratios," *SNIA Data Management*, no. 2, 2008.

[10] M. Bellare and S. Keelveedhi, "Interactive Message-Locked Encryption and Secure Deduplication," in *Proc. of the 18$^{th}$ IACR International Conference on Practice and Theory in Public-Key Cryptography*, Mar. 2015, pp. 516–538.

[11] L. Bai, "A Strong Ramp Secret Sharing Scheme Using Matrix Projection ," in *Proc. of the International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'06)*, 2006, pp. 5–6.

[12] A. Shamir, "How to Share a Secret," *Communications of the ACM*, vol. 22, no. 1, pp. 612–613, Jan. 1979.

[13] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-Preserving Public Auditing for Secure Cloud Storage," *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 362–375, 2013.

[14] S. Liu and K. Chen, "Homomorphic Linear Authentication Schemes for Proofs of Retrievability ," in *Proc. of the 2011 3$^{rd}$ International Conference on Intelligent Networking and Collaborative Systems*, 2011, pp. 258–262.

[15] "Crypto library for Python," https://pythonhosted.org/pycrypto/Crypto-module.html, Tech. Rep.

[16] "Solidity," https://docs.soliditylang.org/en/v0.8.4/, Tech. Rep.

[17] "Ethereum, browser-only ethereum ide and runtime environment," https://remix.ethereum.org, Tech. Rep., 2021.

[18] J. Li, J. Li, D. Xie, and Z. Cai, "Secure Auditing and Deduplicating Data in Cloud," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2386–2396, 2016.

[19] H. Jin, H. Jiang, and K. Zhou, "Dynamic and Public Auditing with Fair Arbitration for Cloud Data," *IEEE Transactions on Cloud Computing*, vol. 6, no. 3, pp. 680–693, July-Sept 2018.