Secure and Efficient Proof of Ownership Scheme for Client-Side Deduplication in Cloud Environments

Amer Al-Amer[1], Osama Ouda*[1,2]
Department of Computer Science, College of Computer and Information Sciences, Jouf University, Saudi Arabia[1]
Department of Information Technology, Faculty of Computers and Information, Mansoura University, Egypt[2]

***Abstract*—Data deduplication is an effective mechanism that reduces the required storage space of cloud storage servers by avoiding storing several copies of the same data. In contrast with server-side deduplication, client-side deduplication can not only save storage space but also reduce network bandwidth. Client-side deduplication schemes, however, might suffer from serious security threats. For instance, an adversary can spoof the server and gain access to a file he/she does not possess by claiming that she/he owns it. In order to thwart such a threat, the concept of proof-of-ownership (PoW) has been introduced. The security of the existing PoW scheme cannot be assured without affecting the computational complexity of the client-side deduplication. This paper proposes a secure and efficient PoW scheme for client-side deduplication in cloud environments with minimal computational overhead. The proposed scheme utilizes convergent encryption to encrypt a sufficiently large block specified by the server to challenge the client that claims possession of the file requested to be uploaded. To ensure that the client owns the entire file contents, and hence resist collusion attacks, the server challenges the client by requesting him to split the file he asks to upload into fixed-sized blocks and then encrypting a randomly chosen block using a key formed from extracting one bit at a specified location in all other blocks. This ensures a significant reduction in the communication overhead between the server and client. Computational complexity analysis and experimental results demonstrate that the proposed PoW scheme outperforms state-of-the-art PoW techniques.**

***Keywords*—Client-side deduplication; proof of ownership; convergent encryption; could storage services**

I. Introduction

Cloud computing is the provision of on-demand access to different computing services and resources, such as storage space, servers, networks, databases, and software, over the internet [1]. The different models of cloud computing can provide a wide range of capabilities, adapted with different business goals, to diverse clients and/or consumers [2, 3]. The rapid development and integration of cloud computing have led organizations, institutions, and individuals to increasingly turn to utilize services provided over the cloud [4]. Consequently, an increasing number of individuals and organizations tend to move their data on cloud storage services (e.g., Dropbox, SkyDrive, Google Drive, iCloud, Amazon S3). This resulted in rapid growth in the volume of data that is stored on the cloud storage servers [5, 6, 7].

To increase efficiency as well as reduce the storage space required on storage servers, cloud storage providers tend to avoid downloading and uploading duplicated data [5, 8]. Data deduplication is an effective mechanism that aims at reducing the required storage space of the cloud storage servers by avoiding storing several copies of the same data. There are two main types of deduplication [9, 10] namely, server-side deduplication and client-side deduplication. The server-side deduplication schemes [11, 12] remove duplicated copies of the same files after uploading them to the server. On the other hand, In client-side deduplication [10, 13], duplicated copies are identified on the client side and not uploaded to the server. Hence, in contrary to server-side deduplication, client-side deduplication can not only save storage space and uploading time but also reduce network bandwidth.

However, client-side deduplication schemes might suffer from serious security threats [13, 14, 15]. For instance, an adversary can spoof the server and gain access to a file that he/she does not possess by claiming that he/she owns it. To thwart such a threat, Halevi et al. [16] proposed a cryptographic primitive, referred to as "proof of ownership" (PoW), to allow the server to verify whether a client owns a file. They pointed out that a robust PoW scheme should alleviate potential security threats without introducing I/O and computational overhead at both client and server sides. Since the introduction of the proof-of-ownership concept, several PoW schemes have been proposed in the past few years [14, 16, 17, 18, 19, 20, 21, 22]. However, the security of such schemes cannot be assured without affecting the computational complexity of client-side deduplication.

An efficient PoW scheme should satisfy several requirements. First, the chances that an adversary successfully passes a PoW run should be negligible if the adversary does not possess the file in its entirety. Second, a small fixed amount of information should be loaded on the server-side regardless of the file size. Third, the amount of processed information on both the client and server sides should be minimal. In addition, the amount of transmitted data between the client and the server should be reduced to minimize the bandwidth. Unfortunately, the PoW schemes discussed above do not fulfill all of these requirements. Thus, new techniques that can resolve the security-efficiency trade-off and reduce communication and storage overhead should be proposed.

This paper proposes a secure and efficient proof-of-ownership scheme for client-side deduplication in cloud environments that fulfills the requirements mentioned above. The proposed technique utilizes convergent encryption to encrypt a sufficiently large block specified by the server to challenge the client that claims possession of the file requested to be uploaded. To ensure that the client owns the entire file contents, and hence resists collusion attacks [23, 24], The server challenges the client by requesting him to split the file he/she asks to upload into fixed-sized blocks and then encrypts

a randomly chosen block using a key formed from extracting one bit at a specified location in all other blocks. This ensures a significant reduction in the communication overhead between the server and client. Moreover, the proposed scheme resists attacks of honest-but-curious servers [25, 26, 27].

The rest of the paper is structured as follows. Section II provides a brief background on the concepts of data deduplication, convergent encryption, and proof-of-ownership. Section III describes the proposed PoW scheme in detail. Section IV presents a computational complexity analysis of the proposed scheme and provides a comparison with the state-of-the-art schemes. Section V describes the experimental results and discussion. Finally, Section VI concludes the paper.

II. Background

A. Data Deduplication

Data deduplication, also called single-instance storage, techniques aim to eliminate duplicate copies of the same data to improve storage utilization and reduce the unnecessary cost of storage capacity needs [13]. A prominent example of the usefulness of data deduplication is redundant file attachments in email systems. Consider a typical email system containing 50 copies of the same 20 megabyte (MB) file attachment. Saving or archiving this email platform requires 1000 MB of storage space. The storage demand can drop to only 20 MB if data deduplication is employed. The example shown in Fig. 1 demonstrates the main concept of data deduplication. There are two main types of data deduplication in cloud environments: server-side deduplication and client-side deduplication. Server-side deduplication techniques identify repeated data after uploading them to the server, whereas client-side deduplication techniques identify duplicate copies of data before they are uploaded to the server. Therefore, client-side deduplication techniques can reduce network data transfers in addition to storage capacity.

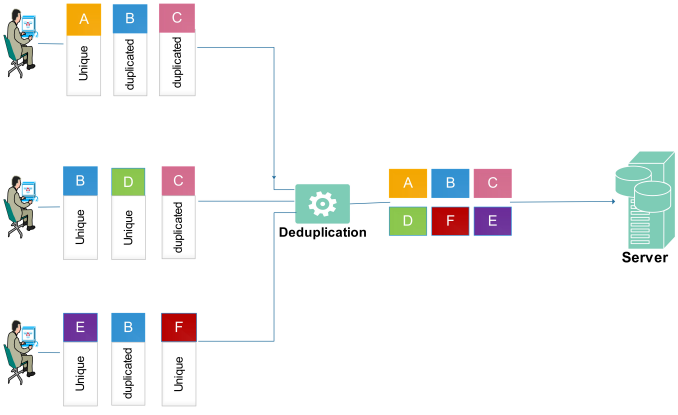

Fig. 1. Data Deduplication Technology.

B. Convergent Encryption

Data deduplication techniques can benefit from convergent encryption (CE) to achieve security smoothly and more easily [10, 15]. Convergent encryption is a cryptographic concept that ensures security in the cloud by achieving confidentiality and data privacy. The main idea behind CE is to create similar ciphertexts from similar plaintexts (see Fig. 2). Unlike traditional cryptography, in which data encryption and decryption are carried out using cryptographic keys that are independent of the data being encrypted, and hence different ciphertexts are obtained from the same plaintexts, CE ensures that the same key is used for the same plaintexts. In CE, the data digest or hash is used as a key to encrypt the data. Encrypting data in CE undergoes the following three steps: 1) the digest (hash value) of the plaintext in question is computed, 2) the plaintext is then encrypted using its digest as a key, and 3) finally, the hash is encrypted with a key chosen by the user and stored along with the obtained ciphertext. These steps ensure that identical data copies will generate the same key and the same ciphertext.

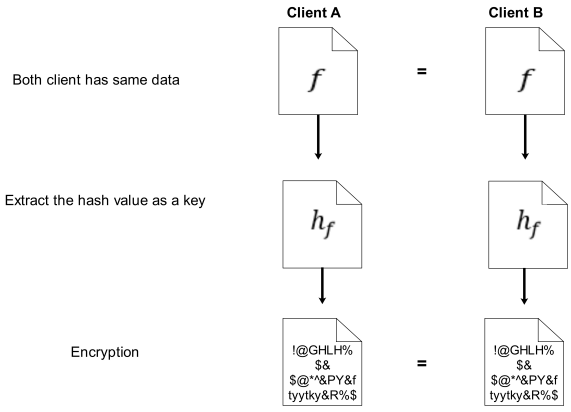

Fig. 2. Convergent Encryption Concept.

C. Proof of Ownership (PoW)

In client-side deduplication techniques, the hash value of the file requested to be uploaded by the client is firstly computed and sent to the server. If this hash value exists in the list of previously uploaded files to the server, the server will request the client not to upload the file again to avoid storing redundant data. However, in order to append the client to the list of owners of that file, the server has to verify that the client owns the entire file and not try to spoof it. Traditional proof of ownership protocols, Such as the one proposed by Halevy et al. [5] oud storage provider (CSP) has access to the original file. In other words, such protocols depend on trust between the cloud storage provider and the client. However, this trust might generate many potential security risks since cloud storage providers (CSPs) should not be fully trusted. The process of adapting PoW protocols so that they can work properly on encrypted data is an open problem so far [13].

Several PoW schemes have been proposed over the past few years. Gonzalez-Manzano et al. [14] proposed an attribute-based symmetric encryption proof of ownership scheme, referred to as ase-PoW, for hierarchical environments. The main goals of this scheme are to resist honest-but-curious servers and to provide flexible access control to ensure that users have access to sensitive files with the right and real privileges. The

idea behind this scheme is based on recursively encrypting parts of the file being uploaded to the server to assure its possession by the user. The ase-PoW scheme has the advantage of its ability to resist guessing attacks on the content and reduce the cloud workload. However, it does not take into account the issues of user revocation and key updating.

Dave et al. [17] proposed a secure proof of ownership scheme based on utilizing Merkle trees. The idea is based on calculating the responses of challenges in advance at the server-side to avoid computational overhead while uploading the file. The cloud server does not need to hold over the resources until the response is received, which is preferable to the utilization of stateless protocols. The user on the client-side is requested to encrypt the file to be uploaded to the server using its digest as a key (a.k.a. convergent encryption (CE)). Afterward, the user computes a file tag (usually a cipher-text hash) to check the file's existence on the server. The file uploading process consists of four stages (metadata generation, challenge generation, response generation, and response verification). If the file tag does not exist in the FileList kept by the server, the client will be requested to upload the file with the tag. In the case of subsequent uploads, the server sends an unused precomputed PoW challenge to the client. If the client owns the entire file, then the client will correctly respond to the server.

Islam et al. [18] proposed a secure and reliable storage scheme for cloud environments with client-side deduplication (SecReS). The authors combined convergent encryption and secret sharing techniques to achieve data confidentiality. They used the Reed-Solomon erasure code to achieve fault tolerance through distributing data to multiple storage servers. Moreover, Merkle trees are utilized to verify the ownership of data and to perform secure data deduplication. Mishra et al. [19] proposed a merged PoW scheme (MPoWS) for block-level deduplication in cloud storage. By employing a random test approach, MPoWS meets the requirements of client-side and server-side mutual verification. In MPoWS, large files are uploaded to the servers, and then their duplication is checked using various blocking flags. The authors used a random test approach to increase security and make it difficult to predict which block will be validated.

Fan et al. [20] proposed a secure deduplication scheme based on a trusted execution environment (TEE), which provides secure key management by using convergent encryption with cloud users' privileges. Trusted execution environments improve cryptographic systems' capability to resist chosen-plaintext and chosen-ciphertext attacks. The authors proposed assigning a set of privileges to every cloud user. Therefore, data deduplication can be performed if and only if the user of the cloud has the right and valid privileges. In [21], Ouda proposed a secure and effective proof of ownership scheme for client-side deduplication in the cloud. This scheme verifies if the client owns the entire file for which he/she claims possession. In other words, the proposed scheme does not allow an adversary to engage in a successful proof of ownership without fully owning the file's content. This can be achieved by requesting the client to encrypt the entire file using the file hash as the key before uploading the file to the server. This prevents the curious server from disclosing the file content.

Du et al. [22] proposed a proof of ownership and retrievability framework (PoOR) in which the cloud client can prove ownership of files to the server without uploading or downloading the files. The proposed framework consists of the pre-processing two phase, proof of ownership phase, and retrievability phase. The proof of ownership phase depends on the Merkle Tree protocol and comprises three steps: prove, challenge, and verify. Cui et al. [28] proposed a new attribute-based storage system that supports secure and efficient deduplication. The proposed system runs on a hybrid cloud environment where the private cloud is responsible for detecting identical copies for storage management. Ma et al. [29] demonstrated how attribute-based encryption can be used to minimize storage space and share data efficiently. In this technique, if the attributes of certain user is matched, then the user is given the right to decipher the encrypted data.

Blasco et al. [30] proposed a PoW scheme, called bf-PoW, that utilizes the Bloom filters to mitigate the server-side overhead. The main drawback of the bf-PoW scheme is that it does not consider data privacy. Di Pietro and Sorniotti [9] introduced another scheme, referred to as s-PoW in which the server requests clients to send bit-values of randomly selected bit positions of files requested to be uploaded by those clients. Although this scheme is computationally efficient at the client-side, it is not efficient on the server-side. Manzano and Orfila [31] proposed a PoW scheme, called ce-PoW, employing the concept of convergent encryption to encrypt file chunks before uploading them to the server. This scheme reduces issues related to key management. However, since the encryption is applied at the chunk level, the number of encryption keys increases linearly with the number of requested chunks. This can put a significant burden on both storage space and bandwidth as the security parameters increase.

Huang et al. [32] proposed a bidirectional and malleable proof-of-ownership scheme for large files in cloud storage (BM-PoW). The proposed BM-PoW protocol allows the server and user to interact to ensure ownership of the file to be uploaded even if the file is updated. Thus, secure and efficient deduplication for large files in static and dynamic archives is guaranteed. Miao et al. [33] proposed a novel PoW protocol that benefits from the distinguishable properties of chameleon hashing. Although this protocol is more efficient than existing PoWs based on Merkle hash tree, it is vulnerable to brute-force attack (BFA) due to its limited keyspace [34].

Although some solutions have been proposed to improve the efficiency at the server-side, other solutions tend to enhance the computational cost at the client-side. Besides, most of the existing schemes cannot satisfy all the security requirements in terms of resisting honest-but-curious servers and collusion attacks without affecting the efficiency and/or communication bandwidth requirements. Therefore, it is promising to study how to develop PoW schemes that can balance the trade-off between the security and efficiency requirements.

III. Proposed PoW Scheme

As we previously mentioned, the main goal of our proposed PoW scheme is to provide an efficient means to prove the ownership of files in client-side deduplication environments securely. Precisely, we aim at minimizing the exchanged

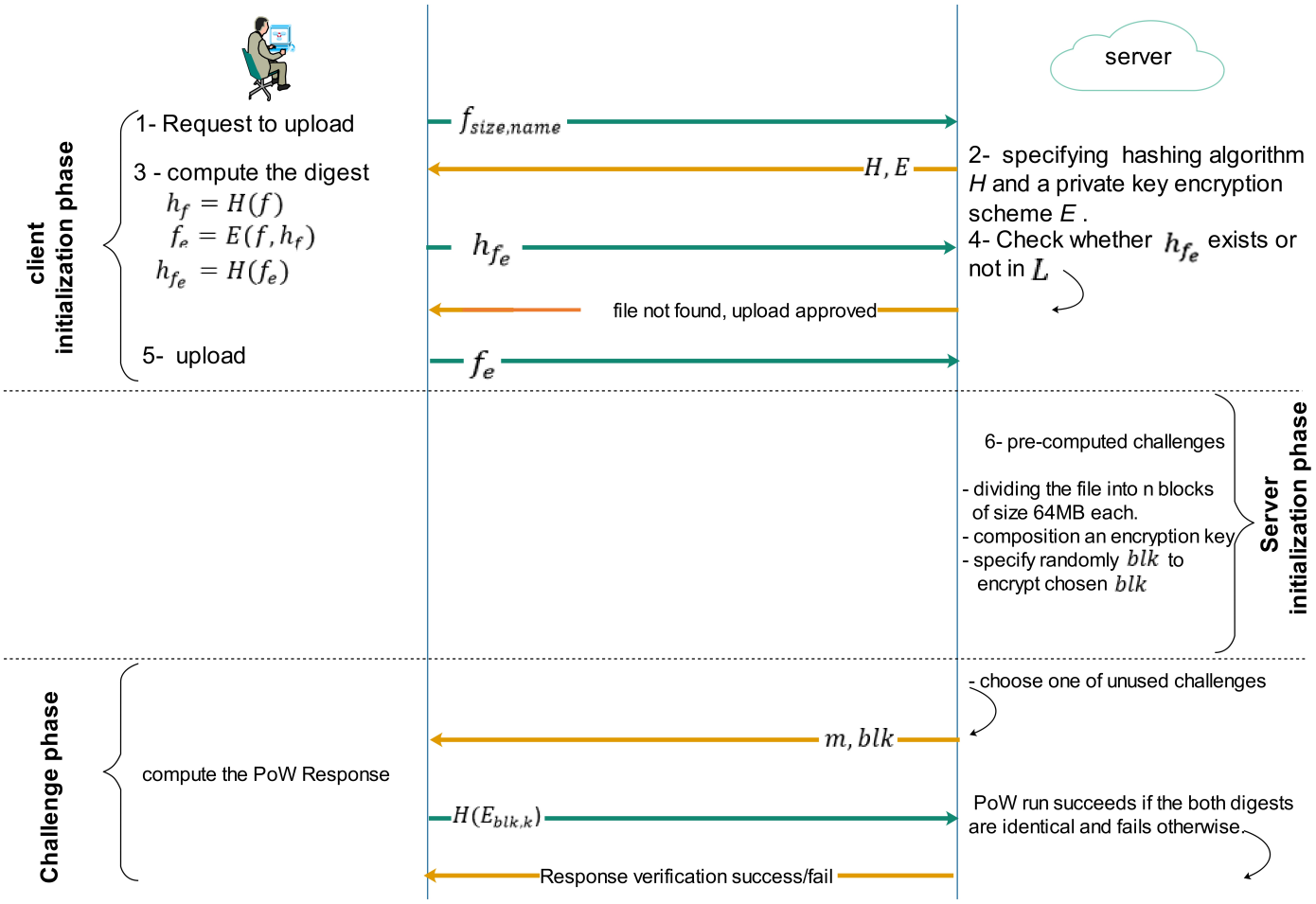

Fig. 3. Proposed PoW Scheme.

information between the client and server, reducing the data uploaded in memory during a PoW session, and decreasing the likelihood that a malicious client can successfully respond to the challenge sent to him/her by the server via increasing the amount of exchanged information between a malicious client and a legitimate owner of the file. Definition of abbreviations and symbols are given in Table I.

As illustrated in Fig. 3, the proposed PoW scheme consists of three phases: the client initialization phase, server initialization phase, and challenge phase. In the client initialization phase (see Algorithm 1), the client initiates a file (f) upload request to the server by simply sending general attributes of the file, such as its name and size. The server responds to the client by sending a message specifying a robust hashing algorithm $\mathcal{H}$ (e.g., MD5, SHA1, etc.) as well as a private key encryption scheme E (e.g., DES, AES, etc.). The client uses the specified hashing algorithm to compute the file digest, $h_f = \mathcal{H}(f)$, which is then used as a key to encrypt the file using the encryption algorithm specified by the server to obtain an encrypted file $f_e = E(f, h_f)$. The encrypted file f_e is then hashed using $\mathcal{H}$ to obtain its digest $h_{f_e} = \mathcal{H}(f_e)$. Finally, the client sends h_{f_e} to the server so that it can decide whether the file has been uploaded before by a different user.

TABLE I. DEFINITION OF ABBREVIATIONS AND SYMBOLS

Abbreviation	Definition
f	File to be uploaded to the server
f_e	Encrypted file
$\mathcal{H}(f)$	Hash of the file f
blk	A block of f
j	Block number
n	Number of non-overlapping blocks
κ	Encryption key
m	Bit position
$\mathcal{L}$	List of uploaded files
$\mathcal{C}$	The client
$\mathcal{S}$	The server

Assuming that the server stores the digest of each previously uploaded encrypted file, the server can decide whether f has been uploaded before by searching for h_{f_e} in the list $(\mathcal{L})$ of the stored digests. It is worth noting that our scheme

Algorithm 1 Client Initialization Phase

Input: File f
Output: $\mathcal{H}(f_e)$, f_e

Client $\mathcal{C}$ sends a request to server $\mathcal{S}$ to upload the file f
$\mathcal{S}$ sends to $\mathcal{C}$ the name of the hashing algorithm $\mathcal{H}$ and encryption algorithm E
$\mathcal{C}$ computes the digest $\mathcal{H}(f)$ of f using $\mathcal{H}$
$f_e \leftarrow E(f, \mathcal{H}(f))$
$\mathcal{C}$ computes $h_{f_e} = \mathcal{H}(f_e)$ and sends it to $\mathcal{S}$
$\mathcal{S}$ searches for h_{f_e} in the list ($\mathcal{L}$) of uploaded files
if $\mathcal{H}(f_e)$ is found in $\mathcal{L}$ **then**
    Go to the Challenge Phase
else
    Allow $\mathcal{C}$ to upload f_e to $\mathcal{S}$
    Go to the Server Initialization Phase
end if

Algorithm 2 Server Initialization Phase

Input: File f_e
Output: The entry $\mathcal{L}[h_{f_e}]$

Divide f_e into n blocks of size 64MB each
for $i \leftarrow 1$ to c **do** /* c = No. of challenges */
    Choose two integers m and $j < n$
    Extract the m-th bit of each block in f_e
    Generate a cryptographic key κ by concatenating the n extracted bits
    Encrypt the j-th block in f_e using κ to obtain $E(blk_j, \kappa)$
    Compute the digest of $E(blk_j, \kappa)$
    $Challange[i] =< m, j, \mathcal{H}(E(blk_j, \kappa)) >$
end for
Create a new entry for f_e consisting of all the generated challenges and append it to $\mathcal{L}$

assumes that files are encrypted before uploading them to the server to prevent honest-but-curious servers from disclosing the contents of the uploaded files. If h_{f_e} is not found in the list of the stored digests, the server sends a message to the client to start uploading f_e; otherwise, the server initiates the challenge phase.

After the file f_e is uploaded, the server initialization phase (see Algorithm 2) starts by creating a new entry for f_e. This entry consists of h_{f_e} and a pointer to a set of pre-computed challenges that will be used to prove the ownership of f_e by other clients who might request to upload the same file in the future. A challenge is created by dividing the file into n non-overlapping blocks sufficiently large to resist collusion attacks. It is assumed that sharing data $\geq$ 64MB among colluders would discourage them launch collusion attacks [30]. Thus, we recommend setting the block size at such levels. Then, the encryption key (κ) is composed by concatenating all bits at a specific position (m) across all blocks. For instance, if $m = 3$ and $n = 128$, then the third bit in each block is extracted, and the set of the 128 extracted bits are concatenated to create a 128-bit cryptographic key κ. An example that illustrates the key generation process, where $m = 3$ and the file and block sizes are 8 GB and 64 MB, respectively, is shown in Fig. 4. The generated key is then used to encrypt a randomly chosen block (blk_j) of the encrypted file f_e using the AES

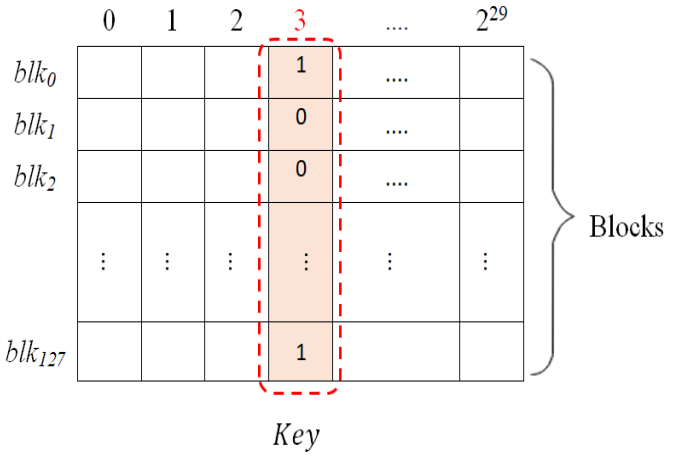

Fig. 4. An Example that Illustrates the Key Generation Process.

encryption algorithm to obtain $E(blk_j, \kappa)$. Finally, the digest of the encrypted block $\mathcal{H}(E(blk_j, \kappa))$ is obtained and stored along with m and j as one challenge for f_e. The previous challenge creation process is then repeated using different values of m and j to encrypt different blocks of f_e in order to generate as many challenges as needed.

The challenge phase is initiated when an entry is found for f_e in the list of uploaded files kept by the server. In this case, the server chooses one of the available challenges in $\mathcal{L}[h_{f_e}]$ to verify that the client possesses the file he/she requests to upload. The server challenges the client by sending m and j corresponding to the chosen challenge. Note that with just these two values, the amount of information sent from the server to the client is minimized. This satisfies an important design objective of the proposed PoW scheme. The client responds to the challenge by dividing the file, after encrypting it using the same procedure described in the client initialization phase, into n blocks, generating κ by extracting bits at position m of all blocks, encrypting the block blk_j specified by the server using κ, and finally calculating the digest of the encrypted block $\mathcal{H}(E(blk_j, \kappa))$ and sends it to the server as a response to the received challenge. The server matches the received block digest with the corresponding digest stored in $\mathcal{L}$. The PoW run succeeds if both digests are identical and fails otherwise.

It is important to note that all design goals of the proposed PoW scheme are satisfied. The data exchanged between the client and server are minimized. In the challenge phase, the server sends two small pieces of information; namely, the bit position index m used to generate the key κ and the index of the block to be encrypted. Similarly, the client is required to respond to the challenge received from the server by just sending the specified block digest. From the security perspective, the block size is set to 64MB because a PoW scheme is considered secure if the amount of exchanged information between a legitimate owner of f and a malicious client required to pass a PoW run is not smaller than 64MB [30]. Moreover, since one bit per block is used to generate the key (κ), a large number of challenges can be generated for each uploaded file. Precisely, more than 2^{29} different challenges can be generated if the block size is set to 64MB.

TABLE II. COMPARISON BETWEEN THE PROPOSED SCHEME AND FIVE RELATED POW SCHEMES CONCERNING SPACE AND COMPUTATIONAL COMPLEXITY. κ: SECURITY PARAMETER, n: NUMBER OF PRE-COMPUTED CHALLENGES, l: TOKEN LENGTH, F: FILE SIZE, B: BLOCK SIZE AND p_f: FALSE POSITIVE RATE (BF-POW SCHEME)

	ase-PoW	ouda-PoW	ce-PoW	bf-PoW	s-PoW	Proposed
Client computation	$O(B).Sym.n_r.hash$	$O(F).CE.hash$	$O(B).CE.hash.hash$	$O(F).hash$	$O(F).hash$	$O(B).CE.hash$
Server init computation	$O(B).hash$	$O(F).hash$	$O(B).hash.hash$	$O(F).hash$	$O(F)$	$O(B).hash$
Server init I/O	$O(F)$	$O(F)$	$O(F)$	$O(F)$	$O(F)$	$O(F)$
Server regular I/O	$O(0)$	$O(0)$	$O(0)$	$O(0)$	$O(n.k)$	$O(0)$
Server memory usage I/O	$O(n.l.k)$	$O(n.l)$	$O(n.l.k)$	$O(\frac{log(l/p_f)}{l})$	$O(n.k)$	$O(n.k)$
Bandwidth	$O(l.k)$	$O(l.k)$	$O(l.k)$	$O(\frac{l.k}{p_f})$	$O(k)$	$O(k)$

Algorithm 3 Challenge Phase

Input: m and j of an unused challenge
Output: PoW response

1: $\mathcal{S}$ sends m and j to $\mathcal{C}$
2: $\mathcal{C}$ divides f_e into n non-overlapping blocks of size 64MB each
3: $\mathcal{C}$ extract the m the bit of each block in f_e
4: $\mathcal{C}$ generate a cryptographic key κ by concatenating the n extracted bits
5: $\mathcal{C}$ encrypt the j the block in f_e using κ to obtain $E(blk_j, k)$
6: $\mathcal{C}$ send to the $\mathcal{S}$ the challenge $\mathcal{H}(E(blk, k))$
7: $\mathcal{S}$ reciving the challenge $\mathcal{H}(E(blk, k))$
8: **if** Clinet $\mathcal{H}(E(blk, k))$ = Server $\mathcal{H}(E(blk, k))$ **then**
9: PoW success
10: **else**
11: Fail to PoW
12: **end if**

IV. COMPLEXITY ANALYSIS

This section demonstrates how the proposed PoW scheme fulfills bandwidth and space efficiency requirements by comparing it to five well-known PoW schemes from the literature, as shown in Table II. Specifically, we compare the complexity of our proposed scheme by focusing on bandwidth, server memory usage, client/server computation, and I/O. It can be noticed from Table II that the complexity of the proposed scheme is similar to the complexity of the other schemes with respect to server initialization I/O as it primarily relies on the file size. For the regular server I/O, the complexity of the proposed scheme is also similar to the complexity of the other schemes except for s-PoW. Moreover, the complexity of the proposed scheme outperforms the complexity of the other schemes with respect to client computation and server initialization computation, mainly because the proposed scheme only encrypts a randomly chosen block in the file rather than encrypting the whole file. It is worth noting that this does not affect the security of the proposed scheme since the employed cryptographic key is extracted from all blocks of the file.

V. EXPERIMENTAL RESULTS

This section describes the experiments conducted to evaluate the performance of the proposed PoW scheme. All experiments were conducted on a personal computer with Intel Core i7-4770 CPU (2.4 GHz) and 8 GiB RAM. The performance of the proposed scheme was compared with the performance of the ce-PoW scheme proposed by Gonzalez-Manzano et al. [14], ase-PoW scheme proposed by Manzano and Orfila [31], Ouda-PoW scheme proposed by Ouda [21], bf-PoW scheme proposed by Blasco et al. [30], and s-PoW scheme proposed by Di Pietro and Sorniotti [9]. In all experiments, the schemes were evaluated using randomly generated test files of sizes ranging from 4MB to 2GB, doubling the size at each step. We used the C++ programming language for the implementation and utilized the OpenSSL cryptographic library [35] for the encryption and hashing operations, namely, AES (in counter mode) and SHA-256.

The clock cycles spent by the client to upload a file for the first time and to respond to the server challenge have been measured and compared with the corresponding clock cycles spent by the other tested PoW schemes. Fig. 5 shows the computational cost (clock cycles) spent in the client initialization phase for the four tested schemes. It can be noticed from the figures that the proposed PoW scheme outperforms all the other schemes. This is mainly because the s-PoW dealing with file level and the server requests from the clients to send bit-values of randomly selected bit positions of files requested to be uploaded. whereas in bf-PoW the clients compute a token for each segment index using the hash function, which in turn increments the executed operations. The ce-PoW scheme requires implementing multiple hashing and encryption operations separately for each file chunk. In ase-PoW, on the other hand, the client has to encrypt each part of the file chunks provided by the Attribute Certificate Service (ACS) symmetrically, whereas in the Ouda-PoW scheme, the entire file should be hashed twice and encrypted once. However, we can also notice that the performance of both the Ouda-PoW and ase-PoW schemes is close to the performance obtained by our proposed scheme with respect to the complexity of client

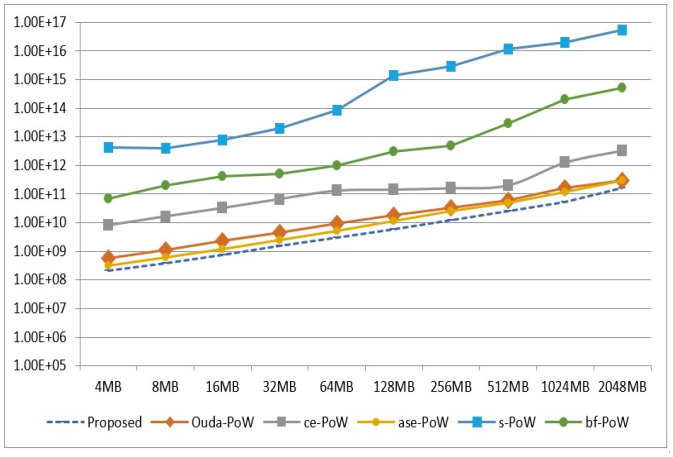

Fig. 5. Clock Cycles Required for the Client Initialization Phase.

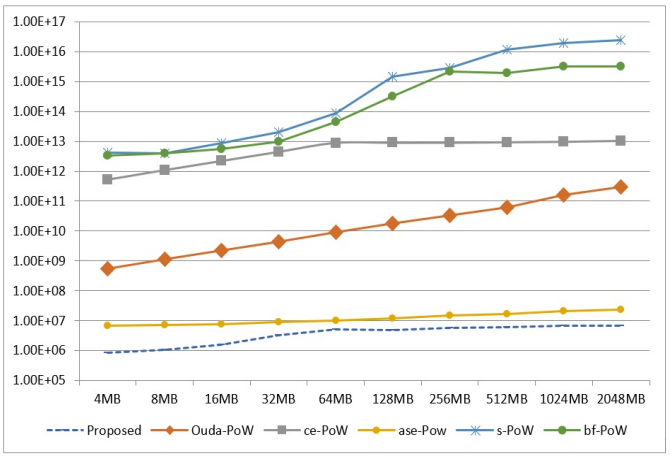

Fig. 6. Client Response Creation Clock Cycles.

initialization.

Fig. 6 shows the results obtained from comparing the proposed scheme with the other evaluated schemes regarding the time spent by the client responding to the server challenge. It is clear from the figure that the proposed scheme outperforms the other three schemes. This result is expected because the server in s-PoW uses a pseudorandom number generator to precompute the challenges that contain random file bits. In bf-PoW, the server initializes the bloom filter and divides the input files into chunks of fixed size, then creates the token chunks of each and inserts the function of each token into the bloom filters. In ce-PoW, the client encrypts all chunks specified by the server and then computes the hash over each encrypted chunk, increasing the computation time. The client in the ase-PoW scheme, on the other hand, performs several frequent encryptions of the designated chunks. In ouda-PoW, the client generates a random string of the same file size using the random seed received from the server, performs an XOR operation on the generated string with the CE file, and finally computes the hash of the resulting string. By contrast, in the proposed scheme, the client extracts the key from all blocks in the file and then encrypts only the block specified by the server.

VI. Conclusion

In this paper, we have proposed a secure and efficient proof-of-ownership scheme to thwart potential collusion attacks against client-side deduplication in cloud environments. The proposed PoW scheme's main idea is to divide the file to be uploaded into a number of fixed-sized blocks and then encrypt a randomly chosen block using a key formed by extracting one bit at a specified location in all other blocks. Unlike existing PoW schemes, the proposed scheme minimizes the exchanged information between the client and server and reduces the amount of data uploaded in memory during a PoW session. Moreover, it decreases the likelihood that a malicious client can successfully respond to the challenge sent to her by the server by increasing the amount of exchanged information between a malicious client and a legitimate file owner. The computational complexity of the proposed scheme was compared to five different PoW schemes, and experimental results showed that the proposed scheme outperforms the state-of-the-art PoW schemes concerning the time spent (clock cycles) for client initialization and response to the challenge received from the server.

Acknowledgment

The authors would like to thank the Deanship of Graduate Studies at Jouf University for funding and supporting this research through the initiative of DGS, Graduate Students Research Support (GSR) at Jouf University, Saudi Arabia.

References

[1] Ali Sunyaev. Cloud computing. In *Internet computing*, pages 195–236. Springer, 2020. doi:10.1007/978-3-030-34957-8_7.

[2] Chris Dotson. *Practical Cloud Security: A Guide for Secure Design and Deployment*. O'Reilly Media, 2019. URL https://www.oreilly.com/library/view/practical-cloud-security/9781492037507/.

[3] Srijita Basu, Arjun Bardhan, Koyal Gupta, Payel Saha, Mahasweta Pal, Manjima Bose, Kaushik Basu, Saunak Chaudhury, and Pritika Sarkar. Cloud computing security challenges & solutions-a survey. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 347–356. IEEE, 2018. doi:10.1109/CCWC.2018.8301700.

[4] Omer K Jasim Mohammad. Recent trends of cloud computing applications and services in medical, educational, financial, library and agricultural disciplines. In *Proceedings of the 4th International Conference on Frontiers of Educational Technologies*, pages 132–141, 2018. doi:10.1145/3233347.3233388.

[5] Wen Xia, Hong Jiang, Dan Feng, Fred Douglis, Philip Shilane, Yu Hua, Min Fu, Yucheng Zhang, and Yukun Zhou. A comprehensive study of the past, present, and future of data deduplication. *Proceedings of the IEEE*, 104(9):1681–1710, 2016. doi:10.1109/JPROC.2016.2571298.

[6] Taek-Young Youn, Ku-Young Chang, Kyung-Hyune Rhee, and Sang Uk Shin. Efficient client-side deduplication of encrypted data with public auditing in cloud storage. *IEEE Access*, 6:26578–26587, 2018. doi:10.1109/ACCESS.2018.2836328.

[7] Shunrong Jiang, Tao Jiang, and Liangmin Wang. Secure and efficient cloud data deduplication with ownership management. *IEEE Transactions on Services Computing*, 2017. doi:10.1109/TSC.2017.2771280.

[8] Won-Bin Kim and Im-Yeong Lee. Survey on data deduplication in cloud storage environments. *Journal of Information Processing Systems*, 17(3): 658–673, 2021. doi:https://doi.org/10.3745/JIPS.03.0160.

[9] Roberto Di Pietro and Alessandro Sorniotti. Boosting efficiency and security in proof of ownership for deduplication. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 81–82, 2012. doi:https://doi.org/10.1145/2414456.2414504.

[10] Weijing You, Lei Lei, Bo Chen, and Limin Liu. What if keys are leaked? towards practical and secure re-encryption in deduplication-based cloud storage. *Information*, 12(4):142, 2021. doi:https://doi.org/10.3390/info12040142.

[11] Dutch T Meyer and William J Bolosky. A study of practical deduplication. *ACM Transactions on Storage (ToS)*, 7(4):1–20, 2012. doi:https://doi.org/10.1145/2078861.2078864.

[12] Jian Liu, Nadarajah Asokan, and Benny Pinkas. Secure deduplication of encrypted data without additional independent servers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 874–885, 2015. doi:10.1145/2810103.2813623.

[13] Youngjoo Shin, Dongyoung Koo, and Junbeom Hur. A survey of secure data deduplication schemes for cloud storage systems. *ACM computing surveys (CSUR)*, 49(4):1–38, 2017. doi:10.1145/3017428.

[14] Lorena González-Manzano, Jose Maria de Fuentes, and Kim-Kwang Raymond Choo. ase-pow: A proof of ownership mechanism for cloud deduplication in hierarchical environments. pages 412–428, 2016. doi:10.1007/978-3-319-59608-2_24.

[15] Taek-Young Youn, Nam-Su Jho, Keonwoo Kim, Ku-Young Chang, and Ki-Woong Park. Locked deduplication of encrypted data to counter identification attacks in cloud storage platforms. *Energies*, 13(11):2742, 2020. doi:https://doi.org/10.3390/en13112742.

[16] Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Proofs of ownership in remote storage systems. pages 491–500, 2011. doi:https://doi.org/10.1145/2046707.2046765.

[17] Jay Dave, Avijit Dutta, Parvez Faruki, Vijay Laxmi, and Manoj Singh Gaur. Secure proof of ownership using merkle tree for deduplicated storage. *Automatic Control and Computer Sciences*, 54(4):358–370, 2020. doi:https://doi.org/10.3103/S0146411620040033.

[18] Tariqul Islam, Hassan Mistareehi, and D Manivannan. Secres: A secure and reliable storage scheme for cloud with client-side data deduplication. pages 1–6, 2019. doi:10.1109/GLOBECOM38437.2019.9013469.

[19] Shivansh Mishra, Surjit Singh, and Syed Taqi Ali. Mpows: Merged proof of ownership and storage for block level deduplication in cloud storage. In *2018 9th international conference on computing, communication and networking technologies (ICCCNT)*, pages 1–7. IEEE, 2018. doi:10.1109/ICCCNT.2018.8493976.

[20] Yongkai Fan, Xiaodong Lin, Wei Liang, Gang Tan, and Priyadarsi Nanda. A secure privacy preserving deduplication scheme for cloud computing. *Future Generation Computer Systems*, 101:127–135, 2019. doi:https://doi.org/10.1016/j.future.2019.04.046.

[21] Osama Ouda. A secure proof of ownership scheme for efficient client-side deduplication in cloud. *Journal of Convergence Information Technology*, 11:82–92, 2016. URL https://bit.ly/3sjSkxj.

[22] Ruiying Du, Lan Deng, Jing Chen, Kun He, and Minghui Zheng. Proofs of ownership and retrievability in cloud storage. pages 328–335, 2014. doi:10.1109/TrustCom.2014.44.

[23] Di Zhang, Junqing Le, Nankun Mu, Jiahui Wu, and Xiaofeng Liao. Secure and efficient data deduplication in jointcloud storage. *IEEE Transactions on Cloud Computing*, 2021. doi:10.1109/TCC.2021.3081702.

[24] VS Lakshmi, S Deepthi, and PP Deepthi. Collusion resistant secret sharing scheme for secure data storage and processing over cloud. *Journal of Information Security and Applications*, 60:102869, 2021. doi:10.1016/j.jisa.2021.102869.

[25] Shanshan Li, Chunxiang Xu, and Yuan Zhang. Csed: Client-side encrypted deduplication scheme based on proofs of ownership for cloud storage. *Journal of Information Security and Applications*, 46:250–258, 2019. doi:http://dx.doi.org/10.1016/j.jisa.2019.03.015.

[26] Jinbo Xiong, Fenghua Li, Jianfeng Ma, Ximeng Liu, Zhiqiang Yao, and Patrick S Chen. A full lifecycle privacy protection scheme for sensitive data in cloud computing. *Peer-to-peer Networking and Applications*, 8 (6):1025–1037, 2015. doi:10.1007/s12083-014-0295-x.

[27] Kangle Wang, Xiaolei Dong, Jiachen Shen, and Zhenfu Cao. An effective verifiable symmetric searchable encryption scheme in cloud computing. In *Proceedings of the 2019 7th International Conference on Information Technology: IoT and Smart City*, pages 98–102, 2019. doi:10.1145/3377170.3377251.

[28] Hui Cui, Robert H Deng, Yingjiu Li, and Guowei Wu. Attribute-based storage supporting secure deduplication of encrypted data in cloud. *IEEE Transactions on Big Data*, 5(3):330–342, 2017. doi:10.1109/TBDATA.2017.2656120.

[29] Hua Ma, Ying Xie, Jianfeng Wang, Guohua Tian, and Zhenhua Liu. Revocable attribute-based encryption scheme with efficient deduplication for ehealth systems. *IEEE Access*, 7:89205–89217, 2019. doi:10.1109/ACCESS.2019.2926627.

[30] Jorge Blasco, Roberto Pietro, Agustin Orfila, and Alessandro Sorniotti. A tunable proof of ownership scheme for deduplication using bloom filters. *2014 IEEE Conference on Communications and Network Security, CNS 2014*, pages 481–489, 12 2014. doi:10.1109/CNS.2014.6997518.

[31] Lorena González-Manzano and Agustín Orfila. An efficient confidentiality-preserving proof of ownership for deduplication. *J. Netw. Comput. Appl.*, 50:49–59, 2015. doi:10.1016/j.jnca.2014.12.004.

[32] Ke Huang, Xiao-song Zhang, Yi Mu, Fatemeh Rezaeibagha, and Xiaojiang Du. Bidirectional and malleable proof-of-ownership for large file in cloud storage. *IEEE Transactions on Cloud Computing*, 2021. doi:10.1109/TCC.2021.3054751.

[33] Meixia Miao, Guohua Tian, and Willy Susilo. New proofs of ownership for efficient data deduplication in the adversarial conspiracy model. *International Journal of Intelligent Systems*, 36(6):2753–2766, 2021. doi:10.1002/int.22400.

[34] Angtai Li, Guohua Tian, Meixia Miao, and Jianpeng Gong. Blockchain-based cross-user data shared auditing. *Connection Science*, pages 1–21, 2021. doi:10.1080/09540091.2021.1956879.

[35] John Viega, Matt Messier, and Pravir Chandra. *Network security with OpenSSL: cryptography for secure communications*. " O'Reilly Media, Inc.", 2002. URL https://dl.acm.org/doi/10.5555/2167247.