ZKG INTERNATIONAL ISSN: 2366-1313

BLOCKCHAIN TECHNOLOGY WITH OWNERSHIP SHARING IN THE CLOUD

[1] PULIMADDI NAGENDRA, [2] A. CHENNAKESAVA REDDY

[1]PG Scholar, Dept. of MCA, Newton's Institute of Engineering, Macherla Guntur, (A.P)

[2]Assistant Professor, Dept. of CSE, Newton's Institute of Engineering, Macherla Guntur, (A.P)

Abstract: *While cloud storage solutions claim to make it easy for users to collaborate and share data, they insist that each file have a single owner who has complete control over who has access to it. Thus, existing clouds do not care about the concept of shared ownership. This may be a serious drawback in many collaborative situations, since one owner may unilaterally remove data or withdraw access without informing the others. In this study, we begin by explicitly defining a concept of shared ownership within the context of a paradigm for controlling access to files. Our suggested shared ownership concept is then given in two different conceivable forms. To guarantee that all access permits in the cloud have the support of an agreed upon threshold of owners, our initial solution, Commune, relies on secure file distribution and collusion resistant secret sharing. Because of this, Commune may be deployed in pre-existing clouds without requiring any infrastructure upgrades. To achieve agreement on an access control decision, we use blockchain technology in our second option, which we've named Comrade. In contrast to Commune, Comrade needs just small adjustments to current clouds in order to transfer access control choices that attain agreement in the blockchain into storage access control rules. We use Amazon Simple Storage Service to deploy our solutions and analyse their security and performance.*

Keywords: *Cloud security; Shared ownership; Distributed enforcement; Blockchain technology.*

I. INTRODUCTION

In spite of the cloud's claims to simplify file sharing and collaboration, users will still maintain control of their own data. Each cloud-based file, in other words, belongs to a single user, who has complete discretion over whether or not to provide access to any other users. However, many cloud-based apps and partnerships don't work well with individual ownership. In order to work together on a research project, many institutions and companies can decide to create a central cloud repository.

If everyone involved in the project contributes to the study, it makes sense for

ISSN: 2366-1313

everyone to have some say in who has access to the collaborative files. There are two primary justifications for considering this an improvement over private property. To start, a single proprietor might misuse his authority by deciding on security measures on his own. Several stories of someone denying others access to previously shared files may be found in the community. Second, even if owners are prepared to elect and trust one of them to make access control choices, that owner may not be interested in being responsible for collecting and accurately analysing the policies of the other owners. Incorrect assessments, for instance, might result in social or financial fallout.

In contrast to traditional notions of ownership, we offer the concept of shared ownership, in which a group of n users collectively own a file and any request to access the file must be approved by a minimum agreed-upon number of owners, t. We point out that currently available cloud solutions like Amazon S3 or Dropbox only provide very limited access control lists and do not enable shared ownership restrictions. In a nutshell, they don't care much for the idea of community property. In addition, modern trust management systems (such Sec PAL [1], Key Note [2], and Delegation Logic [3]) that enable shared ownership rules handle all access choices at a single Policy Decision Point (PDP).

Since the person who administers the PDP may alter the policy rules provided by the owners and apply his own policies, this is not ideal for implementing our shared ownership model.

In this work, we investigate the issue of how to implement shared ownership in a decentralised manner across different cloud storage services. We refer to this method of enforcing rules wherein t of n owners independently agrees to allow access to a set of files in a shared repository as "distributed enforcement." To formally describe the provided enforcement challenge and establish our idea of shared ownership, we develop the Shared Ownership file access control Model (SOM). Then, to implement distributed shared ownership regulations, we suggest two different SOM model instantiations.

Our prior work [4] is built upon in this study. Specifically, we provide more formally oriented information on the SOM model. We also suggest a novel implementation of the SOM model called Comrade, which uses blockchain technology to facilitate agreement on security-related matters. Comrade, in contrast to the Commune architecture

ISSN: 2366-1313

suggested in [4], necessitates the participation of the cloud provider, who is tasked with translating access control choices that obtained agreement on the blockchain into storage access control rules. Comrade, on the other hand, outperforms Commune by a wide margin. We implement a smart contract instantiating Comrade on the Ethereum blockchain, link it to Amazon's cloud storage [5], and evaluate its efficacy in relation to file size and user count in comparison to Commune [4]. Here's a quick rundown of what we've contributed:

• We establish a new access control issue of distributed enforcement of shared ownership in existing clouds, and we formalise the idea of shared ownership inside a file access control model called SOM.

• We suggest Commune as a first approach, which enforces SOM in a distributed manner and can be used with any public cloud service. When using Commune, you can be certain that (i) only those who have been allowed read access by at least t of the owners will be able to read files from the shared repository, and (ii) only those who have been granted write access by at least t of the owners will be able to write files to the shared repository.

• We suggest a second approach, which we call Comrade, that uses the capabilities of blockchain technology to arrive to an agreement on access control decisions. If you want to boost Commune's performance, you'll need to make some modest adjustments to your cloud so that it can transform access control choices that obtained consensus in the blockchain into storage access control rules.

• We create working models of Commune and Comrade and assess how they fare on Amazon S3 with increasing file sizes and user loads.

Here is how the rest of the paper is structured. In Section II, we explain our methodology for regulating access to files when several people have a stake in them. In Section III, we break down Commune and examine its safety in depth. Section IV provides an overview of Comrade and an examination of its features.

In Section V, the effectiveness of Commune and Comrade in Amazon S3 is assessed. Additional observations on the Commune and the Comrade are discussed in Section VI.

After a review of relevant work in Section VII, we draw a conclusion in Section VIII.

II. RELATED WORKS

ISSN: 2366-1313

In theory, the top-of-the-line access control systems available today, such as Sec PAL [1], Key Note [2], and Delegation Logic [3], may express t out of n rules. However, these languages need a centralised PDP component in order to assess their policies. Furthermore, their PDPs are not compatible with any external cloud service deployment. These access control systems need an administrator to develop and administer access control policies, as described in Section II. This implies that under our system, a group of owners must choose a single enforcer with absolute control over their data.

The issue of joint ownership may be solved, and access to a shared resource can be managed by numerous parties, with the help of Multi-Authority Attribute-Based Encryption (MA-ABE) [18], [29]. While MA-ABE has been proposed before, most of the current implementations depend on innovative cryptographic assumptions and require costly operations because of the need of a bilinear map.

Instead, we use CRSS in this study. CRSS depends on standard assumptions and can only support threshold policies since only a cyclic group of prime order is sufficient. We contend that threshold rules are adequate for the purpose at hand, and that our system may thus profit from the minimal complexity of CRSS. In addition, exactly like CRSS, MA-ABE can only be used to control who has access to an encryption key. Combining CRSS with SFD is one alternative method for regulating access to huge files.

Distributing a secret among a group of shareholders through a secret sharing method [30] ensures that only authorised shareholders may piece together the secret. Each set of shareholders with a cardinality equal to or larger than the dealer-defined threshold t is permitted to reconstruct the secret in threshold secret sharing systems [20], [31]. Secure secret sharing (i.e., the secret cannot be retrieved by an unauthorised group of shareholders) is impracticable for exchanging huge data due to its high computing and storage requirements.

While Rabin's [11] concept for disseminating information has a lower overhead than [31], it offers no security assurances in the event that just a few shares are available (less than the threshold). Krawczyk [32] is a hybrid of Shamir's [31] and Rabin's [11] methods; in [32], a file is encrypted using AES before being distributed using Rabin's [11] technique, and the encryption key is shared using Shamir's [31] method.

Erasure-code based methods of information distribution [16] are powerful methods to improve the security of cloud-

ISSN: 2366-1313

based data storage [33], [36]. Ramp systems [37] are a compromise between the confidentiality assurances of secret-sharing protocols and the speed with which information may be dispersed.

Changes That Can Only Be Made One Way: After being presented in [13], all-or-nothing transformations have been studied in [14], [38]. Most AONTs make use of an encrypted key stored inside the data blocks that are ultimately sent. The key may be retrieved, allowing for the reversal of individual blocks after all output data has been generated. A Fast Fourier Transform-like transformation was previously mentioned by Rivest [13].

A "proof of encryption" for cloud-stored data was subsequently built using Van Dijk et al.'s [17] application of Rivest's transformation. In this study, we develop an AONT scheme by expanding on Rivest's transformation to ensure that the scheme retains its all-or-nothing nature even if the opponent has access to the secret key. When applied to the setting of distributed storage systems, the combination of AONT and information dispersion proposed by Resch et al. [12] provides both fault-tolerance (i.e., decoding needs only t out of n shares) and data secrecy (i.e., confidentiality is ensured with respect to parties that gather less than t shares). A malicious actor that has cached the encryption key may still decrypt individual shares in [12]. By encrypting the data beforehand and then post-processing it with a linear transform, Karama et al. [39] demonstrated that a secure encryption mode with the same guarantees as all-or-nothing transformations is possible to build.

III PROTOTYPE DESIGN & EVALUATION

In this section, we describe prototype implementation of Commune and Comrade integrated with Amazon S3 [5] and

	New Owner Contract	New Permission	New File
4 Owners	3.05	0.08	0.00
8 Owners	5.47	0.08	0.00

TABLE I

TRANSACTION FEES IN USD FOR OUR COMRADE PROTOTYPE. evaluate their performance.

A. Collective Deployment

To instantiate S, we make use of Amazon S3; specifically, we set up individual Amazon S3 accounts for each user in U, into which they may transfer data and set permissions at their discretion. Tokens are sent from the file's author to the group of owners OU using the access control capabilities of Amazon S3. For simplicity's sake, let's suppose that (i) each user creates one "temporary" folder to which all other peers have write access and (ii) each user

ISSN: 2366-1313

creates one "main" folder to which endorsed tokens may be placed and retrieved. The token is written to Oj's temporary folder when the file's author wishes to share it with the owner. Oj is the only user with read access to the temporary folder, therefore the newly generated token is safe there. Now Oj may approve the token for Ul by putting it in his primary folder and giving Ul read access to it.

Our Commune prototype provides a multithreaded Java client-side interface to Amazon S3-hosted repositories. The user's local copy of the client communicates with the repositories to add and remove files. Rijndael [15] is used as the block cypher for AON-FFT in the client's implementation of SFD, and systematic Reed-Solomon codes [16] are used to disseminate data. We decided on a security parameter of =128 bits and a symbol size of 16 bytes.

Our prototype processes file unit actions in smaller chunks, or "pieces," to improve speed. Each new file unit is broken up into smaller bits that may be handled in parallel. One SFD output chunk per unit's component parts is included in the token for that unit. The reconstruction threshold t is set in relation to the security parameter, and the piece size w is selected such that tow. To put it another way, this condition guarantees that (i) a piece can be encrypted in an integer number of ciphertext blocks of bits, (ii) an encrypted piece can be divided into an integer number of input chunks for the Reed-Solomon encoder, and (iii) the size of each chunk of the Reed-Solomon encoder/decoder is at least bits.

B. Communist Party of China

In our implementation of Comrade, a solidity-based smart contract5 on the Ethereum blockchain communicates with a python-based client using a shared Amazon S3 bucket that is held by the owner contract. We use an Amazon EC2 instance to host the PDP since Amazon does not currently support blockchain-aware PDPs. All decisions about S3 account access are made by the PDP in light of the current blockchain state. All file-related inquiries from our customers are handled by the PDP.

Our smart contracts are deployed on the public Ethereum blockchain. The basic rationale of the Comrade works is reflected in the owner contract that comes next. Users cast their votes on the security settings using Ethereum transactions. Keep in mind that we save transaction costs by doing experiments on a private Ethereum [24] chain. The costs are shown in Table I below. For 4 owners, the price of creating

ISSN: 2366-1313

our Ethereum owner contract is $3.056, and for 8 owners, it is $5.47. It costs $0.08 to provide rights, however there is no charge for adding a new file.

Each user adds their public key to the PDP's owner contract in order to facilitate authentication with the PDP. We use compact elliptic curve cryptography (ECC) due to the high cost of storing data inside the blockchain (as ECC public keys are more compact than RSA keys). In order to get access to a file, a client must first register its public key in a client certificate and then establish a TLS connection to the PDP. The PDP verifies the client's identity with the help of the key and then makes the choice about access based on a local evaluation of a function of the owner contract. Our Comrade prototype disassembles units in the same way as Commune does.

In our system, Amazon S3 is used to store both the encrypted, shareable files and the wrapped keys. It's important to note that Comrade doesn't need any extra redundancy for stored data, in contrast to Commune. Therefore, the only extra space that Commune has to keep are the wrapped keys. The storage overhead for a file is equal to 36 bytes times the number of owners, or 36 bytes total. Most commonly used files are at least 1 MB in size; therefore, this is a very small storage hit.

C. Unit-of-Analysis Evaluation Write/Read

For a single file unit read and write, we compare the efficiency of Commune and Comrade across a range of parameters, including (i) the piece size w (default value w=128 bytes), (ii) the reconstruction threshold t(default value t = 4), (iii) the number of owners n (default value n = 10), and (iv) the size of the file unit |Fi | (default value |Fi |=10 MiB).

The performance of the system is monitored while we gradually alter one of the variables. We take the time to (i) generate and upload Fi (labelled Write in our plots) and (ii) get Fi (labelled Read) for each setup. These times are calculated from the moment an operation is initiated until the result is accessible in the archives (in the case of Write) or on the local disc (in the case of Read). By transferring randomly generated binary streams at each iteration, we neutralise the cache's impact.

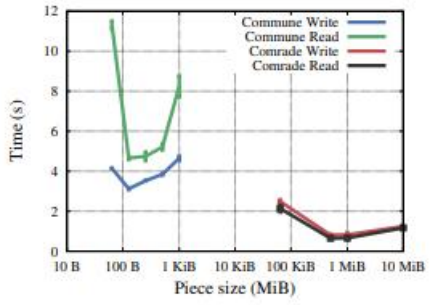
The Commune client retrieves endorsed tokens from t randomly selected owners during Read. A (t,n) systematic erasure code, in case you forgot, produces t data chunks and nt parity chunks. Our assessment takes into consideration the most likely situation, in which the chance that a token includes a data chunk is

ISSN: 2366-1313

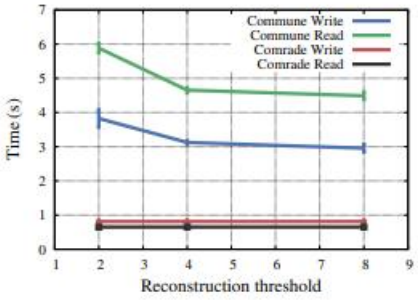
constrained by t n, such that no decoding is required. However, the time needed to endorse a token or submit a blockchain transaction (i.e., the time necessary to provide read rights) is not evaluated since it is independent of the criteria under consideration.

Figure 5 shows the outcomes of our research. Figure 5(e) depicts how we track the execution duration of Commune's intermediate phases across many setups.

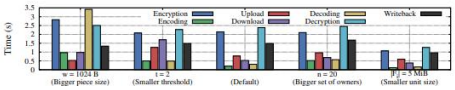
According to our findings, in Commune it is cheaper to write a new unit than to read it, but in Comrade the roles are inverted. Write performance in Comrade is impacted by the time spent uploading wrapped keys for all owners, whereas the former effect is caused by the cost of thread synchronisation while storing decoded bits on the local disc.



(a) Impact of the piece size.



(b) Impact of the reconstruction threshold.



	Peak Throughput (Mbps)	
	Commune	Comrade
Write	43.39	190.26
Read	29.52	225.37

TABLE II
PEAK THROUGHPUT. EACH DATA POINT IS THE AVERAGE OF 20 MEASUREMENTS.

IV. DISCUSSION

Additional details on the architecture of Commune and Comrade, as well as their potential growth, are discussed below.

User-Friendly Transparency: As was previously mentioned, Commune allows its customers to centrally manage who has access to what in the cloud.

As mentioned in Section V, the client application is responsible for implementing all Commune activities. It is not necessary for users to "manually" disperse or retrieve tokens. In reality, users need simply declare the access policy for the files they are appointed owners of and determine the list of owners for the files they generate.

ISSN: 2366-1313

In Comrade, the owner contract is made by the owners themselves at the outset.

After then, the owner contract coordinates the activities of all participants.

The encrypted text and its matching wrapped key are transparently retrieved by Comrade.

By adjusting t, we mean: For the sake of uniformity in Commune, we do not permit the modification of threshold t for each given file F. Owners in O would need to have fresh tokens calculated and distributed if the threshold were to be changed, say from t to t 0. After then, all O token holders must swap over their old tokens for the new ones. Since each holder has complete control over their tokens, there is no way to have them all update at once. Some tokens may correspond to a file version with threshold t, while other tokens may correspond to a file version with threshold t 0. This might lead to an inconsistent state. As a result, Commune does not support raising the bar. Comrade, on the other hand, agrees that threshold t may be changed by adjusting the owner contract. The prerequisites for such a modification, such as the consent of all owners, are spelt out in the owner contract. After the necessary steps have been taken to implement a modification, the new standard will be used in all subsequent cloud PDP assessments of the owner contract.

V Conclusion

Although current cloud systems are used as communal storage areas, no such concept of joint ownership is supported.

We see this as a major restriction since the parties that are providing resources cannot agree on how those funds should be allocated.

In this work, we present SOM, a formal access control model that describes a new idea of shared ownership. Our suggested shared ownership concept is then given in two different conceivable forms. To guarantee that all access permits in the cloud have the backing of an agreed upon threshold of owners, our initial solution, Commune, relies on secure file distribution and collusion-resistant secret sharing. Since there is no need to alter the underlying infrastructure, Commune may be deployed in already agnostic clouds.

Our second approach, named Comrade, uses blockchain technology to achieve agreement when deciding who gets access.

Comrade differs from Commune in that it necessitates the cloud to transform consensus-based blockchain-based access control choices into storage access control

ISSN: 2366-1313

rules. However, compared to Commune, Comrade proves to be more effective.

We propose that, with the proliferation of personal clouds (e.g., [9], [10]), Commune and Comrade are particularly well-suited to the task of establishing distributed-management repositories atop users' individual clouds. As a result, we anticipate that our results will inspire further studies in this field.

REFERENCES

[1] M. Y. Becker, C. Fournet, and A. D. Gordon, "Sec PAL: Design and Semantics of a Decentralized Authorization Language," in Journal of Computer Security (JCS), 2010, pp. 597–643.

[2] M. Blaze, J. Ioannidis, and A. D. Keromytis, "Trust Management for IPsec," in ACM Transactions on Information and System Security (TISSEC), 2002.

[3] N. Li, B. N. Grosof, and J. Feigenbaum, "Delegation logic: A Logic-based Approach to Distributed Authorization," in TISSEC, 2003.

[4] C. Soriente, G. O. Karame, H. Ritzdorf, S. Marinovic, and S. Capkun, "Commune: Shared ownership in an agnostic cloud," ser. SACMAT '15, 2015.

[5] "Amazon Simple Storage Service(S3)," http://aws.amazon.com/s3/.

[6] S. Ceri, G. Gottlob, and L. Tanca, "What you always wanted to know about Datalog (and never dared to ask)," in Knowledge and Data Engineering, IEEE Transactions on, 1989.

[7] Y. Gurevich and I. Neeman, "DKAL: Distributed-Knowledge Authorization Language ," in CSF '08.

[8] J. DeTreville, "Binder, a Logic-based Security Language," in Proceedings of IEEE Symposium on Security and Privacy, 2002, pp. 105 – 113.

[9] "The Respect Network," https://www.respectnetwork.com/.

[10] "WD My Cloud," http://www.wdc.com/en/products/products.aspx?id=1140.

[11] Prasadu Peddi (2017) "Design of Simulators for Job Group Resource Allocation Scheduling In Grid and Cloud Computing Environments", ISSN: 2319-8753 volume 6 issue 8 pp: 17805-17811.