A Reliable Data Sharing Ownership Using Blowfish Algorithm in Cloud

P.Divyabarathi[1] R.Priya[2] B.M.Nifla Begum[3] M.Gayathri[4]
[1]Assistant Professor
[1,2,3,4]Department of Computer Science & Engineering
[1,2,3,4]The Kavery Engineering College, Salem, India

Abstract— Cloud storage platforms promise a convenient way for users to share files and engage in collaborations, yet they require all files to have a single owner who unilaterally makes access control decisions. Existing clouds are, thus, agnostic to the notion of shared ownership. This can be significant limitation in much collaboration because, for example, one owner can delete files and revoke access without consulting the other collaborators. In this project, a Blowfish algorithm is proposed which is a symmetric encryption algorithm, meaning that it uses the same secret key to both encrypt and decrypt messages. Proposed Blowfish is also a block cipher, meaning that it divides a message up into fixed length blocks during encryption and decryption. Here, first formally define a notion of shared ownership within a file access control model. Then propose two possible instantiations of our proposed shared ownership model. Unlike Commune, Comrade requires that the cloud is able to translate access control decisions that reach consensus in the block chain into storage access control rules, thus requiring minor modifications to existing clouds.

Keywords: Cloud, Blowfish, Sharing Ownership, Block Chain

I. INTRODUCTION

Cloud computing is a wide-ranging term that transmits hosted services over the internet and recognized as an alternative to traditional information technology due to its intrinsic resource-sharing and low-maintenance characteristics. One of the most fundamental services offered by cloud providers is data storage. With cloud computing and storage, users are able to access and to share resources offered by cloud service providers at a lower marginal cost. With dropbox, for example, data is stored in the cloud (operated by amazon), and shared among a group of users in a collaborative manner. Let us consider a real time scenario. A professor allows his students and colleagues to share files stored in cloud. Those files may include internal assessment of students, his personal information, student's personal information, research details, other staff details etc. By utilizing cloud, students and other employees can get relevant information on a request basis. However, it poses some confidentiality risks. The cloud service provider or third party may not fully trusted by the users [2]. A possible approach would be to encrypt entire data files before outsourcing in order to achieve more integrity. Unfortunately, designing an efficient and secure data sharing scheme for groups in the cloud is not an easy task due to the following challenging issues.

First, identity privacy is one of the most significant obstacles for the wide deployment of cloud computing. Without the guarantee of identity privacy, users may be unwilling to join in cloud computing systems because their real identities could be easily disclosed to cloud providers and attackers. Second, it is highly recommended that any member in a group should be able to fully enjoy the data storing and sharing services provided by the cloud, which is defined as the multiple-owner manner. Compared with the single-owner manner [3], the multiple-owner manner is more flexible in practical applications. On the other hand, an efficient ownership sharing based on blowfish algorithm without updating of the secret keys of the remaining users minimize the complexity of key management , signed receipt is collected after every member revocation in the group it minimizes the multiple copies of encrypted file and also reduces computation cost.

A. Related Works and Shared Ownership Using Blockchain

In the existing method, it address the problem of distributed enforcement of shared ownership within cloud storage providers. By distributed enforcement, this article mean enforcement where access to files in a shared repository is granted if and only if t out of n owners separately support the grant decision. Therefore, here introduced the Shared-Ownership file access controlModel (SOM) to define our notion of shared ownership, and to formally state the given enforcement problem. We then propose two instantiations of the SOM model to enforce shared ownership policies in a distributed fashion. This paper extends our previous work. More specifically, we provide additional formal details about the SOMmodel. We also propose a new instantiation of the SOM model, Comrade that leverages functionality from the blockchain in order to reach consensus on access control decisions. Unlike the Commune framework proposed, Comrade requires cooperation from the cloud provider that is expected to translate access control decisions that reached consensus in the blockchain into storage access control rules. Comrade, however, exhibits considerably better performance than Commune. We deploy a smart contract instantiating Comrade within the blowfish, connect it to Amazon cloud storage, and compare its performance to the one of Commune with respect to the file size and the number of users.

Even though existing cloud platforms are used as collaborative platforms, they surprisingly do not support any notion of shared ownership. We consider this to be a severe limitation because collaborators cannot jointly decide how their resources are used. The problem of enforcing shared ownership in the cloud is even more difficult since a cloud platform does not allow deployment of a third-party enforcement component. In this paper, we introduced a novel concept of shared ownership and we described it through a formal access control model, called SOM. We also proposed our scheme, Commune that distributively enforces SOM. Commune can be used in existing clouds without requiring any modifications to the platforms. This paper implemented and evaluated the performance of our solution within Amazon S3.

B. Comrade: Blockchain-Based Shared Ownership

In this section, we present an alternative solution for enforcing shared ownership in the cloud by leveraging functionality from the blockchain. The solution, dubbed Comrade, enables a distributed blockchain-based enforcement of the SOM access control policy in a cooperative cloud. Unlike Commune, Comrade does not assume an agnostic cloud, and requires the cloud operator to cooperate and to interface with the blockchain. Since SOM does not specify concrete file access operations, we instantiate Comrade with write and read actions. Before introducing our solution, we provide some background on the Blockchain and describe the system model. A. Blockchain and Smart Contracts The notion of Blockchain was originally introduced by the well known proof-of-work hash-based mechanism that confirms crypto currency payments in Bitcoin. The PoW-based Blockchain ensures that all transactions and their order of execution are available to all blockchain nodes, can be verified by all involved entities andaids the consensus between the parties. Bitcoin'sblockchain fueled innovation, and a number of innovative applications have already been devised by exploiting the secure and distributed provisions of the underlying block chain. Prominent applications include secure time stamping, and smart contracts. Smart contracts refer to binding contracts between two or more parties that are executed by all blockchain nodes. Namely, smart contracts implement state machine replication. Smart contracts typically consist of a self-contained code and persistent storage available to all blockchain nodes. For example, Ethereumisa decentralized platform that enables the execution of arbitrary applications (or contracts) on its blockchain. Owing to its support for a Turing-complete language, Ethereum (which currently also relies on PoW-based consensus) offers an easy means for developers to deploy their distributed applications in the form of smart contracts. To make smart contracts more powerful, techniques have been developed to securely insert real-world facts into blockchains, such as Town Crier. These facts, such as weather information or flight delays, allow contracts to take real-world events into account and to offer new functionalities.

II. PROPOSED TECHNIQUES

Here, formalize the notion of shared ownership within a file access control model named SOM, and use it to define a novel access control problem of distributed enforcement of shared ownership in existing clouds. To propose a first solution, called Commune, which distributive enforces SOM and can be deployed in an agnostic cloud platform. Commune ensures that (i) a user cannot read a file from a shared repository unless that user is granted read access by at least t of the owners, and (ii) a user cannot write a file to a shared repository unless that user is granted write access by at least t of the owners. To propose a second solution, dubbed Comrade, this leverages functionality from the blowfish in order to reach consensus on access control decision. Comrade improves the performance of Commune, but requires that the cloud is able to translate access control decisions that reached consensus in the blowfish into storage access control rules, thus requiring minor modifications of existing clouds.

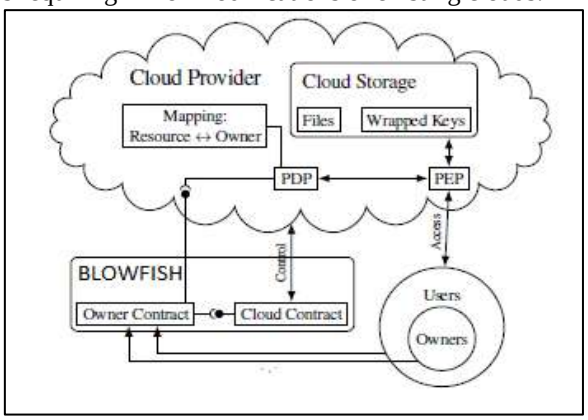

Fig. 1: Proposed block diagram

A. Ownership

This risk comes as a surprise to many cloud customers, but often the customer is not the only owner of the data. Many public cloud providers, including the largest and best known, have clauses in their contracts that explicitly states that the data stored is the provider's -- not the customer's. A cloud vendor like owning the data because it gives them more legal protection if something goes wrong. Plus, they can search and mine customer data to create additional revenue opportunities for themselves. I've even read of a few cases where a cloud vendor went out of business, then sold their customers' private data as part of their assets to the next buyer. It's shocking. Make sure you have this known unknown on lockdown: Who owns your data and what can the cloud provider do with it?

B. Blowfish Algorithm

Blowfish has a 64-bit block size and a variable key length from 32 bits up to 448 bits. It is a 16-round Feistel cipher and uses large key-dependent S-boxes. In structure it resembles CAST-128, which uses fixed S-boxes. The diagram to the left shows Blowfish's encryption routine. Each line represents 32 bits. There are five subkey-arrays: one 18-entry P-array (denoted as K in the diagram, to avoid confusion with the Plaintext) and four 256-entry S-boxes (S0, S1, S2 and S3).

Action 1	XOR the left half (L) of the data with the r th P-array entry
Action 2	Use the XORed data as input for Blowfish's F-function
Action 3	XOR the F-function's output with the right half (R) of the data
Action 4	Swap L and R

The F-function splits the 32-bit input into four eight-bit quarters, and uses the quarters as input to the S-boxes. The S-boxes accept 8-bit input and produce 32-bit output. The outputs are added modulo 2^{32} and XORed to produce the final 32-bit output (see image in the upper right corner). After the 16th round, undo the last swap, and XOR L with K18 and R with K17 (output whitening)..Decryption is exactly the same as encryption, except that P1, P2, …,P18 are used in the reverse order. This is not so obvious because xor is commutative and associative. A common misconception is to

use inverse order of encryption as decryption algorithm (i.e. first XORingP17 and P18 to the ciphertext block, then using the P-entries in reverse order).

Blowfish's key schedule starts by initializing the P-array and S-boxes with values derived from the hexadecimal digits of pi, which contain no obvious pattern (see nothing up my sleeve number). The secret key is then, byte by byte, cycling the key if necessary, XORed with all the P-entries in order. A 64-bit all-zero block is then encrypted with the algorithm as it stands. The resultant ciphertext replaces P_1 and P_2. The same ciphertext is then encrypted again with the new subkeys, and the new ciphertext replaces P_3 and P_4. This continues, replacing the entire P-array and all the S-box entries. In all, the Blowfish encryption algorithm will run 521 times to generate all the subkeys - about 4KB of data is processed.

Because the P-array is 576 bits long, and the key bytes are XORed through all these 576 bits during the initialization, many implementations support key sizes up to 576 bits. While this is certainly possible, the 448 bits limit is here to ensure that every bit of every subkey depends on every bit of the key, as the last four values of the P-array don't affect every bit of the ciphertext. This point should be taken in consideration for implementations with a different number of rounds, as even though it increases security against an exhaustive attack, it weakens the security guaranteed by the algorithm. And given the slow initialization of the cipher with each change of key, it is granted a natural protection against brute-force attacks, which doesn't really justify key sizes longer than 448 bits.

C. Collusion Resistant Secret Sharing (CRSS)

Introduce our second building block, called Collusion Resistant Secret Sharing (CRSS). Similar to threshold secret-sharing schemes, CRSS allows one party to distribute a secret among a set of designated shareholders, so that any subset of shareholders of size equal to or greater than the threshold can reconstruct the secret. Furthermore, CRSS allows shareholders to issue to other users delegation to reconstruct the secret. If a user collects enough (i.e., above the threshold) delegations, he can rightfully reconstruct the secret. However, users cannot pool their delegations to reconstruct the secret, unless one of them has collected enough delegations. In Commune, CRSS is used to secret-share the key K used in SFD, in order to achieve collusion resistance. CRSS is inspired by decentralized Attribute Based Encryption where shares of a secret are blinded with shares of 0, such that, if a user collects enough shares for his identity, the blinding cancels out and the secret can be reconstructed.

D. Cloud and Attacker Model

Focus on a cloud storage platform, S, where a set of users U have personal accounts onto which they upload files. For example, users might set up their own personal clouds or might create personal accounts in existing public clouds. We assume that S authenticates users before they get access to the platform. A user U 2U can unilaterally decide who access to files has stored on his account. In particular, S allows each user to define access control policies of the type p:U_fwrite;readg!fgrant;denyg.We also assume that S correctly enforces individual access control policies. This model reflects the functionalities provided by existing cloud platforms, such as Amazon S3. We focus on both external and internal adversaries. An adversary may try to gain read access to a file even if fewer than t owners have issued the corresponding credentials. We refer to this adversary as a "malicious reader". Alternatively, an adversary,who has been granted write access by fewer than t owners, may try to publish a file F as if Fwere authored by a userwho had been grantedwrite access by t ormoreowners.We refer to this adversary as a "malicious writer".We also consider sets of users who collude to escalate their access rights.

E. Commune & Comrade Access Control

Create a File. During file creation, one user the file creator writes the initial version of the file into the repository. This requires the file creator to have write permissions for the directory the file is created in. The file creator also encrypts the file using a randomly chosen file key before uploading it. The encrypted file is uploaded as F using a write action. To securely distribute the file key to Ui, the file creator also uploads wrapped keys Fk;Ui containing the file key for file F encrypted with the public key of user Ui. By default a file creator uploads wrapped file keys for all owners. Notice that the access control policy for Fk;Ui is defined such that a user Uj can access Fk;Ui if and only if Uj=Ui and Uj can access F.

F. Commune & Comrade Access Control

Create a File. During file creation, one user the file creator writes the initial version of the file into the repository. This requires the file creator to have write permissions for the directory the file is created in. The file creator also encrypts the file using a randomly chosen file key before uploading it. The encrypted file is uploaded as F using a write action. To securely distribute the file key to Ui, the file creator also uploads wrapped keys Fk;Ui containing the file key for file F encrypted with the public key of user Ui. By default a file creator uploads wrapped file keys for all owners. Notice that the access control policy for Fk;Ui is defined such that a user Uj can access Fk;Ui if and only if Uj=Ui and Uj can access F.

G. Key-expansion:

Blowfish uses a large number of subkeys. These keys must be precomputed before any data encryption or decryption.
The P-array consists of 18 32-bit subkeys:
P1, P2,...,P18.
There are four 32-bit S-boxes with 256 entries each:
S1,0, S1,1,..., S1,255;
S2,0, S2,1,..,, S2,255;
S3,0, S3,1,..., S3,255;
S4,0, S4,1,..,, S4,255.

H. Generating the Subkeys:

The subkeys are calculated using the Blowfish algorithm:

1) Initialize first the P-array and then the four S-boxes, in order, with a fixed string. This string consists of the hexadecimal digits of pi (less the initial 3): P1 = 0x243f6a88, P2 = 0x85a308d3, P3 = 0x13198a2e, P4 = 0x03707344, etc.
2) XORP1 with the first 32 bits of the key, XORP2 with the second 32-bits of the key, and so on for all bits of the key (possibly up to P14). Repeatedly cycle through the key

bits until the entire P-array has been XORed with key bits. (For every short key, there is at least one equivalent longer key; for example, if A is a 64-bit key, then AA, AAA, etc., are equivalent keys.)

3) Encrypt the all-zero string with the Blowfish algorithm, using the subkeys described in steps (1) and (2).
4) Replace P1 and P2 with the output of step (3).
5) Encrypt the output of step (3) using the Blowfish algorithm with the modified subkeys.
6) Replace P3 and P4 with the output of step (5).
7) Continue the process, replacing all entries of the P array, and then all four S-boxes in order, with the output of the continuously changing Blowfish algorithm.

In total, 521 iterations are required to generate all required subkeys. Applications can store the subkeys rather than execute this derivation process multiple times.

1) Data Encryption:

It is having a function to iterate 16 times of network. Each round consists of key-dependent permutation and a key and data-dependent substitution. All operations are

XORs and additions on 32-bit words. The only additional operations are four indexed array data lookup tables for each round.

Algorithm: Blowfish Encryption

Divide x into two 32-bit halves: xL, xR
For i = 1to 16:
xL = XL XOR Pi
xR = F(XL) XORxR
Swap XL and xR
Swap XL and xR (Undo the last swap.)
xR = xRXORP17
xL = xLXORP18
Recombine xL and xR

- Update a File. Assume Ul wants to write a new version of a file F. Ul encrypts F using its file key and issues a write action as described . In case the owner contract implements a threshold-based access control policy, the request succeeds if there are at least t owner votes in favour.
- Grant/Deny Read Rights. Analogously to write rights, an owner Oj grants or denies read rights for a resource F to an entity Ul by submitting a corresponding vote to the owner contract. As mentioned earlier, this vote corresponds to a blockchain transaction v(Oj, Ul, read, F).
- Read a File. Assume Ul wants to read a file F. Ul issues a read request for F and Fk;Ui as described in Section IV-B. In case the owner contract implements a threshold-based access control policy, the request succeeds if there are at least t owner votes in favour. Ul decrypts Fk;Ui using its private key to obtain the file key and finally decrypts F.

III. EXPERIMENTAL RESULTS

In this method, we are using cloudMe tool for the data storage which is shown in the below experimental results. A CloudMe is a powerful & stylish multipurpose Cloud Storage & File-Sharing Services WordPress theme. It has a modern business design created for online presentation of cloud storage and file-sharing services. It's a fresh theme that's also perfect for all sorts of applications, web hosting business or any kind of technology websites. Therefore, this technique is coded in the python language and pycharm software are used to execute the proposed techniques which is shown below.

Fig. 2: Homepage


Fig. 3: data owner registration


Fig. 4: user registration

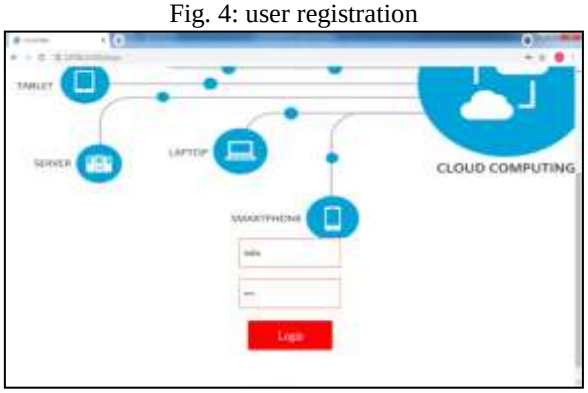

Fig. 5: owner login

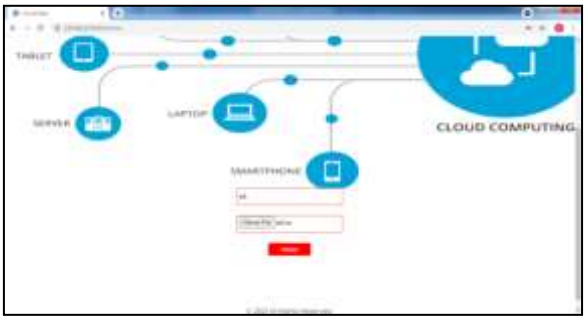

Fig. 6: file upload

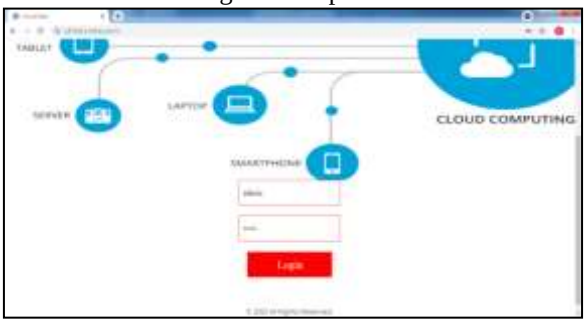

Fig. 7: admin login

Fig. 8: encryption process

Fig. 9: user requesting for file

Fig. 10: request view to owner

Fig. 11: user received response

Fig. 12: downloading with key

IV. CONCLUSION

In this paper, we introduced a novel concept of shared ownership and we described it through a formal access control model, called blowfish SOM. We then propose two possible instantiations of our proposed shared ownership model. Our first solution, called Commune, relies on secure file dispersal and collusion-resistant secret sharing to ensure that all access grants in the cloud require the support of an agreed threshold of owners. As such, Commune can be used in existing agnostic clouds without modifications to the platforms. Our second solution, dubbed Comrade, leverages the blowfish technology in order to reach consensus on access control decision. Unlike Commune, Comrade requires that the cloud is able to translate access control decisions that achieved consensus in the blowfish into storage access control rules. Comrade, however, shows better performance than Commune. Given the rise of personal clouds we argue that Commune and Comrade find direct applicability in setting up shared repositories that are distributively managed atop of the various personal clouds owned by users. We therefore hope that our findings motivate further research in this areas.

REFERENCES

[1] M. Y. Becker, C. Fournet, and A. D. Gordon, "SecPAL: 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, "TrustManagement 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] "WDMy Cloud," http://www.wdc.com/en/products/products.aspx?id=1140.

