**INTRODUCTION**

Even though the cloud promises a convenient way for users to share files and effortlessly engage in collaborations, it still retains the notion of individual file ownership. That is, each file stored in the cloud is owned by a single user, who can unilaterally decide whether to grant or deny any access request to that file. However, the individual ownership is not suitable for numerous cloud-based applications and collaborations. Consider a scenario where a number of research organizations and industrial partners want to set up a shared cloud repository to collaborate on a joint research project. If all participants contribute their research efforts to the project, then they may want to share the ownership over the collaboration files so that all access decisions are agreed upon among the owners. There are two main arguments why this may be preferred to individual ownership. First, a sole owner can abuse his rights by unilaterally making access control decisions. The community features a number of anecdotes where users revoke access to shared files from other collaborators. Second, even if owners are willing to elect and trust one of them to make access control decisions, the elected owner may not want to be held accountable for collecting and correctly evaluating other owners’ policies. For example, incorrect evaluations may incur negative reputation or financial penalties. In contrast to individual ownership, we introduce a novel notion of shared ownership where n users jointly own a file and each file access request must be granted by a pre-arranged threshold of t owners.We remark that existing cloud platforms, such as Amazon S3 or Dropbox, provide no support for shared ownership policies, and offer only basic access control lists. In short, they are agnostic to the concept of shared ownership. Furthermore, state-of-the-art trust management systems that can support shared ownership policies (e.g., SecPAL , KeyNote , Delegation Logic ) mak all access decisions using a centralized Policy Decision Point (PDP). This is not suitable for enforcing our shared ownership model, because the user who administrates the PDP can arbitrarily change the policy rules set by the owners and enforce his own policies. In this paper, we address the problem of distributed enforcement of shared ownership within cloud storage providers. By distributed enforcement, wemean 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, we introduce 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 SOM model. 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 in , 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 Ethereum blockchain, 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.We summarize our contributions as follows: We 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. We propose a first solution, called Commune, which distributively 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. We propose a second solution, dubbed Comrade, which leverages functionality from the blockchain technology 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 blockchain into storage access control rules, thus requiring minor modifications of existing clouds. We build prototypes of Commune and Comrade and evaluate their performance within Amazon S3 with respect to the file size and the number of users. The remainder of the paper is organized as follows. introduces our notion of shared ownership in a file access control model. details Commune and analyzes its security. I we introduce Comrade and analyze its provisions. evaluates the performance of Commune and Comrade through an implementation within Amazon S3., we discuss further insights with respect to Commune and Comrade. reviews related work. The main idea behind Comrade is that a smart contract can

instantiate a trusted third party that can evaluate user credentials against owners access policies in a trustworthy manner. This is a basic provision of the blockchain technology that holds as long as the security assumptions on the blockchain hold. (We will argue on those assumption in the security analysis). Hence, in Comrade,

a smart contract assists the cloud’s PDP ensuring trustworthy handling of policies and credentials. Differently from Commune, however, Comrade needs the cloud to be “shared-ownership aware” and enforce the policies defined by the smart contract. (Recall that we assume the cloud to correctly enforce access policies.) In more details, cloud accounts in Comrade are not owned by users, but by a smart contract that is running within a blockchain We refer to such a smart contract by owner contract and we rely on it to ensure access control as agreed upon by the file owners. The cloud’s PDP makes access control decisions by evaluating a standardized function within the owner contract, as depicted into grant or deny access rights, the owners submit their votes to the owner contract, which stores them in the blockchain. The PDP’s decision then depends on the access control policy, encoded in the owner contract, and data stored inside the blockchain, the owner contract at system setup. The owner contract tracks each user’s balance (e.g., and punishes them for delayed payments). To pay for cloud storage, the owner contract forwards a part of the users’ deposits to a deposit inside the cloud contract. In turn, the cloud contract deducts the operational costs from the

deposit and requests the deposit to be refilled before it reaches zero. Once it reaches zero, access to the cloud resources is denied and after some grace period the cloud resources are released. i.e., owners’ votes or securely inserted facts.

To perform an action a on file F in Comrade, user Ul proceeds as follows. Ul issues a standard access request to the cloud storage. The request is authenticated using Ul’s private key. The cloud PDP determines the corresponding owner contract for F and evaluates the has Access() function inside that owner contract:

has Access(F;Ul;a). hasAccess() is evaluated based on the contract’s access control policy, the owners’ votes and potentially additional blockchain data. The derived access control decision is then enforced by the cloud’s Policy Enforcement Point (PEP). Notice that the cloud PDP performs this evaluation by locally executing hasAccess() on the current state of the blockchain, i.e., the evaluation triggers no action on the blockchain and requires no fees. The owner contract also manages the users. Users can join the system by sending a request to the owner contract. For every user, the contract’s storage contains the user’s public key, used for authentication and data encryption as explained below. The storage also contains every user’s accounting balance. Finally, the contract contains procedures for initializing and closing the cloud account. Recall that the owner contract stores the votes inside its storage. To minimize the overhead associated with such a voting scheme (i.e., storage costs in the blockchain), Comrade employs a hierarchical file structure and groups files into directories. This allows users to issue a directory specific vote; votes on directories are valid for all contained files and subdirectories (unless a more precise vote exists). We additionally group users into roles by leveraging role-based access control (RBAC) .RBAC allows full flexibility at higher efficiency as owners only need to vote on access rights for the roles. Similar to Commune, we assume that the cloud provider will enforce access control decisions correctly at all time (although the provider might be interested in learning the contents of files). Comrade also ensures fair payment by all owners, protect the cloud provider from free-riding, and punishes unfair behaviour. To do so, each user in Comrade makes a policy-defined deposit at the owner contract at system setup. The owner contract tracks each user’s balance (e.g., and punishes them for delayed payments). To pay for cloud storage, the owner contract forwards a part of the users’ deposits to a deposit inside the cloud contract. In turn, the cloud contract deducts the operational costs from the deposit and requests the deposit to be refilled before it reaches zero. Once it reaches zero, access to the cloud resources is denied and after some grace period the cloud resources are released.