Unix-like Access Permissions in Fully Decentralized File Systems

Johanna Amann[*]
Technische Universität München
amann@so.in.tum.de

Thomas Fuhrmann
Technische Universität München
fuhrmann@so.in.tum.de

Current fully decentralized file systems just offer basic access semantics. In most systems, there is no integrated access control. Every user who can access a file system can read and sometimes even write all data. Only few systems offer better protection. However, they usually involve algorithms that do not scale well, e. g. certificates that secure the file access rights. Often they require online third parties. To the best of our knowledge all systems also lack the standard Unix access permissions, that users have become accustomed to.

In this poster, we show a decentralized, cryptographically secure, scalable, and efficient way to introduce nearly the full spectrum of Unix file system access permissions into distributed file systems. All access rights are enforced by symmetric cryptography only. Moreover, we do not require online third parties.

Our proposal is twofold. It consists of the integrity protection of the file system and the confidentiality protection of the individual files.

For integrity protection, we create a new directory structure. Each user and group has a dedicated directory hierarchy. All files and directories belonging to a user reside in the user's own directory structure. The same holds true for groups. Every file or directory is referenced at least two times in the directory structure: once in the user directory hierarchy and once in the group hierarchy. Fig. **??** shows an example for this structure where the traditional Unix directory structure is mapped onto this new directory structure.

A simple Merkle hash-tree secures the write-operations. The hash of the top-level directory is signed with a symmetric algorithm. Other users can validate the current file system state by recomputing the hashes and checking the signature. Using this approach we can guarantee fork-consistency, the highest level of consistency possible in a fully distributed system.

For data protection, we split each group into a public and a private part. World-readable files are stored in the public part of the group-directory in addition to the owner's user-directory. Files that are only readable by the group are stored in the private part of the group-directory. Files that are only readable by the owning user are only stored in the owner's user-directory. The private group directory is secured with a group encryption key. This encryption key is distributed to the group members using the subset difference encryption scheme. When a file-owner writes a file, the copy in her user-directory is updated. When a group-member updates a file, the copy in the group-directory is updated. Both places have to be checked on access for the most recent version. The file owner can set write permissions in her home directory.

With this scheme we can support nearly all POSIX access semantics. Problems only arise for some rarely used features of POSIX such as the possibility for non-owners to remove empty directories they do not have access to.

To allow more flexible file permissions, POSIX ACLs can be layered on top of our approach. However, as we can show, this introduces a significantly higher overhead and requires the use of asymmetric cryptography.

Our approach can be implemented on every kind of block or chunk oriented storage. We currently implement it on top of IgorFs, a fully decentralized distributed file system developed in our research group.

IgorFs provides a decentralized storage backend for variable-sized data chunks. Each file or directory is represented by one or more chunks. The different chunks are identified by the hash values of their encrypted content. To access a file system within IgorFs a user needs to know the hash and the encryption key of the file system root. From there, hash and encryption keys can be read recursively. Anybody can create a new file system by creating a new (empty) root block. Multiple cryptographically seperated file systems share the network and the underlying storage facilities. Nodes can subscribe to file systems; changes are then propagated to the subscribed nodes within a few seconds.

[*]Student

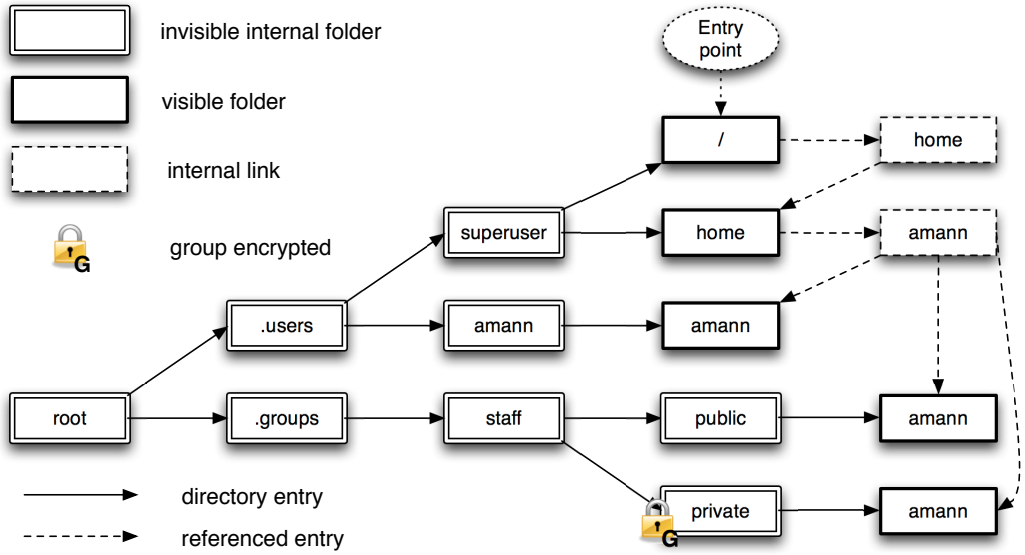

Figure 1: Internal directory structure