SECFS Design Document

Shraman Ray Chaudhuri, Conner Fromknecht

The VSL was implemented as a list of tuples of fields as enumerated by the SUNDR paper, but only kept track of the most recent VS (minimizing bandwidth usage). Signatures were verified by means of asymmetric encryption of the non-signature parts of the VS tuple. Users' updates would include all group I-handles so that multiple updates by the same user are handled seamlessly, without overwriting previous updates. Multi-client security posed the problem of which users to trust on the VSL (a malicious server could arbitrarily populate it). We solved this problem by always trusting 'root' (the file system mounter) to give us the correct usermap, and handled the special case of initializing the usermap by only validating the root VS in the VSL before validating every other VS.

Group-level operations and permissions were implemented by maintaining the invariant that *all* references to group-owned files had to point to a *group* itable (and go through that level of indirection), including references in directory mappings. This made checking group permissions straightforward by checking if any I belongs to a group (invariant) and then ensuring that the user is part of that group.

Encryption was handled at the I-node level, where there is a mapping from uid(s) (multiple if group-encrypted) to an asymmetrically encrypted symmetric key for deciphering the data blocks. Clients would facilitate decryption of the symmetric key on the user's behalf, while encryption/decryption of data blocks was an augmentation of the I-node interface (and crypto) for cleaner system-wide changes.

Extra Feature 1: Extra commands were implemented to allow the user to remove (unlink) and move (rename) files. Regular commands like 'rm -r' are already translated to serial calls to SecFS.remove and SecFS.rmdir so we took out our initial recursive strategy for file removal (which was quite complex). 'Remove' was implemented by a two-step process of deleting all directory references to the I and deleting all itable mappings from to its ihash, taking into consideration group itables and indirection. This involved quite a bit of extension on the itable and directory interfaces (e.g. tables.mod*un*map), but we used these extensions to implement atomic 'rename' by changing inode references between the old and new locations, rather than a simpler but more expensive remove+read+copy operation.

Extra Feature 2: Hiding List of Principals was implemented by changing the Inode to hold a list of *keypairs* where a custom tag accompanies each encrypted symmetric key, and is generated using a user’s private key (secfs.crypto.get_privhash), so that other users cannot determine the users who have permission (the final step would be to disable ‘ls -l’ on encrypted files, but that would break the original test cases). Another note is that we used a secure, irreversible hashlib library so that private keys aren’t leaked (and if they are, then the entire server would be readable anyways).

Extra Feature 3: Garbage Collection was implemented with a flag-and-trash method such that whenever *any* reference to an ‘I’ is deleted, we delete the corresponding ihash in the server-side hash-table. However, this was a very straightforward strategy and doesn’t have support for symbolic links (where deleting one reference shouldn’t delete the file itself, but only a single reference to it). Therefore, we changed our strategy to reference-counting GC, where each itable entry is augmented with the number of references that point to it. We only remove when this number reaches 0.