Linaro

connect
San Francisco 2015

SFO15-503: Secure storage in OP-TEE

James Kung, SY Chiu

Presented by

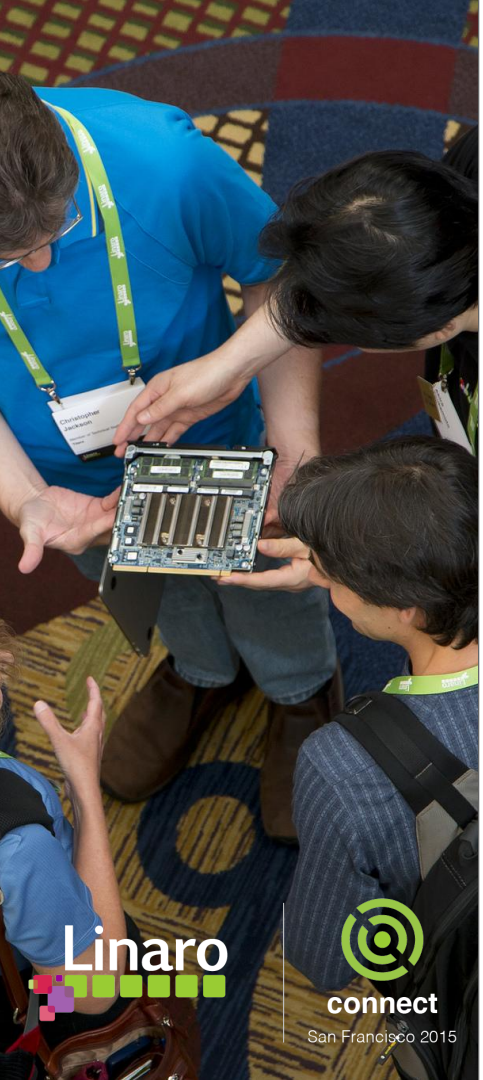
James Kung, Sheng-Yu Chiu

Date

Friday 25 September 2015

Event

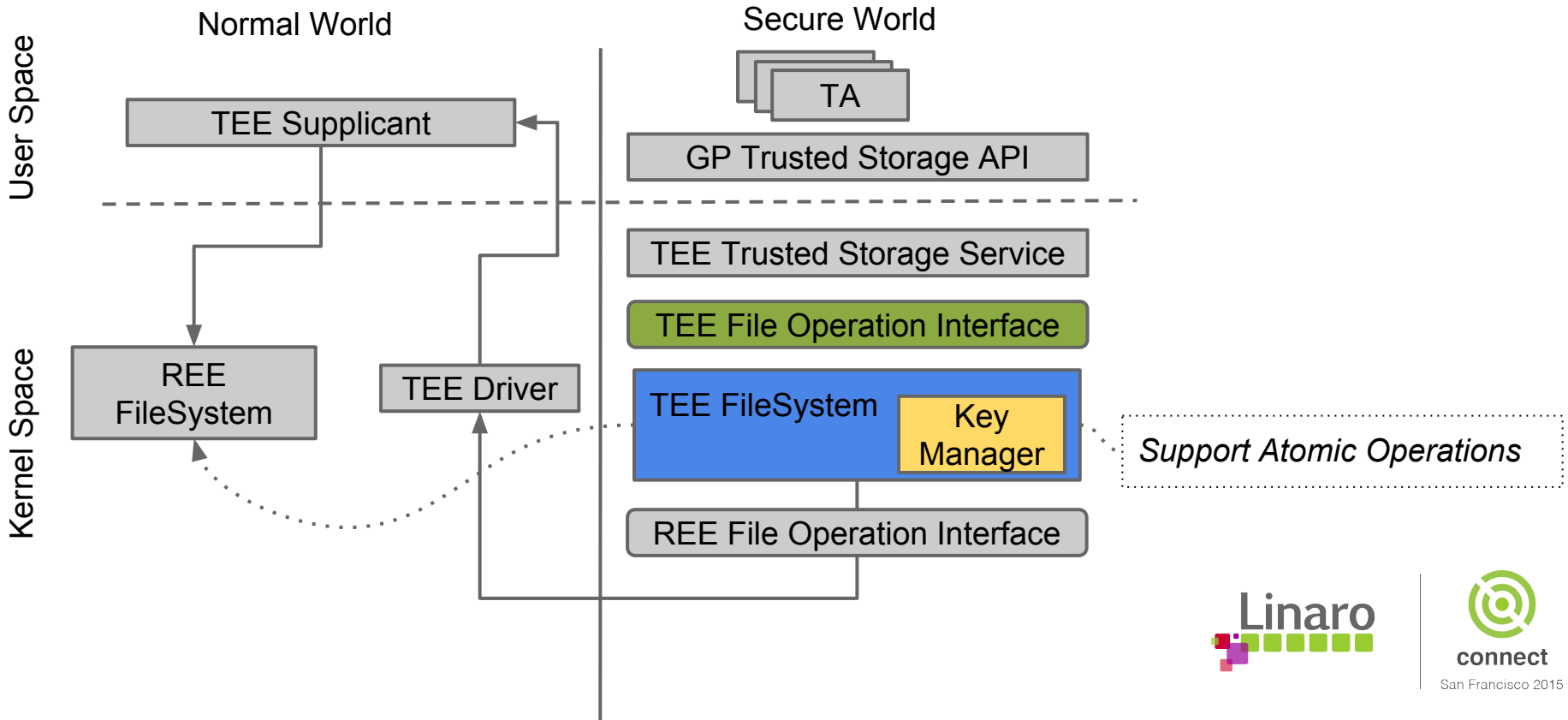
SFO15



Agenda

- Overview
- Key Manager
- Atomic Operations
- Future Work

Secure Storage System Architecture



GP Trusted Storage Requirement

From GlobalPlatform TEE Internal Core API v1.1:

(section 2.5 and section 5.2)

- Must be bound to a particular device
- Guarantees on the **confidentiality** and **integrity** of the data
- Guarantees on the **atomicity** of the operations that modify the storage
- Ability to hide sensitive key material from the TA itself
- Separation of storage among different TAs
- Provide protection against rollback attacks (future work)

TEE File Operation Interface

int (*__open__)(const char *file, int flags, ...);

int (*__close__)(int fd);

int (*__read__)(int fd, void *buf, size_t len);

int (*__write__)(int fd, const void *buf, size_t len);

tee_fs_off_t (*__lseek__)(int fd, tee_fs_off_t offset, int whence);

int (*__rename__)(const char *old, const char *new);

int (*__unlink__)(const char *file);

int (*__ftruncate__)(int fd, tee_fs_off_t length);

int (*__access__)(const char *name, int mode);

int (*__mkdir__)(const char *path, tee_fs_mode_t mode);

tee_fs_dir *(*__opendir__)(const char *name);

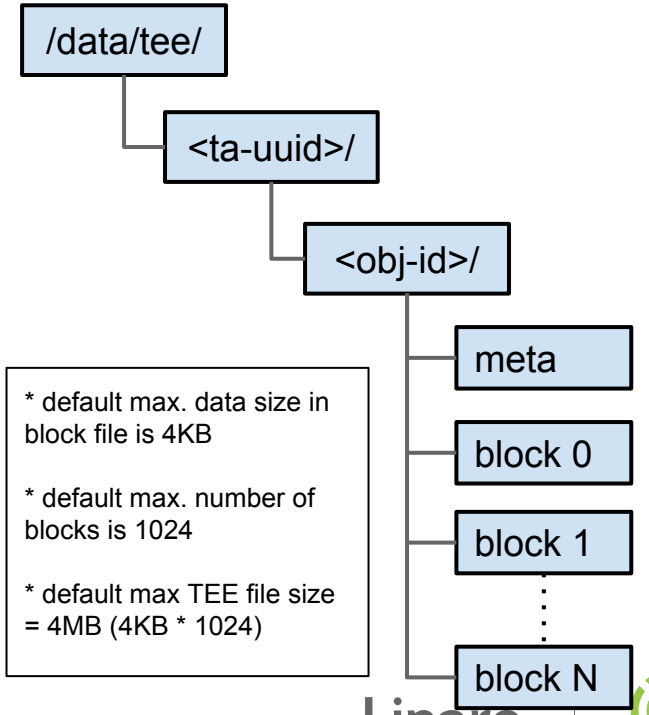
int (*__closedir__)(tee_fs_dir *d);

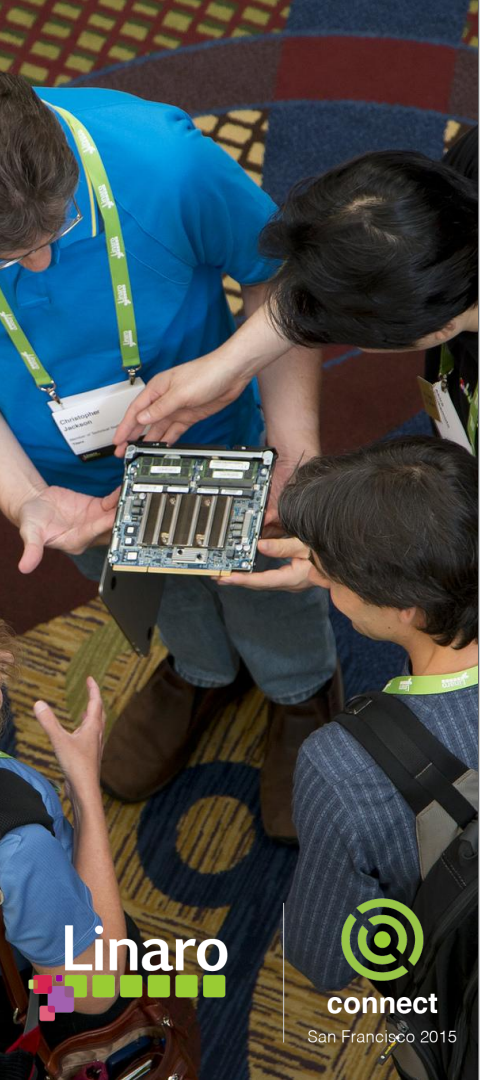
struct tee_fs_dirent *(*__readdir__)(tee_fs_dir *d);

int (*__rmdir__)(const char *pathname);

TEE Storage And File Structure In REE File System

- Provide separate folder for each TA
- TEE filename == object id (provided by TA)
- TEE file == a folder contains a meta file and several block files (folder name is object id)
- Meta file
  - Storing info. of a TEE file. e.g. file length.
  - Always encrypted
- block file
  - Storing TEE file data
  - Optionally encrypted depends on the compile time flag - CFG_ENC_FS





- Overview
- Key Manager
- Atomic Operations
- Future Work

Key Manager

- Provide file encryption/decryption functions
- Key management
  - Secure Storage Key (SSK)
    - Per-device key
    - Used by secure storage subsystem for FEK encryption/decryption
    - Generated and stored in secure memory at boot time
  - File Encryption Key (FEK)
    - Per-file key
    - Used for TEE file encryption/decryption
    - Generated, encrypted and stored in meta file when a new TEE file is created

Key Derivation

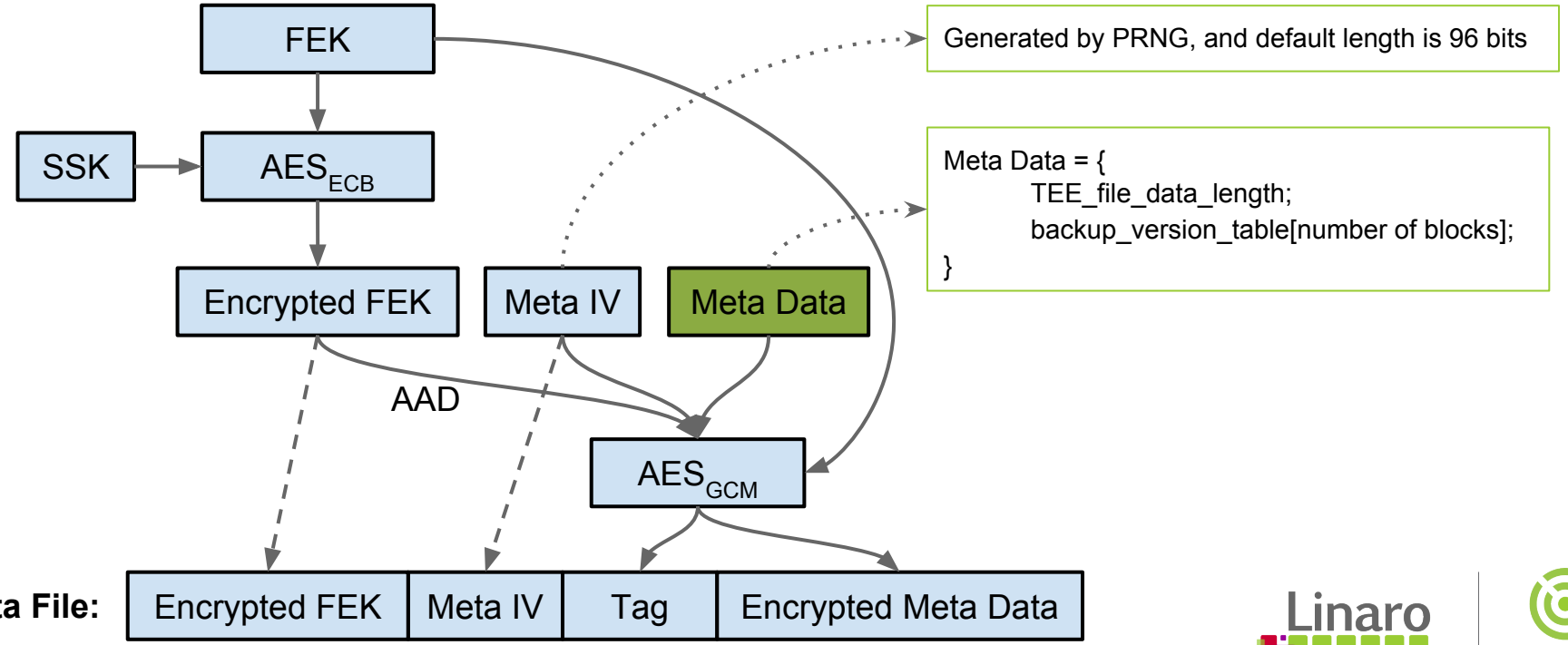
* Can be implemented in platform porting layer

- Secure Storage Key (SSK)
  - $SSK = HMAC_{SHA256}(HUK, Chip\ ID \,||\, \text{“static string”})$
  - HUK: Hardware Unique Key
- File Encryption Key (FEK)
  - By default, generated by Fortuna (PRNG*)
  - Default key length: 128 bits

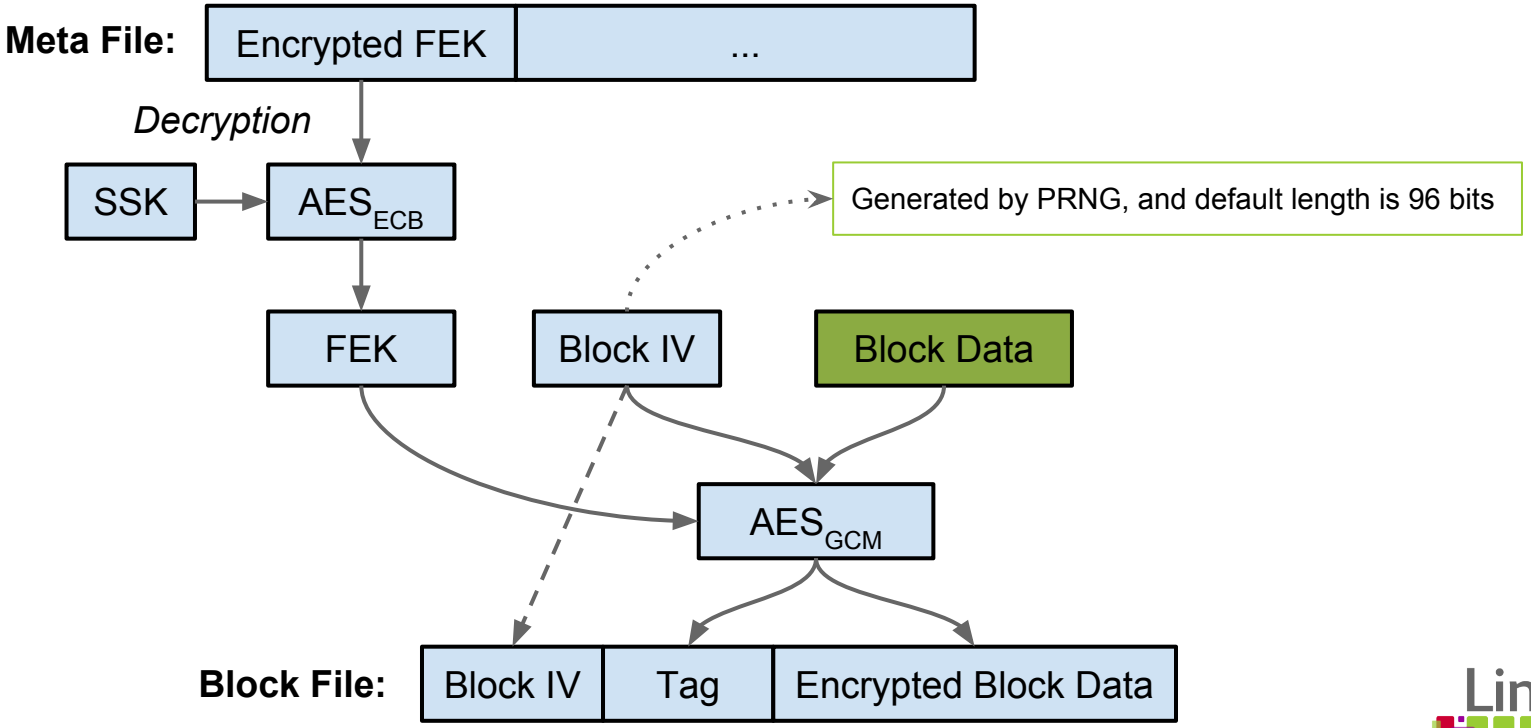
* To avoid other subsystems to generate the same per-device key

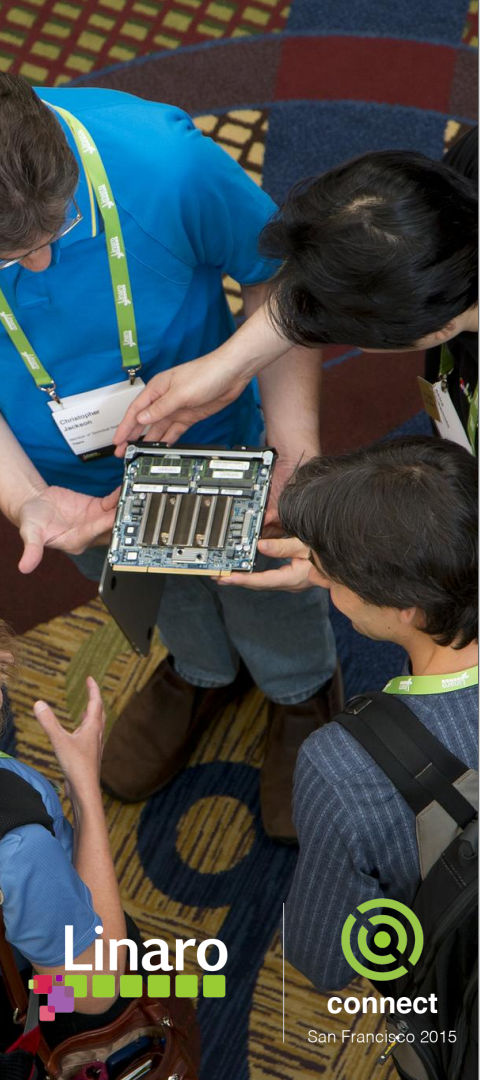
*** It is better to leverage platform H/W RNG(TRNG) if your platform supports that**

Meta Data Encryption



Block Data Encryption





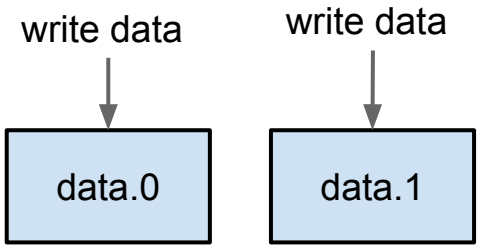
- Overview
- Key Manager
- Atomic Operations
- Future Work

What is Atomic Update

- Each update can only have two results
  - Update succeed
  - Update failed, rollback to old version
- We cannot modify the file contents directly
  - Out-of-place update
  - Modify a copy of file content instead

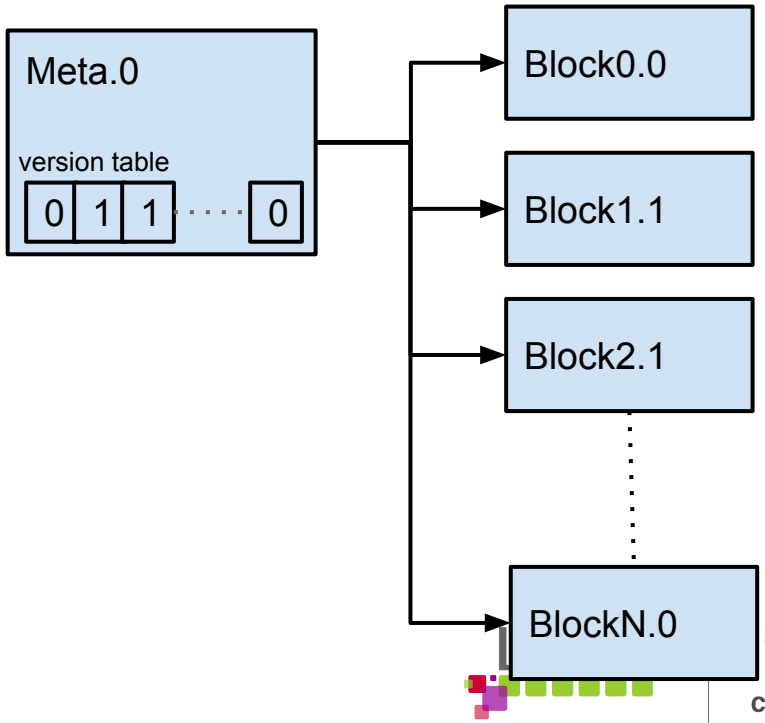
Changes to Support Out-of-place Update

- Both Meta and Block file has an additional attribute called backup version, backup version is 0 or 1
- We toggle the backup version each time when we want to update something
- We should follow the step to update meta or block file



How to Keep Track Backup Version

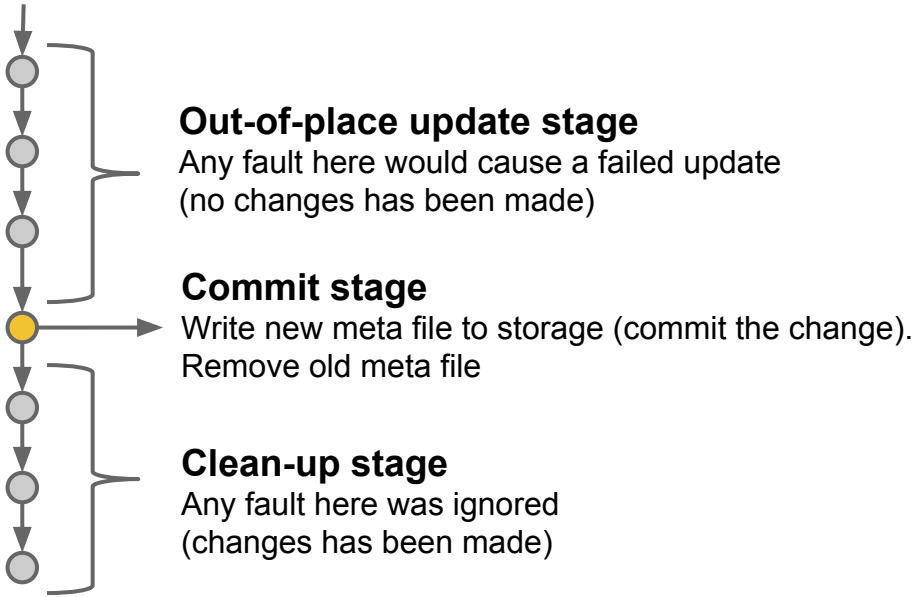
- Introduce a version table in meta data, which stores the backup version of each Block file. This is used to keep track of the backup version of each block file.



GP Internal Core API Says...

- The following operations should be atomic
  - Write
  - Truncate
  - Rename
  - Create/Delete

Ensure Atomic Operation



Out-of-place update stage
Any fault here would cause a failed update
(no changes has been made)

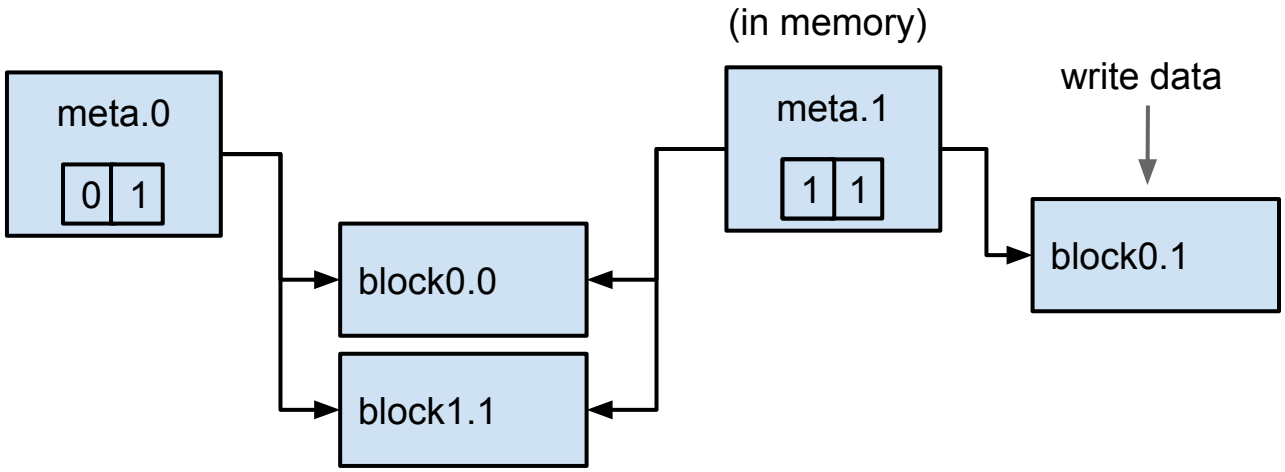
Commit stage
Write new meta file to storage (commit the change).
Remove old meta file

Clean-up stage
Any fault here was ignored
(changes has been made)

* should having a tool to do garbage collection to clean up invalid block or meta. The idea is similar to fsck for Linux file system.

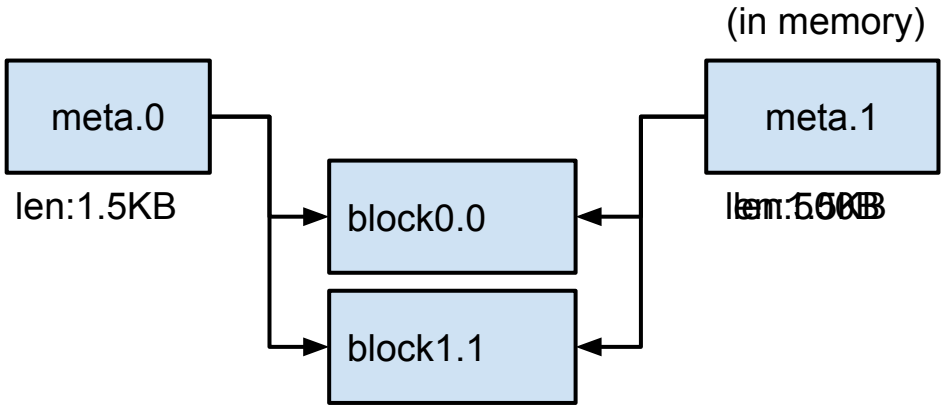
Atomic Write Operation

Assume block size is 1KB, we want to write 20 Bytes at position 0



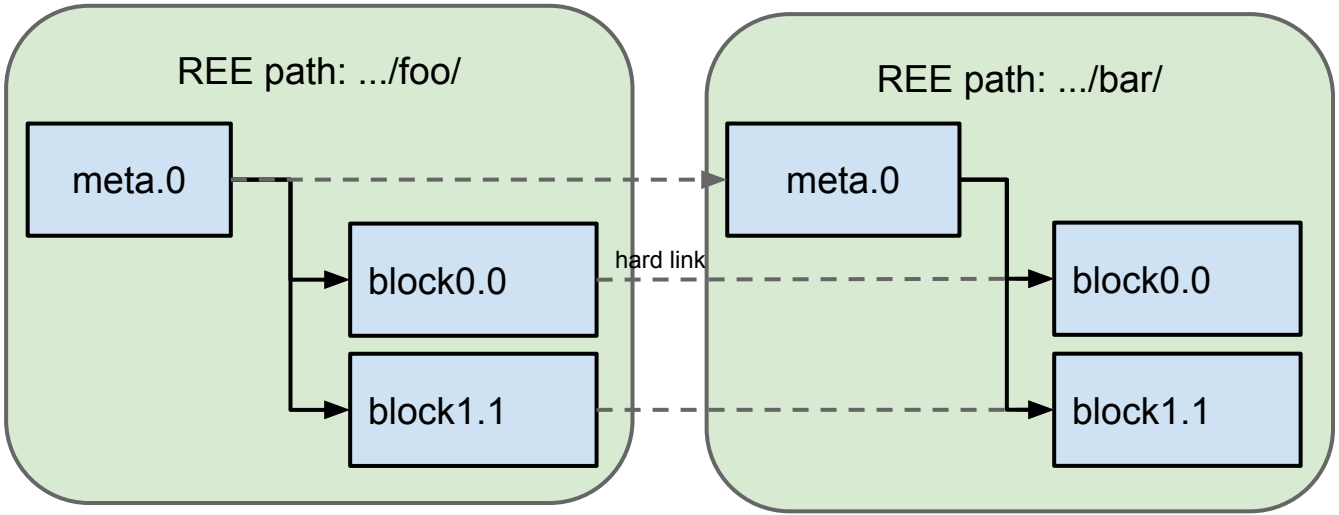
Atomic Truncate Operation

Assume block size is 1KB, file length is 1.5 KB, we want to truncate it to 500 Bytes



Atomic Rename Operation

Assume we want to rename ***foo*** to ***bar***

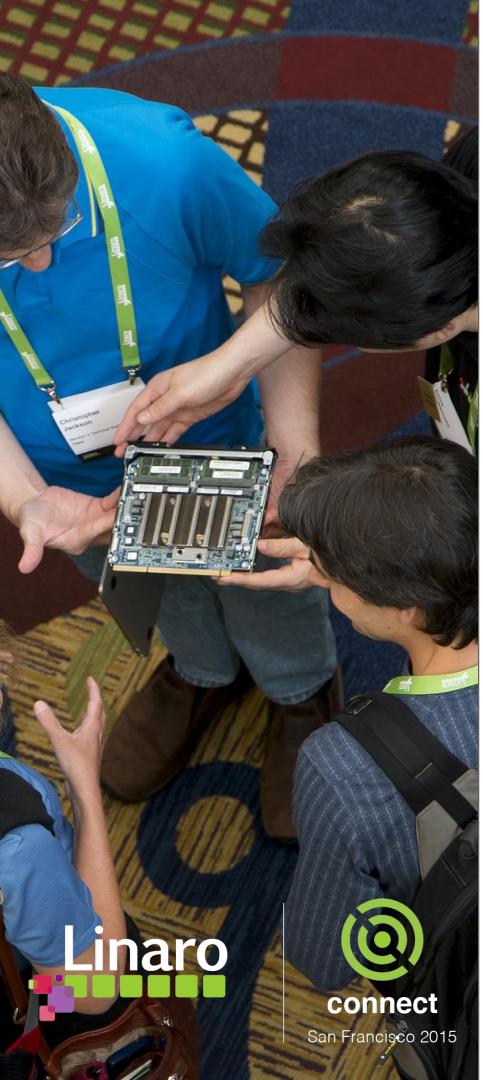


* ISO C requires `rename` to be atomic. We are considering to use it.
** Atomic override is not supported yet.

Atomic Create/Delete Operation

- Create
  - If the meta file is successfully encrypted and stored, we are done.
  - If any failure occurs during meta file creation, no file is created.
- Delete
  - Rename <filename> to <filename>.trash
  - Then remove <filename>.trash

* `unlink` is not need to be atomic. Thus we can't simply unlink('meta')



- Overview
- Key Manager
- Atomic Operations
- Future Work

Binding TEE File to TA

- TEE file folder can be copied or moved into other TA's folder in normal world
- We do not have a mechanism to prevent TA's from opening other TA's TEE file(s)
- A simple way to bind TEE file to TA is adding TA's UUID to meta file

Rollback Attack Detection

- TEE file content can be backed up and then restored without any error. TEE should be able to detect this kind of actions.
- The solution is to add a file version number in meta file and store it in another safe place.
- The safe place can be
  - normal world file system (Protection Level = 100)
  - RPMB (Protection Level = 1000)

Thank you~