Inline Encryption

Satya Prateek Tangirala
satyat@google.com

Motivation

Turns out users care about their data being secure

Lost/stolen phones

Inline encryption speeds up reading/writing encrypted stuff to disk

Many vendors with inline encryption hardware - each with their own patches - so we'd like to reduce the maintenance burden.

Generic Kernel Image (GKI)

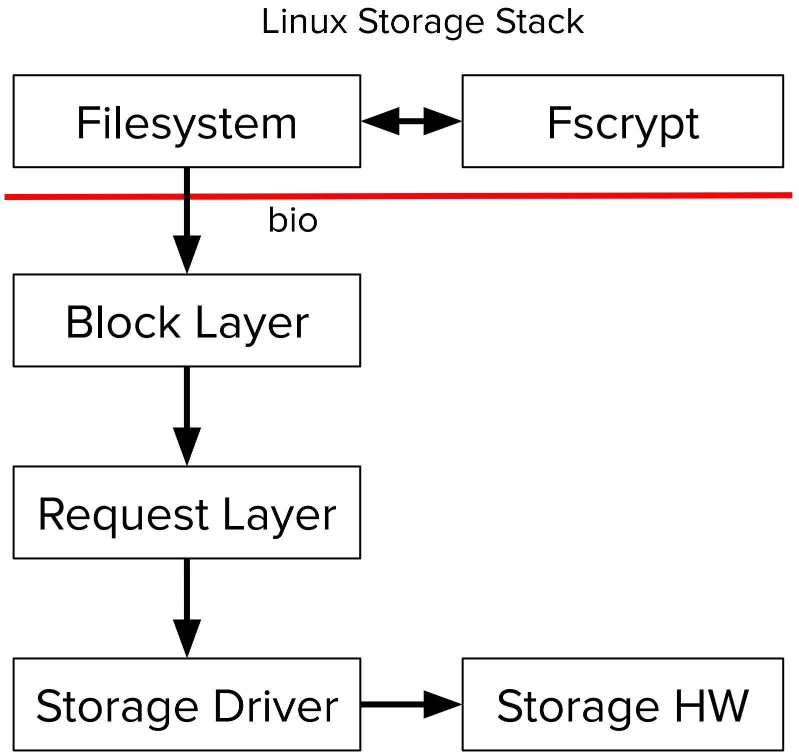
FDE, FBE and fscrypt

FDE - single key for all data on disk

FBE - Per-file encryption key

EXT4, F2FS, UBIFS -> fscrypt

```
struct bio {
        location in memory;
        location in disk;
        size;
        operation (e.g. READ/WRITE);
}
```



Inline Encryption

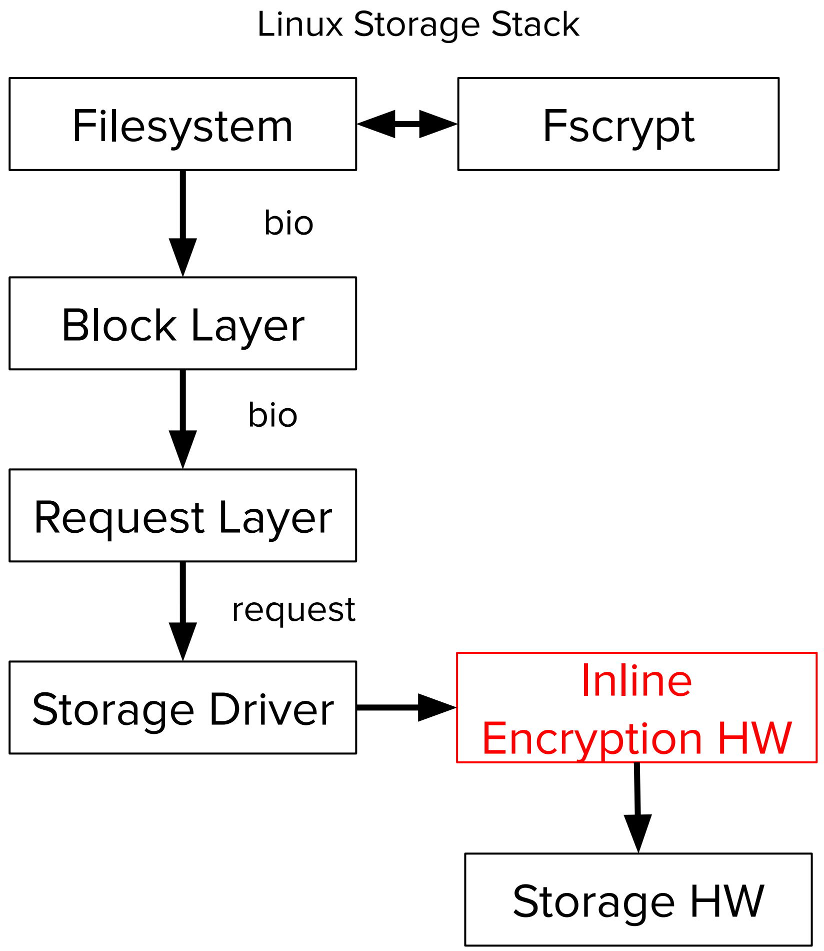
Android uses fscrypt for FBE

Inline encryption hardware speeds up FBE

Inline encryption hardware has some programmable keyslots

Take (data, keyslot) and en/decrypt data with key in keyslot as data flows through

Distinct from self encrypting drives and OPAL!



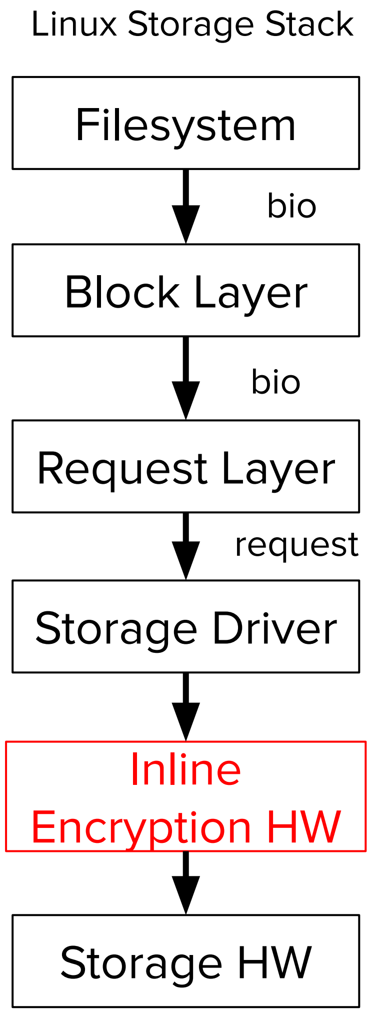
The context

Linux doesn't currently support Inline encryption hardware

Fscrypt already gives us a way of associating a file with an encryption context

Most of the stack needs to be encryption aware

Many past attempts



High Level Objectives

Useful for any inline encryption hardware (i.e. not particular to some storage driver or inline encryption hardware)

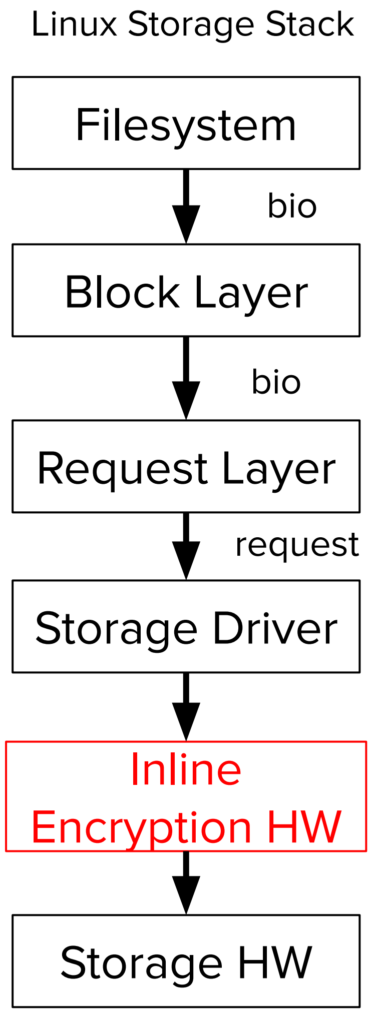
Must be possible/easy to add support for our approach to storage drivers

Handle #in-flight encryption contexts > #keyslots in hardware

Handle differences/quirks in inline encryption hardware

Should not be tied to a particular filesystem

Kernel crypto API fallback in case inline encryption is not available



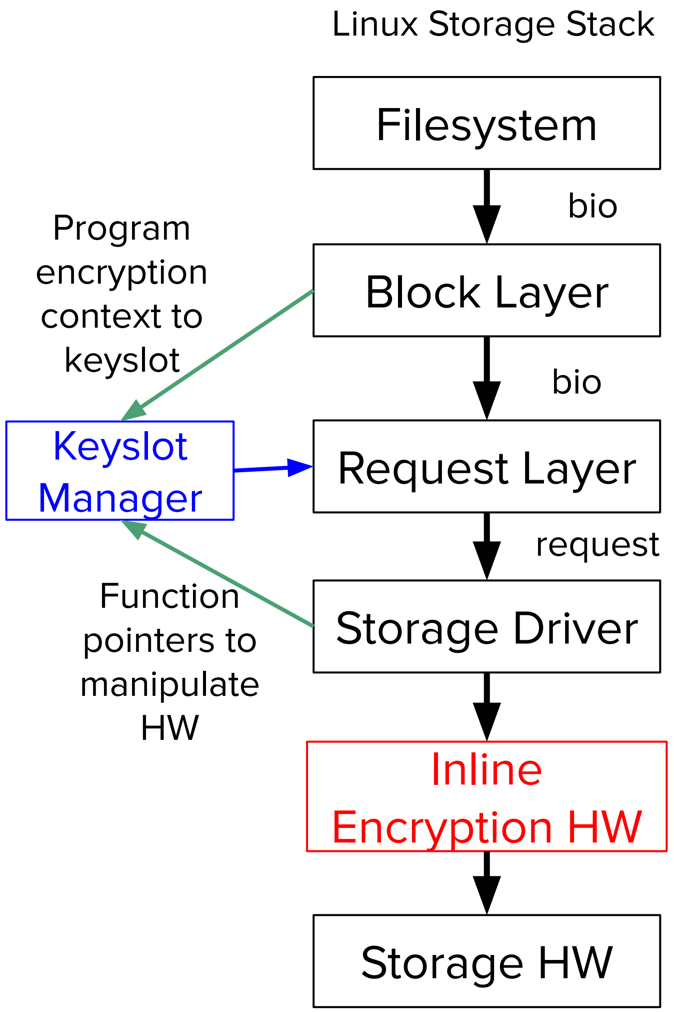
bio-crypt-ctx and Keyslot Manager

Add struct bio-crypt-ctx to struct bio

Introduce Keyslot Manager (KSM) to solve the keyslot sharing problem

Upper layers call get_keyslot(encryption_context) on KSM - KSM returns a slot that has the encryption context programmed, and holds a refcount on that slot

Lower layer storage driver will set up the KSM in its request queue

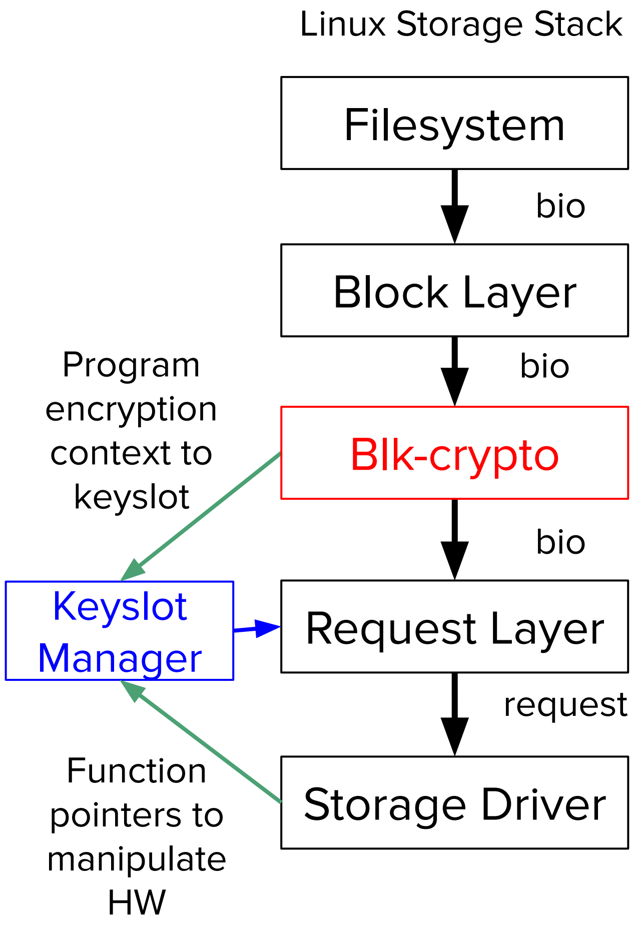


Blk-crypto

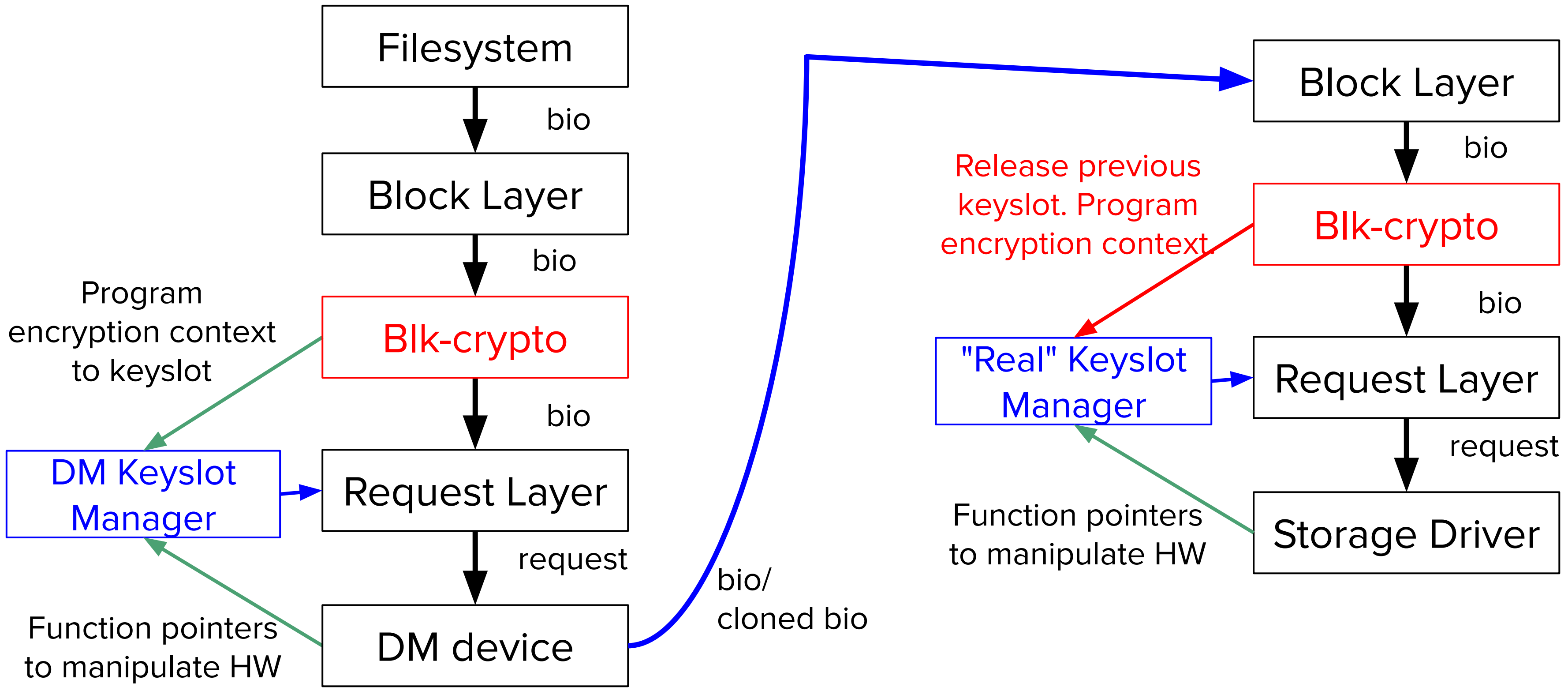
Add Blk-crypto between Block and Request layers

Uses keyslot manager in request queue to program encryption context into a keyslot.

Contains a kernel crypto API fallback



Blk-crypto and stacked devices



Passthrough Keyslot Manager

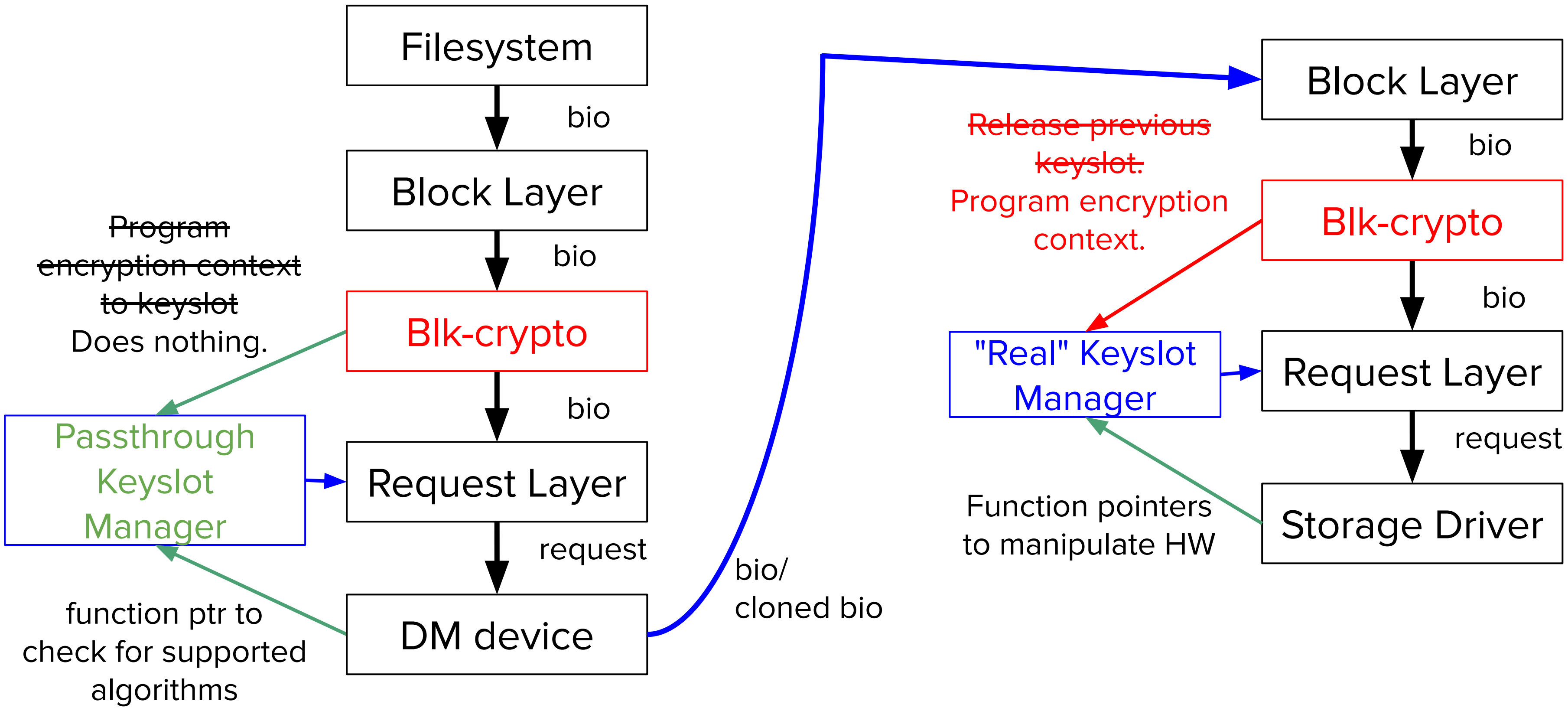
Does not actually manage any keyslots (so not much memory needed)

Does not need function ptrs to manipulate HW (like program keys into keyslots)

Still needs a function ptr to query which algorithms are supported

blk-crypto will do nothing if request_queue has passthrough KSM

Blk-crypto and stacked devices


Filesystem
bio
Block Layer
bio
~~Program encryption context to keyslot~~ Does nothing.
Blk-crypto
bio
Passthrough Keyslot Manager
Request Layer
request
function ptr to check for supported algorithms
DM device
bio/ cloned bio
Block Layer
bio
~~Release previous keyslot.~~ Program encryption context.
Blk-crypto
bio
"Real" Keyslot Manager
Request Layer
request
Function pointers to manipulate HW
Storage Driver

Interface to filesystems

Set up the bio_crypt_ctx for each bio - i.e. call bio_crypt_set_ctx (bio, key, algorithm, data_unit_num, data_unit_size_bits, gfp_mask) any time before the bio is submitted.

Call blk_crypto_start_using_mode(algorithm, data_unit_size, request_queue) at some point before submitting a bio that is encrypted with that algorithm - must NOT be called from the data path

Tear down the bio_crypt_ctx by calling bio_crypt_free_ctx(bio)

Interface to storage drivers

Setup a keyslot manager in their request queue -
keyslot_manager_create(#keyslots, function ptrs, void *private_data)

For each request, call bio_crypt_get_keyslot(bio) to get the keyslot it should use for inline encryption for this request.

Or they can set up a passthrough keyslot manager

UFS inline encryption support

"Proof of concept" and for context

Following the JEDEC UFSHCD v2.1 spec

UFS driver is split into UFS base code and vendor specific driver - inline encryption additions are in base code

Introduce variant ops to handle UFS Inline encryption hardware quirks

Fscrypt and F2FS

"Proof of concept" and for context

New policy flag and on-disk format for inline encrypted files

DM Changes!

Testing and adoption by partners

Qualcomm and Mediatek on board

Variant ops have been useful to deal with spec ambiguity

Both partners currently testing the patches

Personal testing with xfstests (through kernel crypto API fallback) and Pixel 3 (through UFS driver with inline crypto additions)

Future

eMMC and other storage drivers

Add support to other filesystems - like EXT4

Thanks everyone!

Especially to:

Mike Halcrow for his design doc from which this work was largely inspired

Eric Biggers, Paul Crowley, Paul Lawrence and Ted Ts'o for their invaluable guidance and wisdom

Jon Corbet for writing an article on this on LWN - https://lwn.net/Articles/797309/