Writing drivers for the Linux Crypto subsystem

Marek Vašut <marex@denx.de>

May 18, 2014

Marek Vasut

- Software engineer at DENX S.E. since 2011
  - Embedded and Real-Time Systems Services, Linux kernel and driver development, U-Boot development, consulting, training.
- Versatile Linux kernel hacker
- Custodian at U-Boot bootloader

The kernel Crypto API?

- Generic in-kernel transformation API
- Can do Cipher, Hash, Compress, RNG,...
- Used by:
  - Network stack: IPsec, ...
  - Device Mapper: dm-crypt, RAID, ...
  - AF_ALG and thus possibly userland

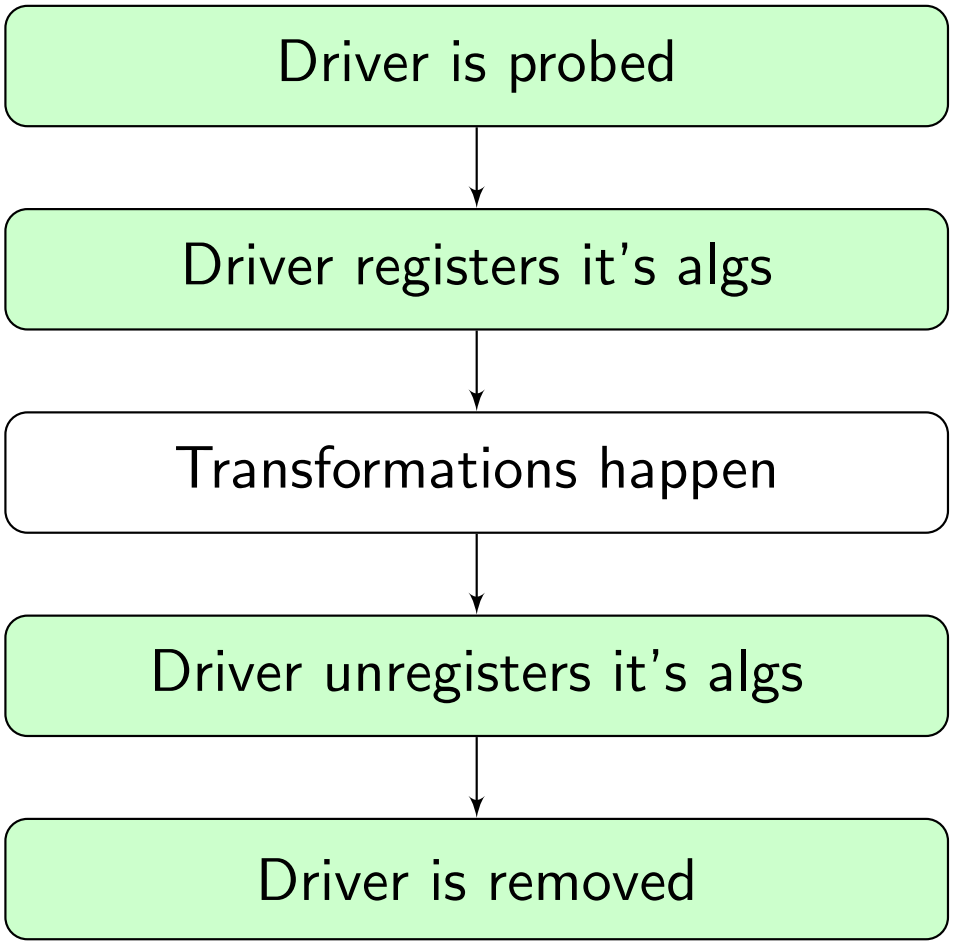
Therefore, you want your drivers to be well written.

Today's plan

- Go through part of `drivers/crypto/geode-aes.c`
- Inspect how a basic transformation driver is written
- Point out mistakes that are easy to make in a new driver

Registering a transformation

Lifespan of a Crypto API driver:



Example registration

```
static int geode_aes_probe(struct pci_dev *dev,
                           const struct pci_device_id *id)
{
    ...
    ret = crypto_register_alg(&geode_alg);
    if (ret)
        goto eiomap;
    ...
    return 0;
    ...
eiomap:
    dev_err(&dev->dev, "GEODE AES initialization failed.\n");
    return ret;
}
```

Registration: important bits

- Crypto API does not track device instances
- Careful when probing multiple devices of the same type
  - The probe() function doesn't need locking
  - Static private data pointer for instances
- `crypto_register_alg(struct crypto_alg *alg)`
  `crypto_register_algs(struct crypto_algs *algs, int count)`
- `crypto_unregister_alg(struct crypto_alg *alg)`
  `crypto_unregister_algs(struct crypto_algs *algs, int count)`

The struct crypto_alg

Block cipher descriptor (telephone list):

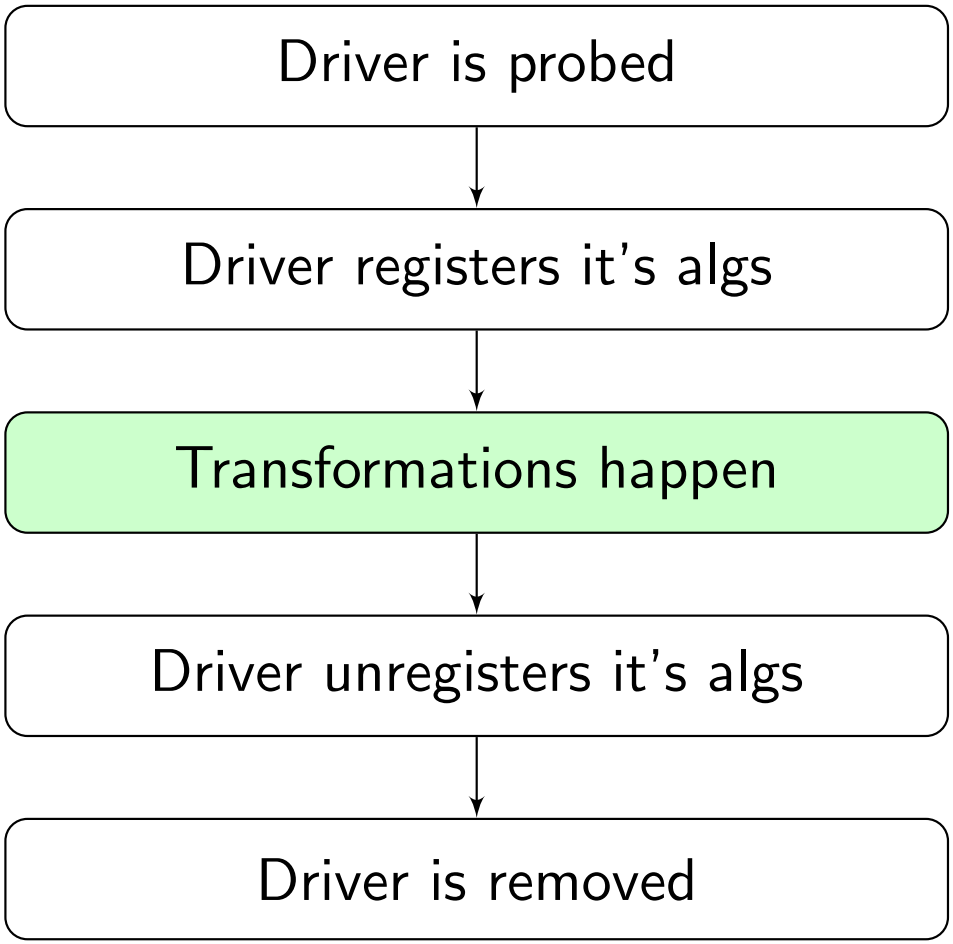
```
static struct crypto_alg geode_alg = {
    .cra_name         = "aes",
    .cra_driver_name  = "geode-aes",
    .cra_priority     = 300,
    .cra_module       = THIS_MODULE,

    .cra_blocksize    = AES_BLOCK_SIZE,
    .cra_alignmask    = 15,

    .cra_ctxsize      = sizeof(struct geode_aes_op),
    .cra_init         = fallback_init_cip,
    .cra_exit         = fallback_exit_cip,
    ----->8----
```

The struct crypto_alg

```
        .cra_flags      = CRYPTO_ALG_TYPE_CIPHER |
                          CRYPTO_ALG_NEED_FALLBACK,
        .cra_u      = {
            .cipher     = {
                .cia_min_keysize    = AES_MIN_KEY_SIZE,
                .cia_max_keysize    = AES_MAX_KEY_SIZE,
                .cia_setkey         = geode_setkey_cip,
                .cia_encrypt        = geode_encrypt,
                .cia_decrypt        = geode_decrypt
            }
        }
};
```

High-level view

Lifespan of a Crypto API driver:



User vs. struct crypto_alg

WARNING: interleaved code, put your 4D glasses on:

```
static struct crypto_alg geode_alg = {
        .cra_ctxsize = sizeof(struct geode_aes_op),
struct crypto_cipher *tfm = crypto_alloc_cipher(alg, type, mask);
        .cra_init = fallback_init_cip,

        .cra_u.cipher = {
crypto_cipher_setkey(tfm, key, keylen);
                .cia_setkey = geode_setkey_cip,

crypto_cipher_encrypt_one(tfm, out, in);
                .cia_encrypt = geode_encrypt,
        }

crypto_free_cipher(tfm);
        .cra_exit = fallback_exit_cip,
};
```

The transformation object

- Representation of transformation instance
- Usually encapsulated in bigger structures
- Contains pointers to transformation functions
- Contains private data for the transformation
- Can be accessed concurrently!

Setting the key

```
static int geode_setkey_cip(struct crypto_tfm *tfm,
                            const u8 *key, unsigned int len)
{
    struct geode_aes_op *op = crypto_tfm_ctx(tfm);
    unsigned int ret;

    op->keylen = len;

    if (len == AES_KEYSIZE_128) {
        memcpy(op->key, key, len);
        return 0;
    }

    if (len != AES_KEYSIZE_192 && len != AES_KEYSIZE_256) {
        /* not supported at all */
        tfm->crt_flags |= CRYPTO_TFM_RES_BAD_KEY_LEN;
        return -EINVAL;
    }
    ---->8----
```

Setting the key recap

- Get TFM private data – `crypto_tfm_ctx(tfm)`
- The key is copied into the private data
- Check the validity of the key size
- All key sizes must be handled

Let's encrypt a block

Buffer size is always `.cra_blocksize` in this case.

```
static void
geode_encrypt(struct crypto_tfm *tfm, u8 *out, const u8 *in)
{
    struct geode_aes_op *op = crypto_tfm_ctx(tfm);

    if (unlikely(op->keylen != AES_KEYSIZE_128)) {
        crypto_cipher_encrypt_one(op->fallback.cip, out, in);
        return;
    }
    ---8<--->8---
    actual_aes_crypt(tfm, out, in);
}
```

Hardware limitations

- Hardware can have limitations/bugs
- Crypto API can still use such hardware
- The `.cra_init` provides means to put workaround in place:

```
static int fallback_init_cip(struct crypto_tfm *tfm)
{
    const char *name = crypto_tfm_alg_name(tfm);
    struct geode_aes_op *op = crypto_tfm_ctx(tfm);

    op->fallback.cip = crypto_alloc_cipher(name, 0,
                CRYPTO_ALG_ASYNC | CRYPTO_ALG_NEED_FALLBACK);

    if (IS_ERR(op->fallback.cip)) {
        printk(KERN_ERR "Error allocating fallback algo %s\n",
               name);
        return PTR_ERR(op->fallback.cip);
    }
    return 0;
}
```

Hardware limitations

```
static int geode_setkey_cip(struct crypto_tfm *tfm,
                            const u8 *key, unsigned int len)
    ---->8----
    /* The requested key size is not supported by HW, do a fallback */
    op->fallback.cip->base.crt_flags &= ~CRYPTO_TFM_REQ_MASK;
    op->fallback.cip->base.crt_flags |= (tfm->crt_flags & CRYPTO_TFM_REQ_MASK);

    ret = crypto_cipher_setkey(op->fallback.cip, key, len);
    if (ret) {
        tfm->crt_flags &= ~CRYPTO_TFM_RES_MASK;
        tfm->crt_flags |=
           (op->fallback.cip->base.crt_flags & CRYPTO_TFM_RES_MASK);
    }
    return ret;
}

static void geode_encrypt(struct crypto_tfm *tfm, u8 *out, const u8 *in)
{
    struct geode_aes_op *op = crypto_tfm_ctx(tfm);
    if (unlikely(op->keylen != AES_KEYSIZE_128)) {
        crypto_cipher_encrypt_one(op->fallback.cip, out, in);
        return;
    }
    ...
```

Alignment pitfalls

- In simple case of `cipher`, Crypto API realigns buffers
- Realigning adds overhead
- Use `crypto_*_alignmask()` to learn of the needs

```
static struct crypto_alg geode_alg = {
    ...
    .cra_alignmask   = 15,
    ...
};
```

Recap

We made it through simple block cipher!

```
static struct crypto_alg geode_alg = {
    .cra_name         = "aes",
    .cra_driver_name  = "geode-aes",
    .cra_priority     = 300,
    .cra_module       = THIS_MODULE,

    .cra_blocksize    = AES_BLOCK_SIZE,
    .cra_alignmask    = 15,

    .cra_ctxsize      = sizeof(struct geode_aes_op),
    .cra_init         = fallback_init_cip,
    .cra_exit         = fallback_exit_cip,
    .cra_u.cipher     = {},
```

Testing the new driver

- TCrypt test framework in kernel
- Device Mapper torture
- IPsec torture
- Userland bombing from OpenSSL

TCrypt

- Available in Linux kernel (crypto/tcrypt.c)
- The "does it even work" testsuite.
- Enable with `CONFIG_CRYPTO_TEST`
- If possible, leave enable

\+ Easily available

\+ Basic algo validity test

– Does not do stress testing

Device Mapper torture

- Available in Linux kernel (drivers/md/)
- Use dm-crypt for block cipher testing (aes-cbc)
- Use dm-crypt for hash testing (md5, crc32)
- Use dm-raid5 for XOR offload testing
- Enable with `CONFIG_DM_CRYPT`
- Many examples how to use `cryptsetup`, but look at: `--hash`, `--cipher`, `--key-size`

\+ Easily available

\+ Use for stress testing

\+ Use to simulate real workload

– Produces small payloads of data

IPsec torture

- Available in Linux kernel (net/)
- Look for AH, ESP v4/v6 support
- Nice for testing hashing and AEAD
- Enable with `CONFIG_INET(6)_(AH,ESP)`
- Try setting up an IPsec tunnel and inspect a crash :-)

+ Easily available
+ Use for stress testing
+ Use to simulate real workload

What else can we do?

- Asynchronous tfm on large data
- Synchronous tfm on large data
- Async/Sync hashing on large data
- AEAD (MAC) on data
- Generate random numbers
- Compress/decompress

Advanced topics

Here are a few keywords on the advanced side of Crypto API:

- Asynchronous tfm on large data
  - Generic ScatterWalk
  - Scatterlist alignment
  - Callback handling
  - Backlog queue handling
  - Crypto request queues
- Async/Sync hashing on large data
  - Harboring obscure hashing engines

Thank you for your attention!

Contact: Marek Vasut <marex@denx.de>
Crypto ML: linux-crypto@vger.kernel.org