Confessions of a security hardware driver maintainer

LINUX SECURITY SUMMIT

Gilad Ben-Yossef
Principal Software Engineer

About me

My name is Gilad Ben-Yossef.

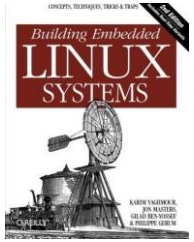
I work on upstream Linux kernel cryptography and security in genera,l and arm® TrustZone® CryptoCell® support in particular.

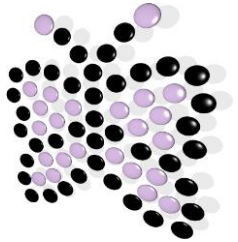
I've been working in various forms with and on the Linux kernel and other Open Source projects for close to twenty years – i.e. I'm an old bugger… ☺

I co-authored "Building Embedded Linux Systems" 2nd edition from O'Reilly.

I'm a co-founder of HaMakor, an Israeli NPO for free and open source software and of August Penguin, the longest running Israeli FOSS conference.



arm

What I am NOT going to talk about today

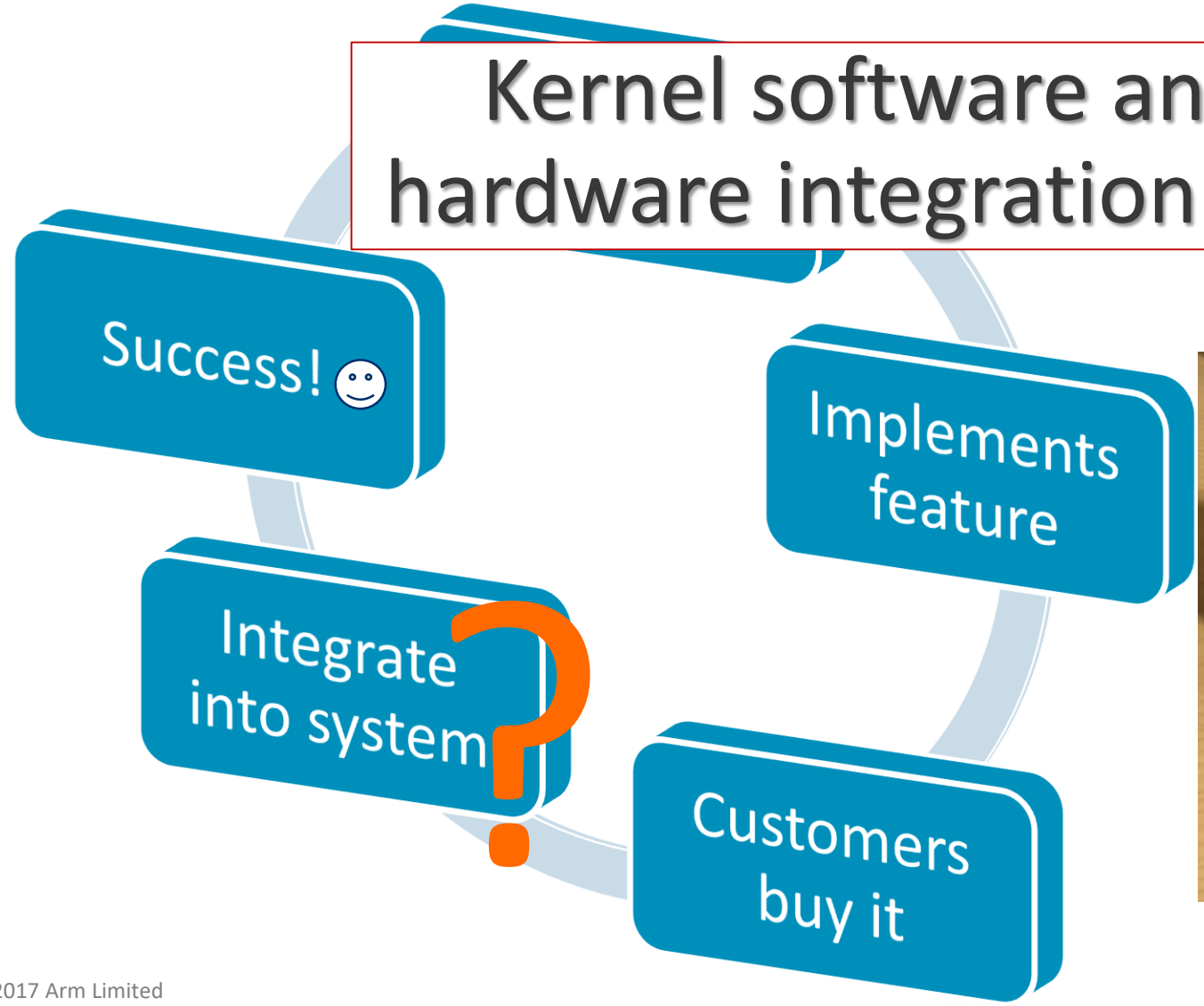
The CryptoCell out-of-tree device driver went into staging

```
commit abefd6741d540fc624e73a2a3bdef2397bcbd064
Author: Gilad Ben-Yossef <gilad@benyossef.com>
Date:   Sun Apr 23 12:26:09 2017 +0300

    staging: ccree: introduce CryptoCell HW driver

    Introduce basic low level Arm TrustZone CryptoCell HW support.
    This first patch doesn't actually register any Crypto API
    transformations, these will follow up in the next patch.

    This first revision supports the CC 712 REE component.

    Signed-off-by: Gilad Ben-Yossef <gilad@benyossef.com>
    Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>
```

arm

What I am going to talk about…

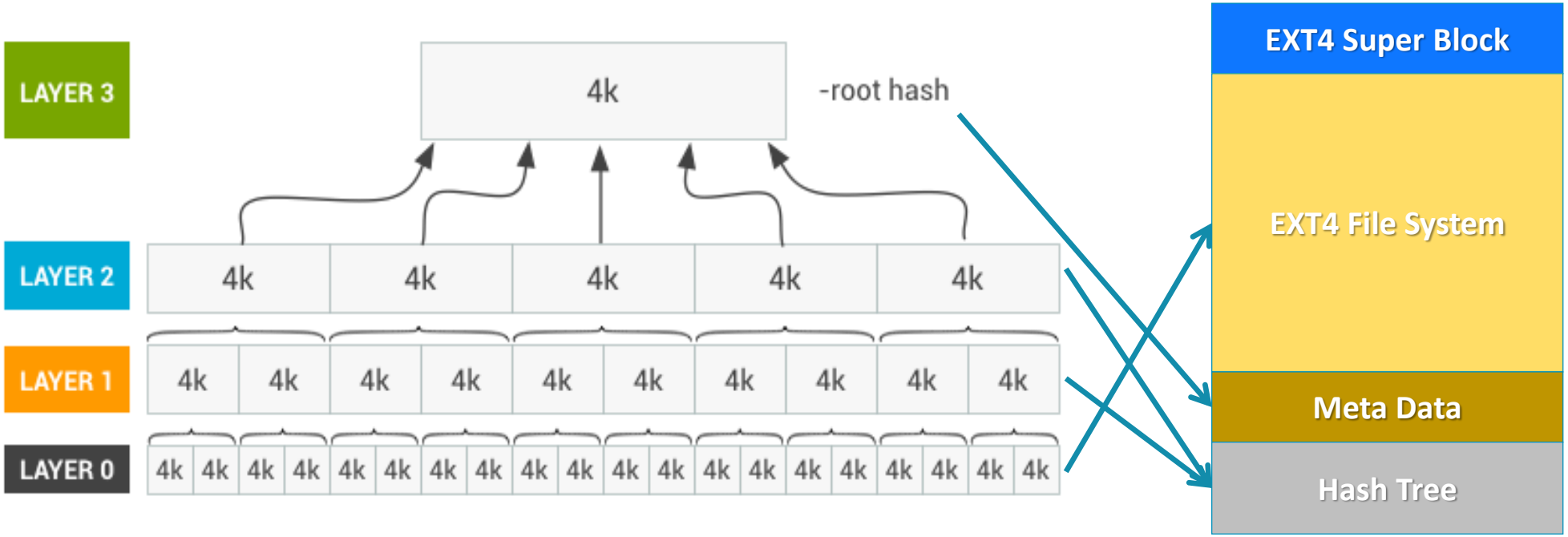
Kernel software and security hardware integration mismatches



arm

DM-Verity

Device-Mapper's "verity" target provides transparent integrity checking of block devices using a cryptographic digest provided by the kernel crypto API.

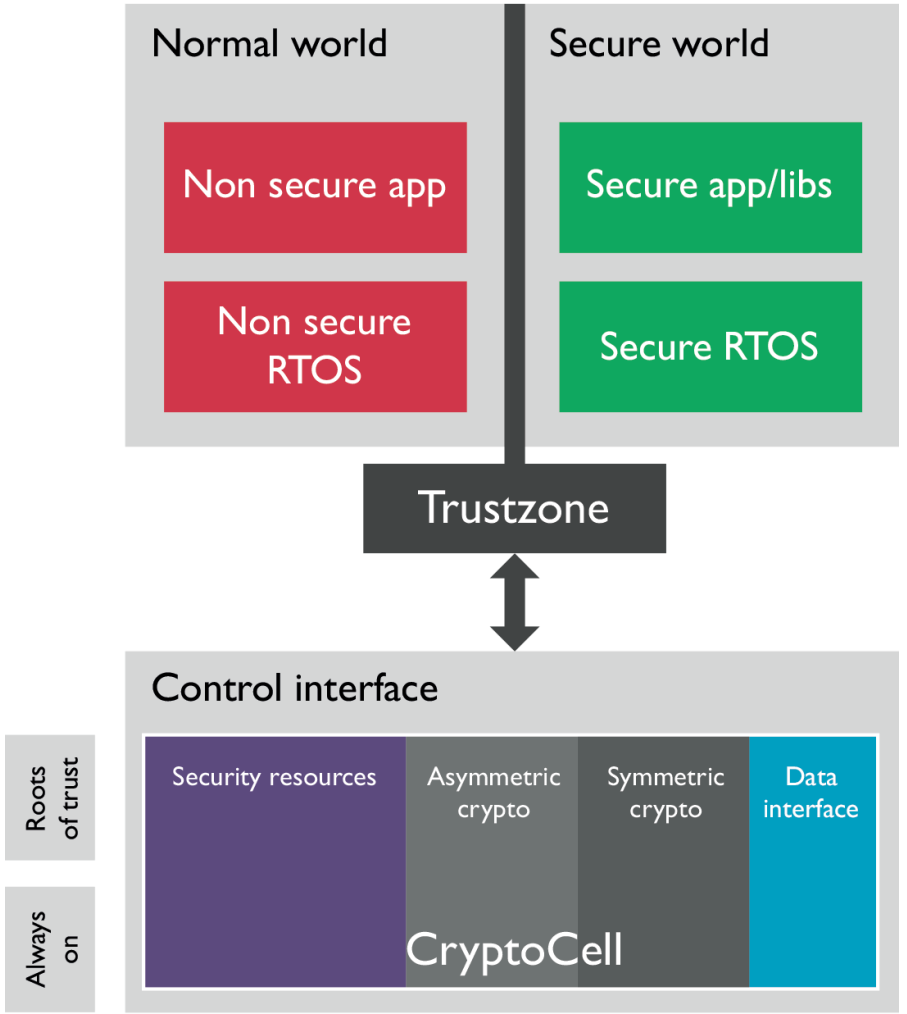


arm

What is Arm® TrustZone® CryptoCell?

Readers Digest version:

- It's a HW block that handles system security.
- It provides crypto, root of trust, secure boot and debug.
- It goes in a SoC, outside the core.
- It serves both trusted and untrusted worlds.



arm

Allowing DM-Verity to use async. transformations

The upstream DM-Verity implementation was using the synchronous crypto API designed for in-core based implementations.

This caused customers to use various unspeakable hacks to get it to work.

We changed the implementation to use the asynchronous API that allows use of off core crypto accelerators.

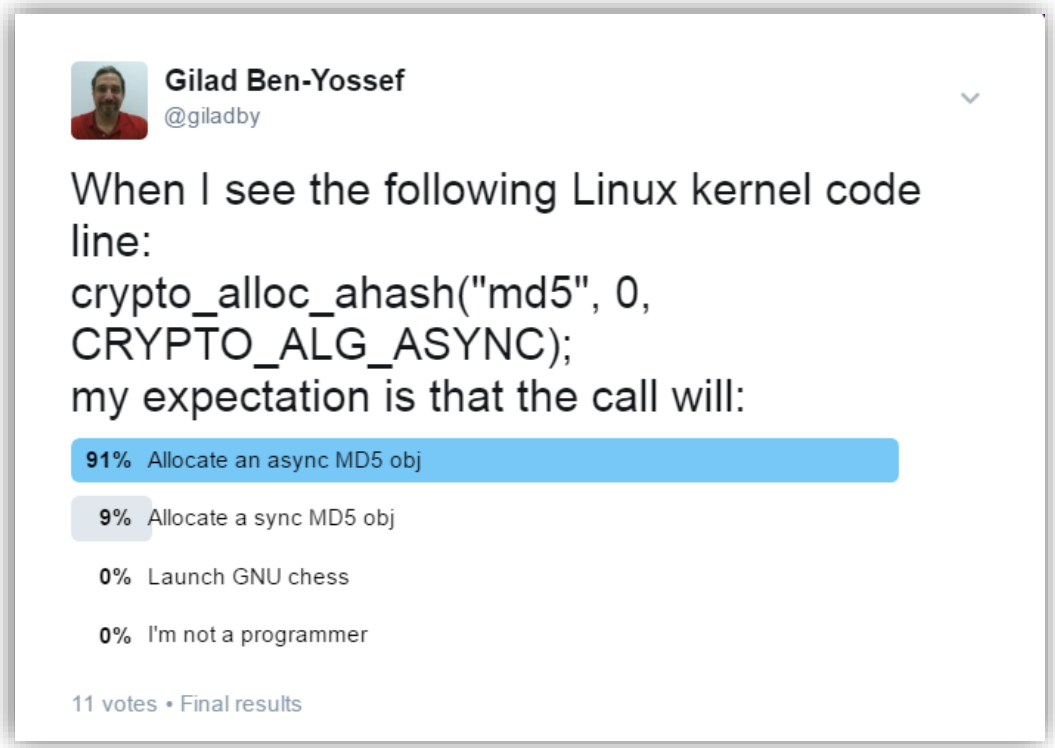
Patch accepted for 4.12.

```
From 785979d093200b658f7700ce8a1dbb6543c7675a Mon Sep 17 00:00:00 2001
From: Gilad Ben-Yossef <gilad@benyossef.com>
Date: Sun, 15 Jan 2017 12:28:44 +0200
Subject: [PATCH v3 RESEND] dm: switch dm-verity to async hash crypto API
MIME-Version: 1.0
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit

Use of the synchronous digest API limits dm-verity to using pure
CPU based algorithm providers and rules out the use of off CPU
algorithm providers which are normally asynchronous by nature,
potentially freeing CPU cycles.

This can reduce performance per Watt in situations such as during
boot time when a lot of concurrent file accesses are made to the
protected volume.

Move DM_VERITY to the asynchronous hash API.

Signed-off-by: Gilad Ben-Yossef <gilad@benyossef.com>
Tested-by: Milan Broz <gmazyland@gmail.com>
CC: Eric Biggers <ebiggers3@gmail.com>
CC: Ondrej Mosnáček <omosnacek+linux-crypto@gmail.com>
---

Resending since I did not get any feedback beyond Milan confirmation that
the latest version passes his simple sanity test.

The patch was tested by me on an Armv7 based dual core Zynq ZC706 development
board with SHA256-asm, SHA256-neon synchronous providers with no visible
degradation of performance, with cryptd based asynchronous versions of the
same and with an off tree Arm CryptoCell asynchronous provider.

Changes from v2:

- Use completion to potentially wait also on crypto_ahash_init() as it
  may finish asynchronously as well in some drivers, such as cryptd, as
```


arm

But how did we get here?

crypto_alloc_ahash(“md5”, 0, CRYPTO_ALG_ASYNC);



arm

Rusty Russel's API Design Manifesto

http://ozlabs.org/~rusty/index.cgi/tech/2008-03-30.html

How Do I Make This Hard to Misuse?

10. It's impossible to get wrong.
9. The compiler/linker won't let you get it wrong.
8. The compiler will warn if you get it wrong.
7. The obvious use is (probably) the correct one.
6. The name tells you how to use it.
5. Do it right or it will always break at runtime.
4. Follow common convention and you'll get it right.
3. Read the documentation and you'll get it right.
2. Read the implementation and you'll get it right.
1. Read the correct mailing list thread and you'll get it right.

What If I Don't Actually Like My Users?

-1. Read the mailing list thread and you'll get it wrong.
-2. Read the implementation and you'll get it wrong.
-3. Read the documentation and you'll get it wrong.
-4. Follow common convention and you'll get it wrong.
-5. Do it right and it will sometimes break at runtime.
-6. The name tells you how not to use it.
-7. The obvious use is wrong.
-8. The compiler will warn if you get it right.
-9. The compiler/linker won't let you get it right.
-10. It's impossible to get right.

arm

Symmetric cryptography code example

```c
struct tcrypt_result {
    struct completion completion;
    int err;
};

/* tie all data structures together */
struct skcipher_def {
    struct scatterlist sg;
    struct crypto_skcipher *tfm;
    struct skcipher_request *req;
    struct tcrypt_result result;
};

/* Callback function */
static void test_skcipher_cb(struct crypto_async_request *req, int error)
{
    struct tcrypt_result *result = req->data;

    if (error == -EINPROGRESS)
        return;
    result->err = error;
    complete(&result->completion);
    pr_info("Encryption finished successfully\n");
}
```

```c
/* Perform cipher operation */
static unsigned int test_skcipher_encdec(struct skcipher_def *sk,
                     int enc)
{
    int rc = 0;

    if (enc)
        rc = crypto_skcipher_encrypt(sk->req);
    else
        rc = crypto_skcipher_decrypt(sk->req);

    switch (rc) {
    case 0:
        break;
    case -EINPROGRESS:
    case -EBUSY:
        rc = wait_for_completion_interruptible(
            &sk->result.completion);
        if (!rc && !sk->result.err) {
            reinit_completion(&sk->result.completion);
            break;
        }
    default:
        pr_info("skcipher encrypt returned with %d result %d\n",
            rc, sk->result.err);
        break;
    }
    init_completion(&sk->result.completion);

    return rc;
}
```


arm

The good, the bad and the ugly

```c
struct tcrypt_result {
    struct completion completion;
    int err;
};

/* tie all data structures together */
struct skcipher_def {
    struct scatterlist sg;
    struct crypto_skcipher *tfm;
    struct skcipher_request *req;
    struct tcrypt_result result;
};

/* Callback function */
static void test_skcipher_cb(struct crypto_async_request *req, int error)
{
    struct tcrypt_result *result = req->data;

    if (error == -EINPROGRESS)
        return;
    result->err = error;
    complete(&result->completion);
    pr_info("Encryption finished successfully\n");
}
```

Boilerplate

```c
/* Perform cipher operation */
static unsigned int test_skcipher_encdec(struct skcipher_def *sk,
                                         int enc)
{
    int rc = 0;

    if (enc)
        rc = crypto_skcipher_encrypt(sk->req);
    else
        rc = crypto_skcipher_decrypt(sk->req);

    switch (rc) {
    case 0:
        break;
    case -EINPROGRESS:
    case -EBUSY:
        rc = wait_for_completion_interruptible(
            &sk->result.completion);
        if (!rc && !sk->result.err) {
            reinit_completion(&sk->result.completion);
            break;
        }
    default:
        pr_info("skcipher encrypt returned with %d result %d\n",
            rc, sk->result.err);
        break;
    }
    init_completion(&sk->result.completion);

    return rc;
}
```

-3. Read the documentation and you'll get it wrong.

arm

Symmetric cryptography code example – after change

```c
/* tie all data structures together */
struct skcipher_def {
    struct scatterlist sg;
    struct crypto_skcipher *tfm;
    struct skcipher_request *req;
    struct crypto_wait wait;
};
```

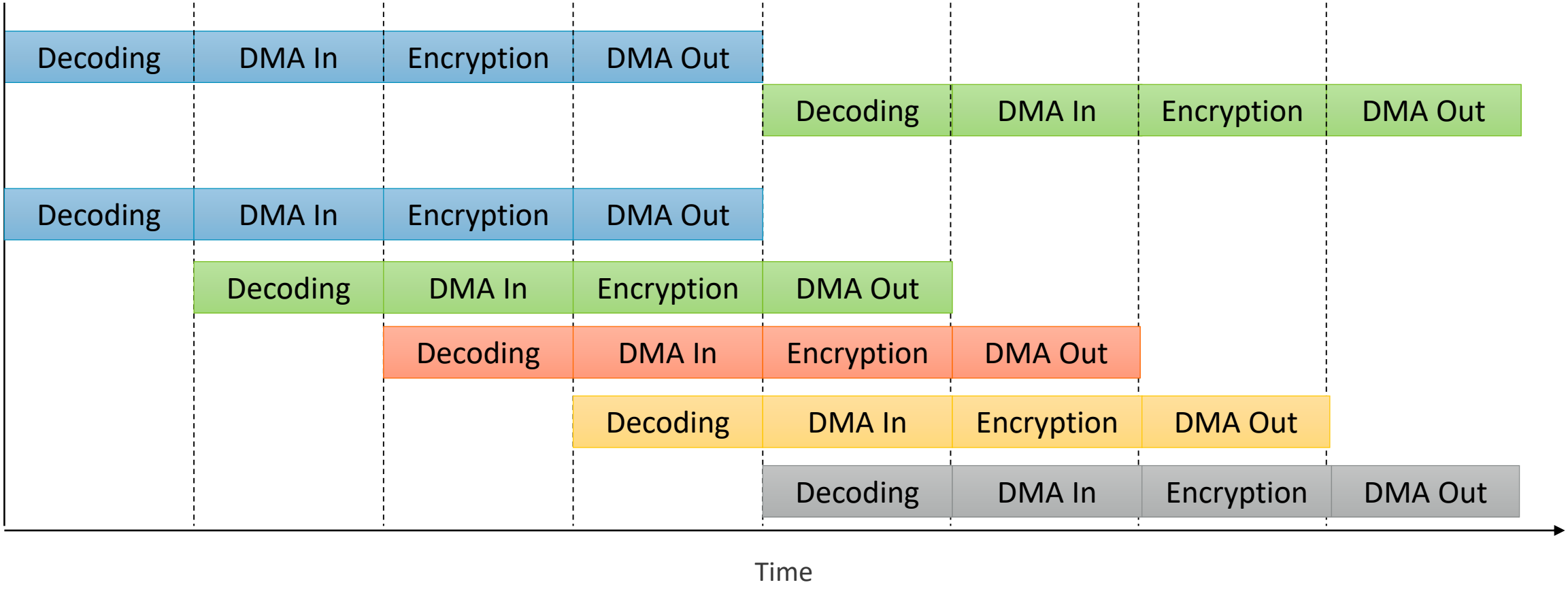
```c
/* Perform cipher operation */
static unsigned int test_skcipher_encdec(struct skcipher_def *sk,
                     int enc)
{
    int rc = 0;

    if (enc)
        rc = crypto_skcipher_encrypt(sk->req);
    else
        rc = crypto_skcipher_decrypt(sk->req);

    rc = crypto_wait_req(sk->req, rc);

    if (rc)
        pr_info("skcipher encrypt returned with %d result %d\n",
            rc, sk->result.err);

    crypto_init_wait(&sk->wait);

    return rc;
}
```

36 files changed, 310 insertions(+), 737 deletions(-)

arm

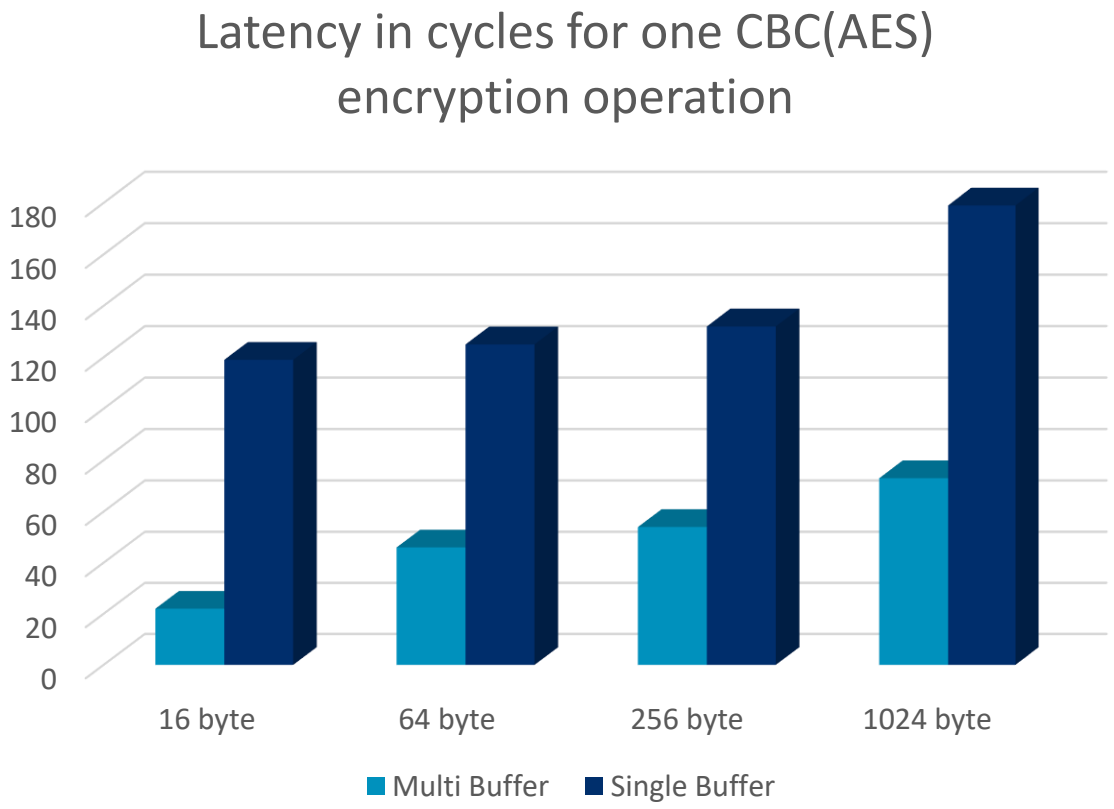
Latency hiding via parallelism

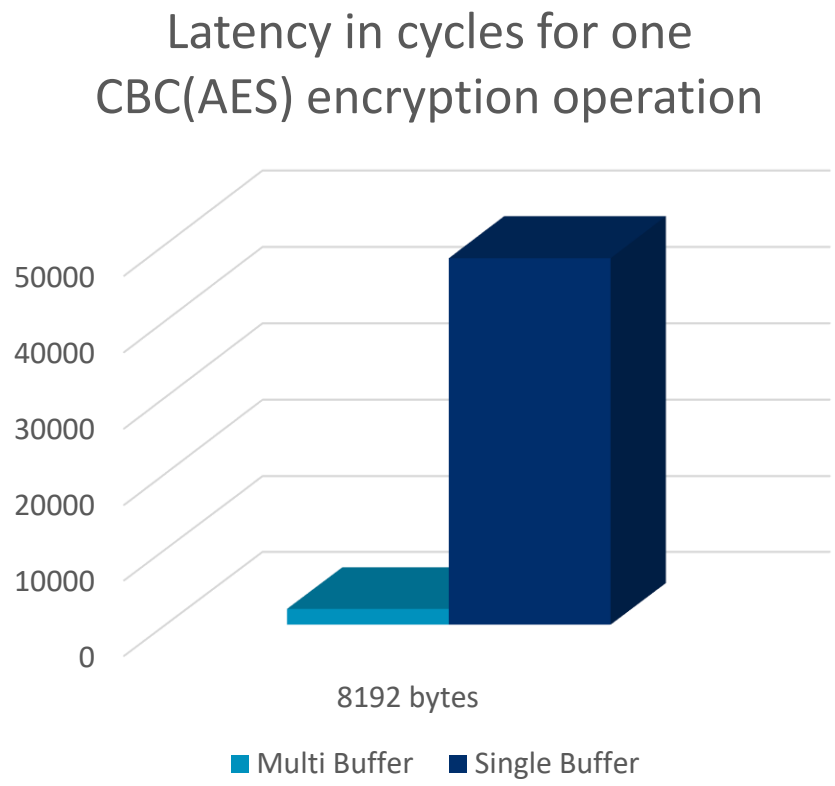
"Hardware is parallel"



arm

Effect of parallelism on latency





arm

Parallelism status in-kernel crypto users

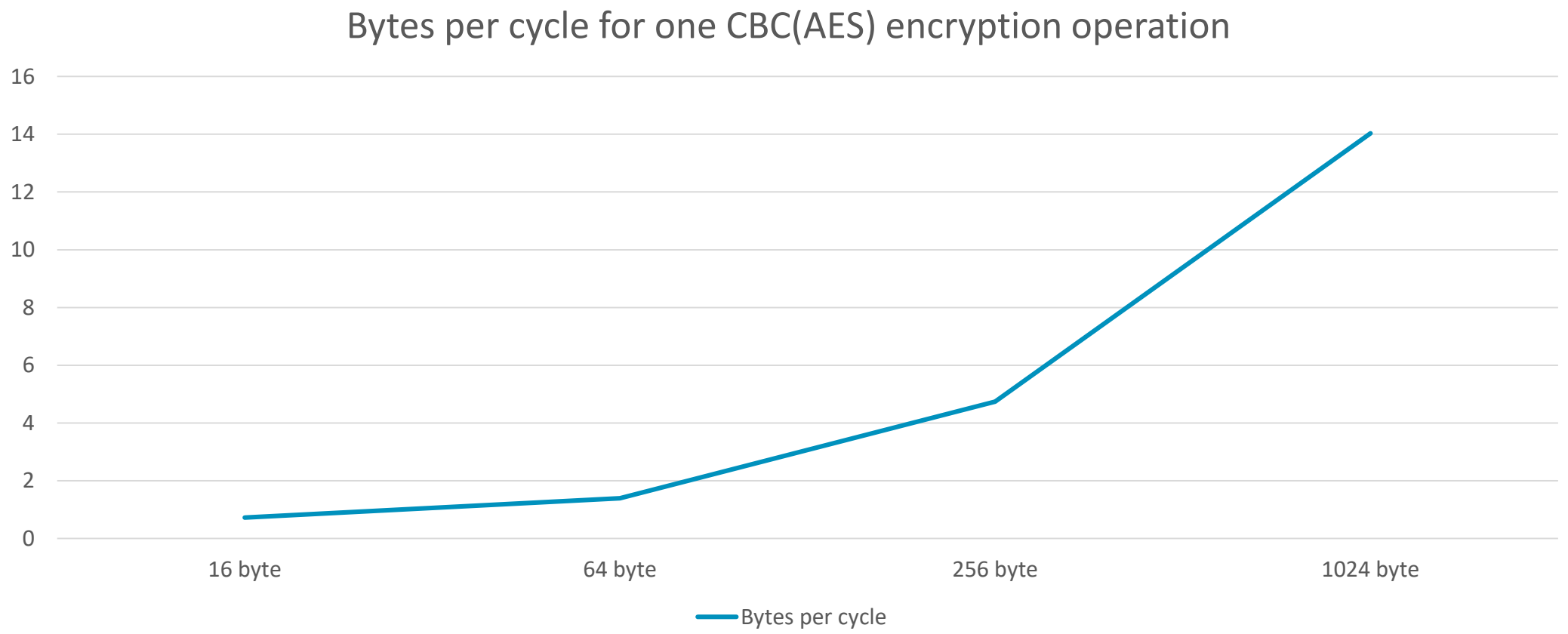
Already doing a very good job of parallelism:

- DM-Crypt
- IPsec code

Working on improving:

- DM-Verity
- Fscrypt
- Probably more

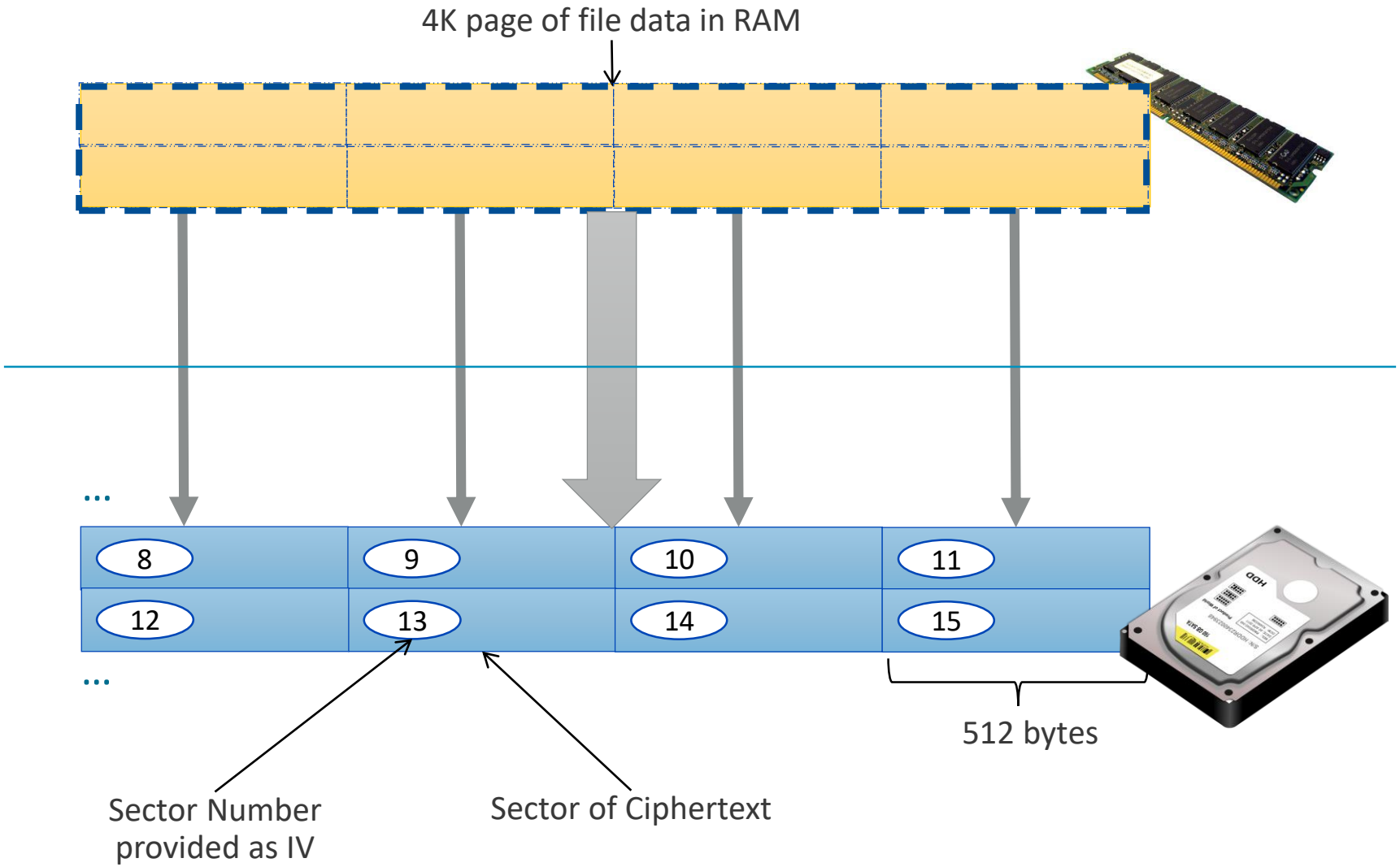
Effect of buffer sizes on latency



arm

DM-Crypt XTS/ESSIV generation handling

- DM-Crypt (and fscrypt) implements IV generation internally.
- Two separate different implementations.
- Limiting ability to use hardware AES/XTS acceleration for storage.
- Working with Binoy Jayan from Linaro on separating IV code.



arm

Sometime a simple solution is better

```
commit 8f0009a225171cc1b76a6b443de5137b26e1374b
Author: Milan Broz <gmazyland@gmail.com>
Date:   Thu Mar 16 15:39:44 2017 +0100

    dm crypt: optionally support larger encryption sector size

    Add  optional "sector_size"  parameter that specifies encryption sector
    size (atomic unit of block device encryption).

    Parameter can be in range 512 - 4096 bytes and must be power of two.
    For compatibility reasons, the maximal IO must fit into the page limit,
    so the limit is set to the minimal page size possible (4096 bytes).

    NOTE: this device cannot yet be handled by cryptsetup if this parameter
    is set.

    IV for the sector is calculated from the 512 bytes sector offset unless
    the iv_large_sectors option is used.

    Test script using dmsetup:

      DEV="/dev/sdb"
      DEV_SIZE=$(blockdev --getsz $DEV)
      KEY="9c1185a5c5e9fc54612808977ee8f548b2258d31ddadef707ba62c166051b9e3cd0294c27515f2bccee924e8823ca6e124b8fc3167ed478bca702babe4e130ac"
      BLOCK_SIZE=4096

      # dmsetup create test_crypt --table "0 $DEV_SIZE crypt aes-xts-plain64 $KEY 0 $DEV 0 1 sector_size:$BLOCK_SIZE"
      # dmsetup table --showkeys test_crypt

    Signed-off-by: Milan Broz <gmazyland@gmail.com>
    Signed-off-by: Mike Snitzer <snitzer@redhat.com>
```


arm

Hardware bound/sealed encryption keys

Many kind of security hardware supports hardware bound/sealed keys.

Hardware bound/sealed keys can only be accessed by the specific hardware security device that sealed them.

Hardware bound/sealed keys may be:

- Randomly generated by the hardware
- Provided by the user sealed and can only be unsealed by the hardware
- Preset secrets.

We already have support for this for asymmetric keys in the TPM sub-system.

However, this is not integrated with the crypto sub-system and does not cover symmetric ciphers.

FIPS error events

Currently FIPS errors are only triggered by the kernel in response to a cryptography verification test failure, with the immediate response of a panic.

However, FIPS errors may be generated in run-time by and reported to other sources. E.g.:

- A cryptographic module detecting run-time failure.
- Physical access to case.
- Encryption code error on Trusted Execution Environment or co-processor.

Perhaps a notification chain mechanism may be an appropriate mechanism here.

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!