Utilizing the crypto accelerators

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

Let's do crypto in userland!

There are two ways of accessing crypto functionality:

- Do it yourself
- Use a library (OpenSSL, GnuTLS...)

There are three ways of providing crypto functionality:

- Use pure software
- Use CPU instructions
- Use dedicated crypto device

DIY crypto in software

Software implementation of crypto algorithms:

+ Full control over the algorithm
+ No black box
+ Available uniformly everywhere
– Consumes lot of CPU cycles
– Is often slower than dedicated hardware

* Example: OpenSSL SW engine

DIY crypto using CPU instructions

CPU implementation of crypto algorithms:

+ Does consume minor amount of CPU cycles
+ Is faster than software counterpart
+ Does not suffer from bus overhead
- Implements subset of algorithms
- Black box in the CPU
- Not available everywhere uniformly

* Example: VIA PadLock, Intel AES-NI

DIY crypto using dedicated crypto hardware

Hardware implementation of crypto algorithms:

+ Does consume minor amount of CPU cycles
+ Is often faster than software counterpart
- Suffers from bus overhead
- Implements subset of algorithms
- Black box in the hardware
- Not available everywhere uniformly

* Example: Marvell CESA, Freescale DCP
* Noteworthy: Altera SoC-FPGA / Xilinx Zynq

Summary

	SW	CPU	HW
Arbitrary alg support	+	−	−
No black box	+	−	−
CPU cycles friendly	−	+	+
Fast on small blocks	+	+	−
Fast on large blocks	−	+	+
Broadly available	+	−	−

Accessing hardware crypto

Linux provides two means:

- Cryptodev
- AF_ALG

Cryptodev intro

* Supports CIPHER, HASH, AEAD
* Uses character device interface

+ Compatible with OpenBSD /dev/crypto
+ API compatible, not OpenBSD code
+ OpenSSL has engine for cryptodev
+ GnuTLS has support for cryptodev
+ Has nice examples
+ Lower latency

– Out of kernel tree code (for years)
– Adds arbitrary IOCTLs
– Synchronous only

Cryptodev getting started

It's quite easy:

- Download latest version from http://cryptodev-linux.org
- Extract cryptodev-linux-*.tar.gz
- Compile:

```
$ make
```

- Insmod:

```
$ insmod cryptodev.ko
```

The `/dev/crypto` pops up or kernel crashes at this point.
Building an included demo:

- Compile:

```
$ gcc -I. -o sha examples/sha.c
```

- Execute:

```
./sha
```

Cryptodev howto

Cryptodev usage pattern:

- int cfd = open("/dev/crypto");
- Fill in common `struct cryptodev_ctx`
- Fill in `struct crypt_op`
- Pass `struct crypt_op` into kernel via ioctl()
- Retrieve results
- close(cfd);

Cryptodev caveats

- Copying of data between userland and kernel
- Lots of context switches needed
- Unfit for asynchronous accelerators
- Recently only bugfix releases with dropping frequency
- The /dev/crypto permissions assigned by udev

AF_ALG intro

* Supports CIPHER, HASH
* Socket-based interface

\+ In-kernel code for years
\+ Inherently asynchronous
– OpenSSL has out-of-tree engine for AF_ALG
– GnuTLS does not have support for AF_ALG
– Not many examples
– Higher latency

AF_ALG prerequisites

- CONFIG_CRYPTO_USER_API=m
- CONFIG_CRYPTO_USER_API_HASH=m
- CONFIG_CRYPTO_USER_API_SKCIPHER=m

AF_ALG hashing demo

It's really easy, let's do a demo:

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <linux/if_alg.h>
#include <linux/socket.h>
#define SHA256_DIGEST_SZ    32

int main(void)
{
    struct sockaddr_alg sa = {
     .salg_family = AF_ALG,
     .salg_type = "hash",
     .salg_name = "sha256"
    };
    unsigned char digest[SHA256_DIGEST_SZ];
    char *input = "Hello World!";
    int i, sockfd, fd;
```

AF_ALG hashing demo

```
    sockfd = socket(AF_ALG, SOCK_SEQPACKET, 0);
    bind(sockfd, (struct sockaddr *)&sa, sizeof(sa));
    fd = accept(sockfd, NULL, 0);

    write(fd, input, strlen(input));
    read(fd, digest, SHA256_DIGEST_SZ);

    close(fd);
    close(sockfd);

    for (i = 0; i < SHA256_DIGEST_SZ; i++)
        printf("%02x", digest[i]);
    printf("\n");

    return 0;
}
```

AF_ALG tips&tricks

- Use the `splice()` call to push data
- Page-align the buffers with payloads

Library plugins

Better use a library (if you're not Bruce Schneier):

- The OpenSSL already has support for /dev/crypto
- The OpenSSL needs easy plugin for AF_ALG

Testing OpenSSL with /dev/crypto

Check and/or enable /dev/crypto with OpenSSL

- Check: `$ openssl engine`
  Look for "(cryptodev)" engine.
- Check: `$ openssl version -f`
  Look for the -D... as below
- Enable:
```
./Configure ...\
        -DHAVE_CRYPTODEV \
        -DUSE_CRYPTODEV_DIGESTS
```

Test:

```
$ openssl speed -evp aes-128-cbc -engine cryptodev
```

! CAREFUL: `openssl speed` measures speed, not validity !

Using OpenSSL with /dev/crypto

In case you want to explicitly use cryptodev OpenSSL engine:

```
    ENGINE *e;

    ENGINE_load_builtin_engines();
    e = ENGINE_by_id("cryptodev");
    if (!e) {
        fprintf(stderr, "Error finding cryptodev engine!\n");
        return -EINVAL;
    }
    if (!ENGINE_set_default(e, ENGINE_METHOD_ALL)) {
        fprintf(stderr, "Cannot use cryptodev engine!\n");
        ret = -EINVAL;
        goto err;
    }
    /* Use engine here */
    ret = 0;
err:
    ENGINE_free(e);
    return ret;
```

Hint on cryptodev engine configuration

OpenSSL library allows engine configuration via `/etc/ssl/openssl.cnf` , see `config(5)`

→ `OPENSSL LIBRARY CONFIGURATION`

⇒ `ENGINE CONFIGURATION MODULE`

NOTE: You can use `OPENSSL_CONF` variable instead.

Example modification:

```
openssl_conf = conf_section
[ conf_section ]
engines = engine_section
...
[ engine_section ]
cryptodev = cryptodev_section
...
[ cryptodev_section ]
engine_id = cryptodev
default_algorithms = ALL
CIPHERS=aes-128-cbc
```

Using OpenSSL with AF_ALG

Things are a bit more involved as AF_ALG plugin is needed:

- Download plugin source here:
  `http://src.carnivore.it/users/common/af_alg/tree/`
- Follow README, in short:
  - Find where the OpenSSL keeps engine plugins:
    `$ echo ‘openssl version -d‘/engines`
  - Compile AF_ALG plugin:
    `$ make`
  - Copy the resulting library into engines/ dir:
    `$ cp libaf_alg.so ‘openssl version -d‘/engines/`

Using OpenSSL with AF_ALG

In case you want to explicitly use af_alg OpenSSL engine:

```
ENGINE *e;

ENGINE_load_builtin_engines();
e = ENGINE_by_id("af_alg");
/* ... */
ENGINE_free(e);
return ret;
```

Hint on AF_ALG engine configuration

Example modification of openssl.cnf :

```
openssl_conf = conf_section
[ conf_section ]
engines = engine_section
...
[ engine_section ]
af_alg = af_alg_section
...
[ af_alg_section ]
engine_id = af_alg
default_algorithms = ALL
CIPHERS=aes-128-cbc aes-192-cbc aes-256-cbc des-cbc
DIGESTS=md4 md5 sha1 sha224 sha256 sha512
```

Final thoughts

- For new projects, please use AF_ALG
- Help improve AF_ALG
  - Additional algo support (compress, RNG, . . .)
  - Documentation
  - Performance?
- Help improve OpenSSL
  - The code itself (LF is already helping!)
  - The AF_ALG engine plugin
- Help improve GnuTLS
  - The AF_ALG support
- Help with documentation
  - OpenSSL documentation
  - AF_ALG examples

Thank you for your attention!

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