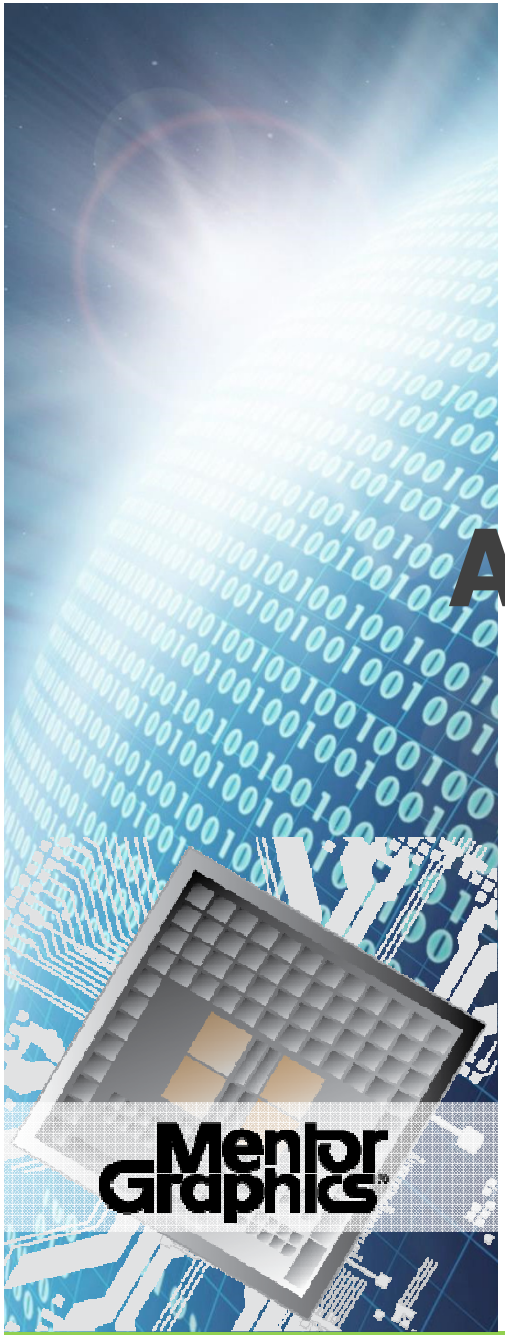



# Linux Cryptographic Acceleration on an i.MX6

Sean Hudson

Embedded Linux Architect

mentor embedded

mentor.com/embedded

# Who am I?

- I am an embedded Linux architect and Member of Technical Staff at Mentor Graphics. I have worked on embedded devices since 1996. I started working with Linux as a hobbyist in 1999 and professionally with embedded Linux in 2006. In OSS, I have been involved with the Yocto Project since it's public announcement in 2010, have served on the YP Advisory Board for two different companies, and am currently a member of the OpenEmbedded Board.

# A special word of thanks

- Much of the hard work gathering data for this presentation was done by a co-worker, Wade Farnsworth, who was unable to attend today.

# Outline

- About this presentation
- A word about Cryptography
- Crypto Hardware types
- i.MX6 CAAM
- Kernel access to HW
- Application access to HW
- Test methods
- Results
- Conclusions and final thoughts
- Q&A / Discussion

# About this presentation

- This talk is geared towards using cryptographic hardware acceleration from user space
  - I won't spend a lot of time on kernel internals and drivers
- The focus comes from work that we did with the i.MX6
- However, much of this is generally applicable

# A word about Cryptography

- In terms of goals, cryptography is pretty simple ☺
  - Send a message from one point to another without someone in the middle being able to read it in a reasonably short, amount of time
- Most cryptographic algorithms rely on asymmetric, computational complexity to guarantee security
  - Brute force attacks should be infeasible in **reasonable** amounts of time
  - Encryption/decryption should be **relatively** cheap
- In a related way, as the need for computational complexity has increased, the time to encrypt/decrypt has also increased, hence the desire for hardware acceleration

# Some more words about Cryptography

- Basic encryption usually requires a couple of things:
  - A strong algorithm, e.g. AES
  - A strong key from a large key space, e.g. random number
- Basic encryption enables several, additional features:
  - Tamper detection
  - Secure storage
  - Key exchange
  - Secure identification/authorization
  - Secure execution
  - …
- Strong Cryptography != Strong Security

# Types of Crypto Hardware

- Standalone
  - E.g. Smartcards
- Instruction set extensions
  - Built into primary CPU
  - E.g. Via Padlock & Intel AES-NI
- Separate co-processors
  - Different interconnect flavors
    - Separate processors connected by an external bus
      - Trusted Computing Module (TPM)
        - Standard for a separate, specialized processor used to accelerate
        - Trusted Computing Group (TCG) manages the standard
        - Found in x86 platforms
      - Offload processors
        - PCIE cards, for example
    - Part of an SOC
      - i.MX6 crypto block

# i.MX6 Crypto Hardware

- The NXP i.MX6 SoC includes a cryptographic acceleration and assurance module (CAAM) block, which provides cryptographic acceleration and offloading hardware.
- The CAAM provides :
  - HW implementation of cryptographic functions
    - Includes several ciphers and hashing algorithms
  - Secure memory
  - Secure key module
  - Cryptographic authentication
  - Random-number generation

# Enabling CAAM in the 4.1 kernel

- The kernel should have the following options enabled in order to access the CAAM module:
  - CONFIG_CRYPTO_DEV_FSL_CAAM=y
  - CONFIG_CRYPTO_DEV_FSL_CAAM_JR=y
  - CONFIG_CRYPTO_DEV_FSL_CAAM_RINGSIZE=9
  - # CONFIG_CRYPTO_DEV_FSL_CAAM_INTC is not set
  - CONFIG_CRYPTO_DEV_FSL_CAAM_CRYPTO_API=y
  - CONFIG_CRYPTO_DEV_FSL_CAAM_AHASH_API=y
  - CONFIG_CRYPTO_DEV_FSL_CAAM_RNG_API=y
  - # CONFIG_CRYPTO_DEV_FSL_CAAM_RNG_TEST is not set
  - CONFIG_CRYPTO_DEV_FSL_CAAM_SM=y
  - CONFIG_CRYPTO_DEV_FSL_CAAM_SM_SLOTSIZE=7
  - # CONFIG_CRYPTO_DEV_FSL_CAAM_SM_TEST is not set
  - CONFIG_CRYPTO_DEV_FSL_CAAM_SECVIO=y
  - # CONFIG_CRYPTO_DEV_FSL_CAAM_DEBUG is not set

# Cryptography in Userspace

- Pure SW implementation
  - Portable & supports arbitrary algorithms
  - Costs CPU cycles
  - CPUs aren't optimized for this work
- Use CPU instruction extensions
  - Makes use of HW acceleration
  - Doesn't involve the kernel
  - Limited to algs that are support by HW, e.g. AES
- Kernel APIs for userspace

# Crypto APIs in the kernel

- Since 2.5.45, the kernel has had a cryptographic framework
  - Used internally for things like IPSEC
- There are two userspace interfaces that provide access to that API
  - Cryptodev (/dev/crypto)
  - AF_ALG
- The userspace APIs provide HW abstraction

# Cryptodev

- API compatible with OpenBSD Cryptographic Framework (OCF) or /dev/crypto
- Cryptodev creates a /dev/crypto device
- Uses standard ioctls to interface with the kernel crypto subsystem

# Enabling the cryptodev module

- cryptodev is implemented as an out-of-kernel module, and therefore must be compiled against the i.MX6 kernel.
- In poky, this is as simple as adding the following to local.conf:
  - CORE_IMAGE_EXTRA_INSTALL = "cryptodev-module"

# AF_ALG

- AF_ALG uses sockets to interface with the kernel
- It is supported in mainline Linux (no external module compile), but requires additional kernel config options

# Configuring the kernel for AF_ALG

- AF_ALG requires the following kernel options to be enabled:

  CONFIG_CRYPTO_USER_API=y

  CONFIG_CRYPTO_USER_API_HASH=y

  CONFIG_CRYPTO_USER_API_SKCIPHER=y

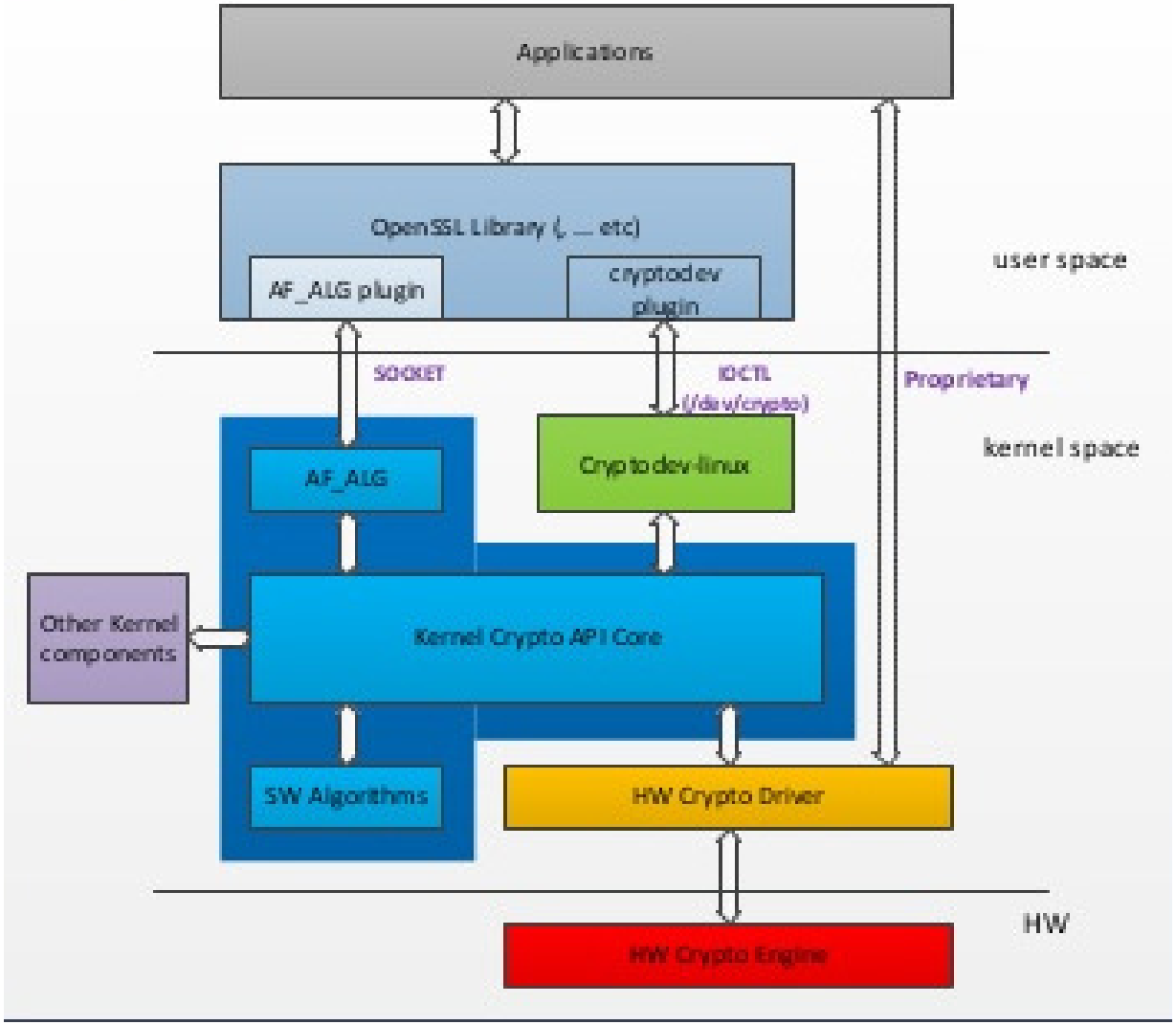
  CONFIG_CRYPTO_USER_API_RNG=y

  CONFIG_CRYPTO_USER_API_AEAD=y

# OpenSSL

- Instead of accessing crypto functions directly via CPU instructions or the kernel APIs, we opted to use the OpenSSL library to wrap that functionality for us
- There are a few steps to enable OpenSSL for each kernel API though (more on that in a bit)

# A Pretty Picture




This picture was found here:
https://image.slidesharecdn.com/slideshare-linuxcrypto-160411115704/95/slideshare-linux-crypto-4-638.jpg?cb=1460375879

# OpenSSL with cryptodev

- In order to support cryptodev, OpenSSL must be built with the following compiler options:
  - -DHAVE_CRYPTODEV -DUSE_CRYPTODEV_DIGESTS
- Additionally, the sysroot should have the cryptodev header installed: usr/include/crypto/cryptodev.h
- In poky, the OpenSSL recipe enables these by default (the header is installed via a DEPENDS on cryptodev-linux).

# OpenSSL with cryptodev (2)

- When running OpenSSL, it is important to make sure that you have the cryptodev module inserted first. After this is inserted, you should see the /dev/crypto node become available, and OpenSSL should report it as an available engine:
- Example:

```
root@mx6q-csp:~# openssl engine
    (cryptodev) BSD cryptodev engine
    (dynamic) Dynamic engine loading support
```

# OpenSSL with AF_ALG

- OpenSSL 1.0.2 does not support AF_ALG natively yet
- A plugin must be used to interface with the kernel
- For these tests, we used the af_alg plugin located here: https://github.com/sarnold/af_alg
- Native support for AF_ALG will be available starting in OpenSSL 1.1.0.

# OpenSSL with AF_ALG (2)

- The plugin should be built as described in the plugin's documentation.
- The resulting library must be placed in /usr/lib/engines
- /etc/ssl/openssl.cnf must contain the following:

```
openssl_conf = openssl_def

[openssl_def]
engines = openssl_engines

[openssl_engines]
af_alg = af_alg_engine

[af_alg_engine]
default_algorithms = ALL
CIPHERS=aes-128-cbc aes-192-cbc aes-256-cbc des-cbc des-ede3-cbc
DIGESTS=md4 md5 sha1 sha224 sha256 sha512
```

# Comparing performance

- We used the OpenSSL speed command to measure performance
- The "-elapsed" argument is used so that the throughput measurements are against wall clock time, rather than user CPU time

# Test Run (SW implementation)

- Example:

```
root@mx6q-csp:~# time -v openssl speed -evp aes-128-cbc -elapsed
You have chosen to measure elapsed time instead of user CPU time.
Doing aes-128-cbc for 3s on 16 size blocks: 5591286 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 64 size blocks: 1570038 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 256 size blocks: 405662 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 1024 size blocks: 102273 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 8192 size blocks: 12812 aes-128-cbc's in 3.00s
<...snip...>
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128-cbc      29820.19k    33494.14k    34616.49k    34909.18k    34985.30k
```

# Test Run (cryptodev)

- Pass OpenSSL the "-engine cryptodev" argument to offload supported cryptographic algorithms to the CAAM
- Example:

```
root@mx6q-csp:~# openssl speed -evp aes-128-cbc -engine cryptodev
engine "cryptodev" set.
Doing aes-128-cbc for 3s on 16 size blocks: 43298 aes-128-cbc's in 0.09s
Doing aes-128-cbc for 3s on 64 size blocks: 42467 aes-128-cbc's in 0.06s
Doing aes-128-cbc for 3s on 256 size blocks: 36657 aes-128-cbc's in 0.07s
Doing aes-128-cbc for 3s on 1024 size blocks: 26992 aes-128-cbc's in 0.03s
Doing aes-128-cbc for 3s on 8192 size blocks: 8101 aes-128-cbc's in 0.00s
<...snip...>
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128-cbc       7697.42k    45298.13k   134059.89k   921326.93k        infk
```

- You can confirm that the CAAM is being used by checking to see if CAAM interrupts are increasing:

```
root@mx6q-csp:~# cat /proc/interrupts | grep jr1
 311:    2168629          0          0          0      GIC 138 Level     2102000.jr1
```

# Test Run (AF_ALG)

- Pass OpenSSL the "-engine af_alg" argument to offload supported cryptographic algorithms to the CAAM
- Example:

```
root@mx6q-csp:/etc/ssl# openssl speed -evp aes-128-cbc -engine af_alg
```

***engine "af_alg" set.***

*Doing aes-128-cbc for 3s on 16 size blocks: 39792 aes-128-cbc's in 0.08s*
*Doing aes-128-cbc for 3s on 64 size blocks: 40170 aes-128-cbc's in 0.09s*
*Doing aes-128-cbc for 3s on 256 size blocks: 33830 aes-128-cbc's in 0.08s*
*Doing aes-128-cbc for 3s on 1024 size blocks: 26698 aes-128-cbc's in 0.05s*
*Doing aes-128-cbc for 3s on 8192 size blocks: 7248 aes-128-cbc's in 0.02s*
*<...snip...>*
*The 'numbers' are in 1000s of bytes per second processed.*

| *type* | *16 bytes* | *64 bytes* | *256 bytes* | *1024 bytes* | *8192 bytes* |
|---|---|---|---|---|---|
| *aes-128-cbc* | *7958.40k* | *28565.33k* | *108256.00k* | *546775.04k* | *2968780.80k* |

# Summary of results

| aes-128-cbc | Number of blocks processed in 3s / Block Size in Bytes | | | | |
|---|---|---|---|---|---|
| | 16 | 64 | 256 | 1024 | 8192 |
| SW Implementation | 5591286 | 1570038 | 405662 | 102273 | 12812 |
| CryptoDev | 43298 | 42467 | 36657 | 26992 | 8101 |
| AF_ALG | 39792 | 40170 | 33830 | 26698 | 7248 |

# Conclusions

- In our case, SW implementation performed best?!
- Digging in further, we observed a drop in CPU utilization using the CAAM module
  - However, we also observed a significant number of context switches
- HW acceleration will not always yield *faster* results
- This was not an exhaustive analysis; make sure to run your own tests

# Q&A

# References


- http://www.linuxjournal.com/node/6451/print
- http://www.slideshare.net/nij05/slideshare-linux-crypto-60753522
- https://en.wikipedia.org/wiki/Cryptography
- http://williamstallings.com/Extras/Security-Notes/lectures/classical.html
- https://www.cl.cam.ac.uk/~mkb23/research/Survey.pdf
- http://www.logix.cz/michal/devel/padlock/
- https://software.intel.com/en-us/blogs/2012/01/11/aes-ni-in-laymens-terms
- https://en.wikipedia.org/wiki/Crypto_API_(Linux)
- https://www.openbsd.org/papers/ocf.pdf