mbed TLS
Adding HW Acceleration

ARM

Ron Eldor – Security Staff Application Engineer
Simon Butcher – mbed TLS Tech Lead
Janos Follath – Senior Security Engineer

Silicon Partner Workshop - Wyboston Lakes
March 2017

Agenda

- Short overview of cryptographic primitives
- mbed TLS as a modular library
- Directory structure overview
- Accelerating Symmetric functions
- Accelerating Asymmetric functions
- Testing
- Benchmarking
- Workshop session objectives

Crypto 101

Definition of Cryptography

From Wikipedia:

"Cryptography is the practice and study of techniques for secure communication in the presence of third parties called adversaries.

*More generally, cryptography is about constructing and analyzing protocols that prevent third parties or the public from reading private messages; various aspects in information security such as **data confidentiality, data integrity**, **authentication**, and non-repudiation are central to modern cryptography."*

Overview of Cryptography

- Data **confidentiality** is achieved when two (or more) entities want to share secret information. This is done either by symmetric algorithms, or asymmetric algorithm

- Data **integrity** is achieved when a text contains additional information confirming the data wasn't manipulated (a signature or digest), and an algorithm run on the text compares the output with the additional information given. Algorithms used for data integrity are digest and checksum algorithms and MAC algorithms

- Data **Authentication** is achieved when one entity needs to verify that the data was not modified AND the origin of the data is approved. This is done by symmetric and asymmetric algorithms

Symmetric algorithms

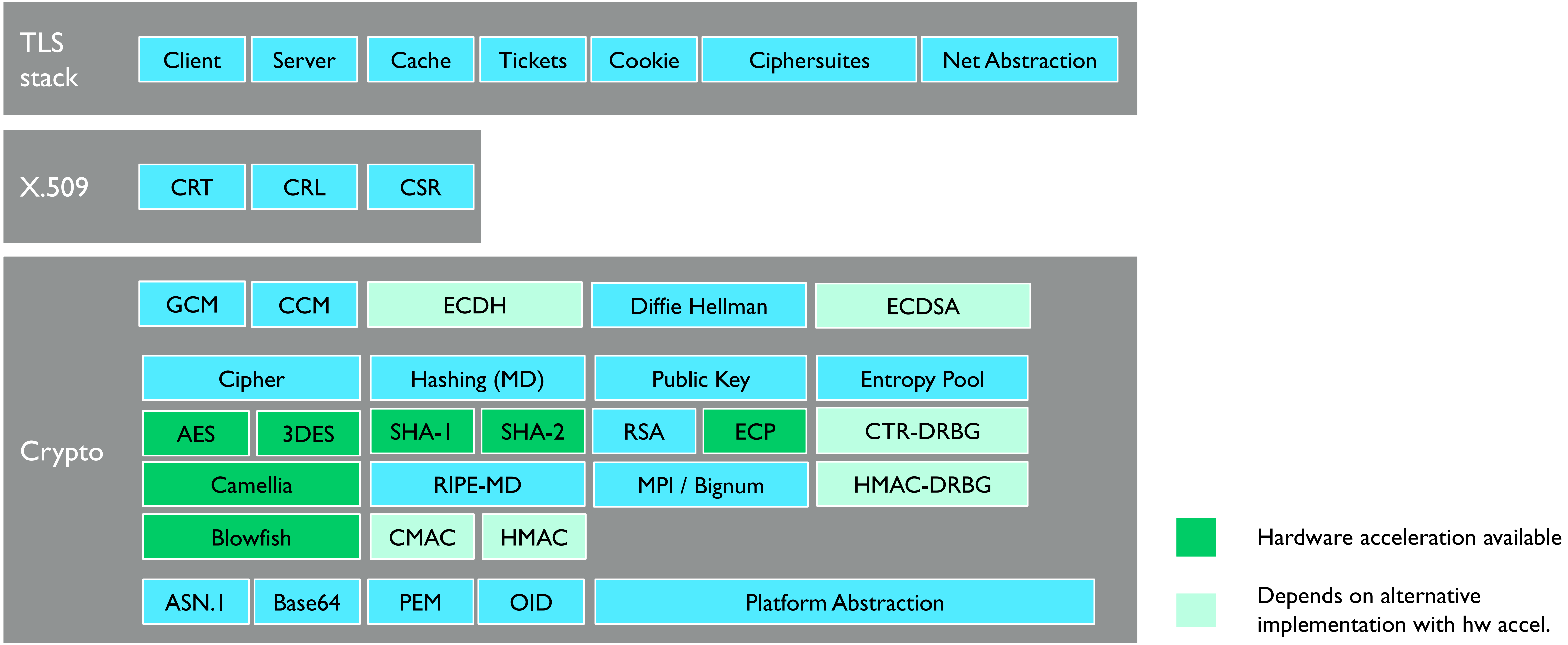
- Shared secret is known to both sides → Both sides are trusted
- Faster than asymmetric algorithms
- Used for authentication, ciphering, and signing data
- The shared secret (Key)
  - Has to be known in advance by both sides
  - Exchanged between two sides with a key exchange scheme

Asymmetric algorithms

- Key pair concept:
  - Private key – only known to one side
  - Public key – publicly distributed
- Applications:
  - Used for signing and verifying data (e.g. certificates)
  - Used for key exchange schemes

HW Acceleration with mbed TLS

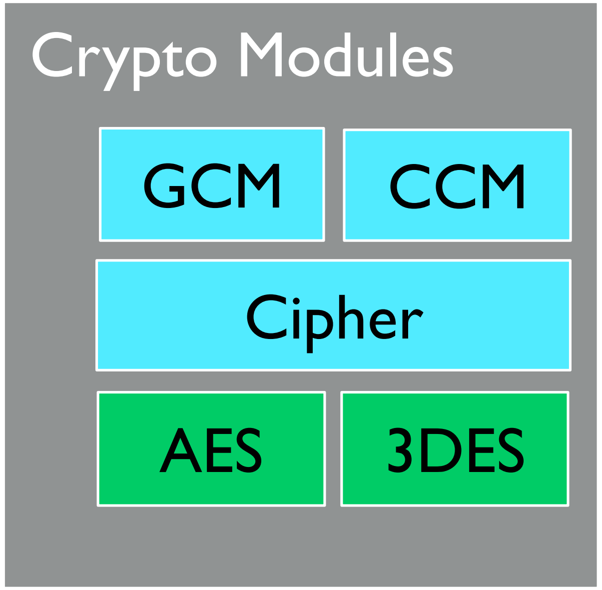
High-level mbed TLS structure



ARM

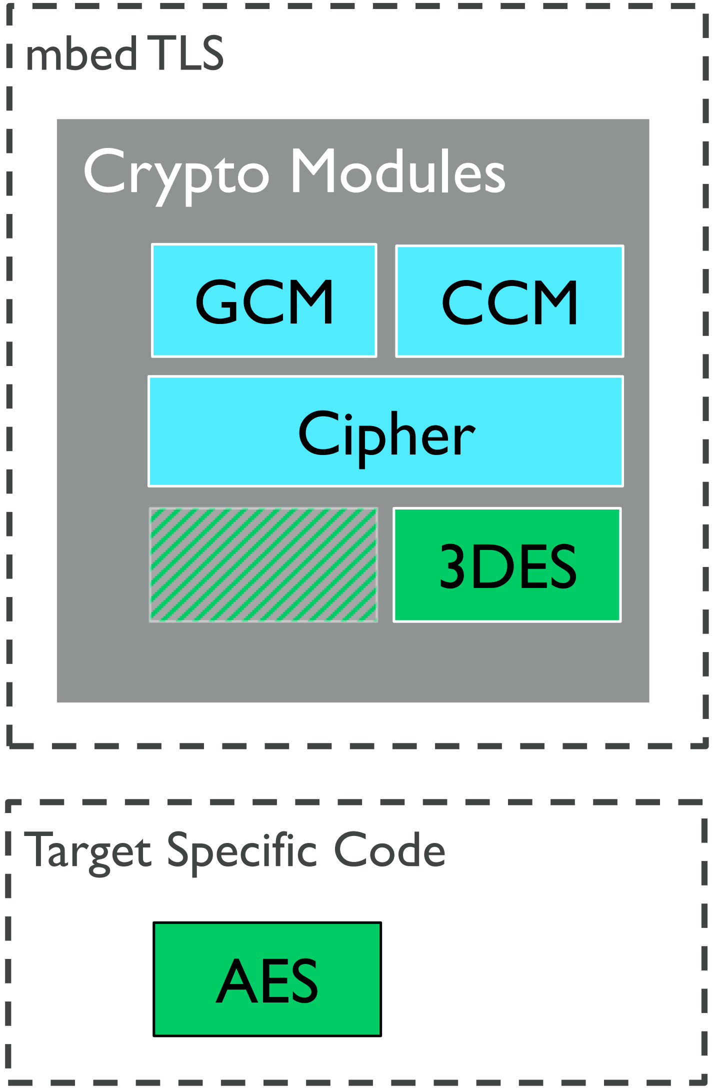
Modular Design

- Each component (Cipher, Public Key etc) can be included or excluded from the library dependent on the application's need
- The library is a library not a component and does not have it's own threads of execution
- Simple interface, to be synchronously called but capable of supporting multi-threading
- Highly configurable through callbacks and parameters



Hardware Acceleration Support

- Modular design means components can easily be swapped out and replaced
- This allows mbed Partners to replace a component with their own equivalent implementation, providing the same interface
- This is the mechanism used to provide hardware acceleration support
- Code can be swapped out by entire module or function



ARM

Structure of mbed TLS inside mbed OS

mbed-os / features / mbedtls

📁	importer	Script to import mbed TLS from it's own repo
📁	inc / mbedtls	Header files for the library
📁	platform	mbed OS Platform integration code
📁	src	mbed TLS library source files
📁	targets	Directory for Target specific mbed TLS code
📄	README.md	Readme
📄	VERSION.txt	mbed TLS Version file

Using alternative cryptographic implementations

- Supported symmetric algorithms:

  AES, ARC4, BLOWFISH, CAMELLIA, DES, XTEA, MD2, MD4, MD5, RIPEMD160, SHA1, SHA256, SHA512

- Supported asymmetric algorithms: ECDH, ECDSA (through the ECP module)

- See documentation:

  https://docs.mbed.com/docs/mbed-os-handbook/en/latest/advanced/tls_hardware_acceleration/

Using alternative cryptographic implementations

- It is possible to use alternative implementations for symmetric and asymmetric algorithms
- Add MBEDTLS_CONFIG_HW_SUPPORT
  mbed-os/targets/targets.json
- Add the relevant *_ALT (e.g. MBEDTLS_AES_ALT, MBEDTLS_SHA256_ALT etc..)
  mbed-os/features/mbedtls/targets/TARGET_XXXX/mbedtls_device.h

Using alternative cryptographic implementations

- Configuration is done in compile time, using precompiled definitions

mbed-os/features/mbedtls/src/aes.c:

```
#if !defined(MBEDTLS_AES_ALT)
```

mbed-os/features/mbedtls/targets/TARGET_XXXX/aes_alt.c:

```
#if defined(MBEDTLS_AES_ALT)
```


ARM

Using an alternative implementation

- For example, for the AES hardware acceleration
  - Add `#define MBEDTLS_AES_ALT` *to mbedtls_device.h*
  - Define AES API * and mbedtls_aes_context *in aes_alt.h*

```
#else  /* MBEDTLS_AES_ALT */
#include "aes_alt.h"
#endif /* MBEDTLS_AES_ALT */
```

  - Implement the alternative API

* The alternative implementation should follow the same function names and signatures as original

ARM

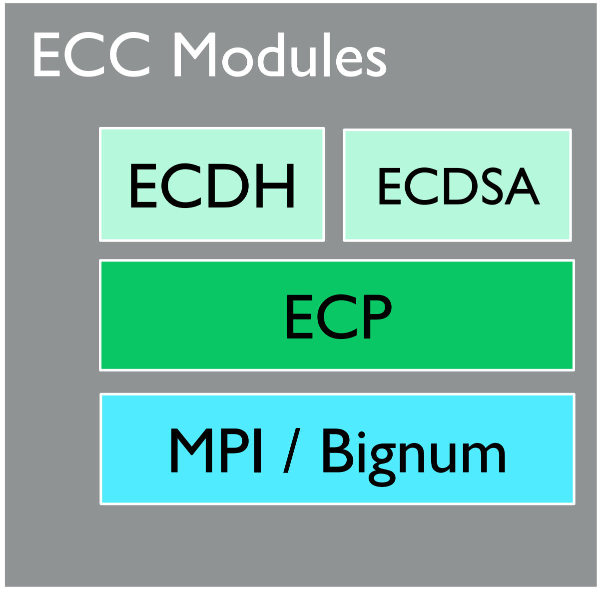
Example: AES module interface

```
void mbedtls_aes_init(…)
void mbedtls_aes_free(…)
int mbedtls_aes_setkey_enc(…)
int mbedtls_aes_setkey_dec(…)
int mbedtls_aes_crypt_ecb(…)

int mbedtls_aes_crypt_cbc(…)          // if defined(MBEDTLS_CIPHER_MODE_CBC)
int mbedtls_aes_crypt_ctr(…)          // if defined(MBEDTLS_CIPHER_MODE_CTR)

int mbedtls_aes_crypt_cfb128(…)    // if defined(MBEDTLS_CIPHER_MODE_CFB)
int mbedtls_aes_crypt_cfb8(…)      // if defined(MBEDTLS_CIPHER_MODE_CFB)
```

- The entire interface must always be implemented

Asymmetric Acceleration Design Approach

- Why the ECP module?
  - Supported hardware accelerators
  - Shared module: Bignum
- Reasons to expose higher level API
  - Context switches
- Reasons to expose lower level API
  - Implementation costs
  - Maintenance costs
- *Feedback welcome!*



Asymmetric implementation (ECP module)

- Add one of the following definitions to *mbedtls_device.h*
  - MBEDTLS_ECP_INTERNAL_ALT – to replace only partial APIs of Elliptic Curve Point calculations, supported by the target
  - MBEDTLS_ECP_ALT – to replace the full implementation of Elliptic curve point calculations. (in case specific function replacement is not achievable)
- Define ECP API and mbed TLS ECP structures in ecp_alt.h
- Implement the alternative API

Asymmetric implementation – cont.

- It is possible to implement only part of the API as HW accelerated, e.g. `mbedtls_internal_ecp_add_mixed`
- Add `MBEDTLS_ECP_ADD_MIXED_ALT` to macros in *mbedtls_device.h*
- Implement 'mbedtls_internal_ecp_add_mixed'
- Alternative implementation can be for specific curves: `mbedtls_internal_ecp_grp_capable`

Elliptic Curves

$$y^2 = x^3 + ax + b$$

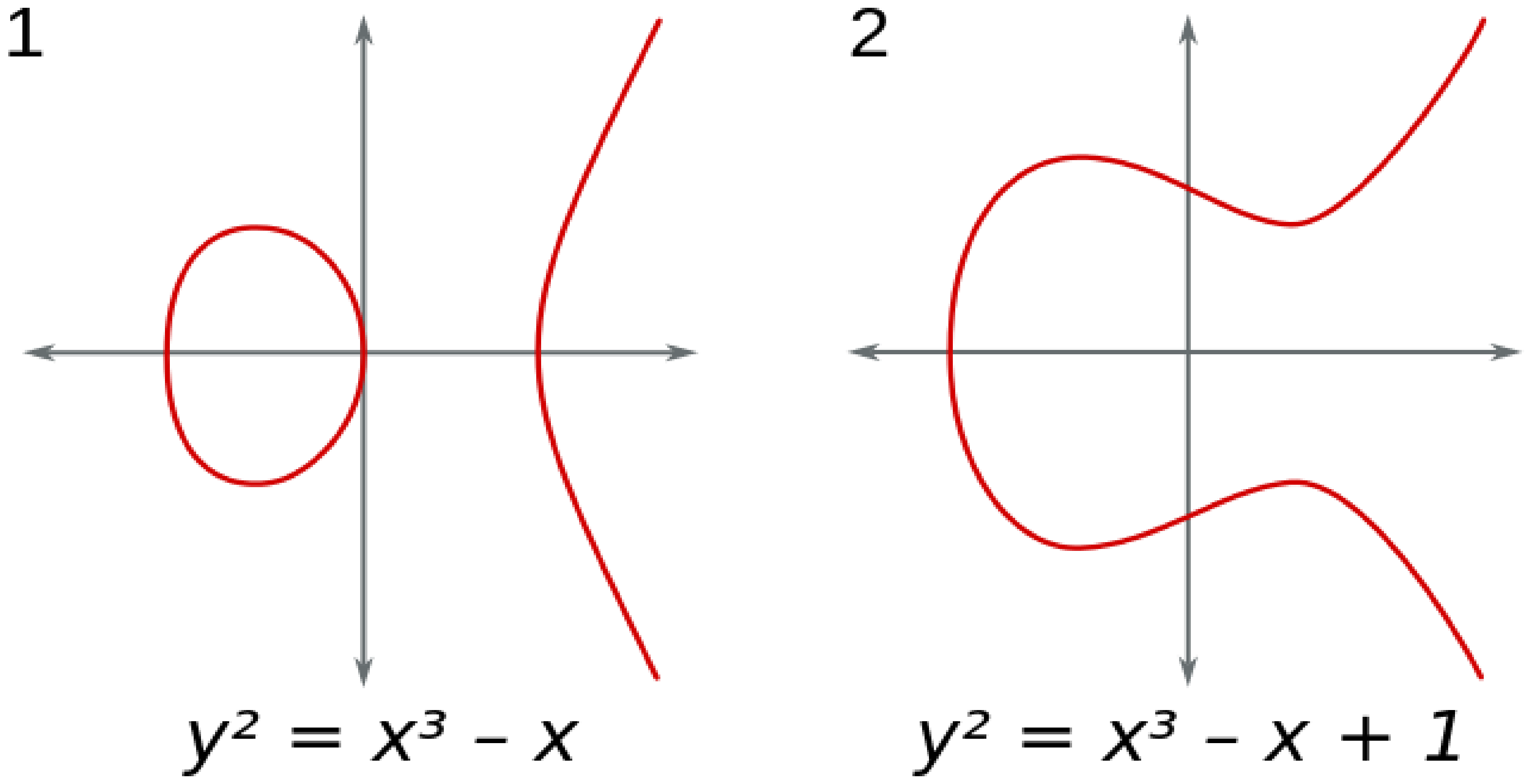


Figure taken from wikipedia

Exposed internal API (ecp_internal.h)

```
unsigned char mbedtls_internal_ecp_grp_capable(…)
int mbedtls_internal_ecp_init(…)
void mbedtls_internal_ecp_free(…)

// if defined(ECP_SHORTWEIERSTRASS)

int mbedtls_internal_ecp_randomize_jac(…)        // if defined(MBEDTLS_ECP_RANDOMIZE_JAC_ALT)

int mbedtls_internal_ecp_add_mixed(…)            // if defined(MBEDTLS_ECP_ADD_MIXED_ALT)

int mbedtls_internal_ecp_double_jac(…)           // if defined(MBEDTLS_ECP_DOUBLE_JAC_ALT)

int mbedtls_internal_ecp_normalize_jac_many(…)   // if defined(MBEDTLS_ECP_NORMALIZE_JAC_MANY_ALT)

int mbedtls_internal_ecp_normalize_jac(…)        // if defined(MBEDTLS_ECP_NORMALIZE_JAC_ALT
```


Exposed internal API (ecp_internal.h)

```
// if defined(ECP_MONTGOMERY)

int mbedtls_internal_ecp_double_add_mxz(…)      // if defined(MBEDTLS_ECP_DOUBLE_ADD_MXZ_ALT)

int mbedtls_internal_ecp_randomize_mxz(…)       // if defined(MBEDTLS_ECP_RANDOMIZE_MXZ_ALT)

int mbedtls_internal_ecp_normalize_mxz(…)       // if defined(MBEDTLS_ECP_NORMALIZE_MXZ_ALT)
```

Testing

- Testing for hardware acceleration ports is driven using the standard mbed OS tools (mbed CLI and mbed-greentea).
- The mbed OS git branch mbedtls-partner-workshop-17Q2 contains a provisional set of tests that can be used to verify hardware acceleration ports.
- The tests consist of a subset of the mbed TLS ECP test suite and the selftest functions.

ARM

Testing (cont)

- To run the tests make sure that mbed CLI and mbed-greentea are installed.

```
$ mbed --version
$ mbedgt --version
```

- Use the following command to compile the mbed TLS tests:

```
$ mbed test -m <MCU> -t <TOOLCHAIN> \
    --compile -n tests-mbedtls-ecp,tests-mbedtls-selftest
```

- Run the test using:

```
$ mbedgt -V -n tests-mbedtls-ecp,tests-mbedtls-selftest
```


ARM

Benchmarking

- Benchmarking application should be used to verify performance improvements of the porting
- It is recommended to run the benchmark application in advance, to see improvement
- https://github.com/ARMmbed/mbed-os-example-tls/tree/mbedtls-partner-workshop-17Q2/benchmark

- Compile, Run and check serial output

ARM

Benchmarking (example of serial output)

```
 SHA-256                     :      1985 KB/s,        59 cycles/byte
 SHA-512                     :       615 KB/s,       191 cycles/byte
 AES-CBC-128                 :      1401 KB/s,        83 cycles/byte
 AES-CBC-192                 :      1223 KB/s,        95 cycles/byte
 AES-CBC-256                 :      1085 KB/s,       108 cycles/byte
 AES-GCM-128                 :       443 KB/s,       265 cycles/byte
 AES-GCM-192                 :       423 KB/s,       278 cycles/byte
 AES-GCM-256                 :       405 KB/s,       290 cycles/byte
 AES-CCM-128                 :       583 KB/s,       201 cycles/byte
 AES-CCM-192                 :       519 KB/s,       226 cycles/byte
 AES-CCM-256                 :       468 KB/s,       251 cycles/byte
 CTR_DRBG (NOPR)             :      1101 KB/s,       106 cycles/byte
 CTR_DRBG (PR)               :       799 KB/s,       146 cycles/byte
...
 ECDHE-secp384r1             :    1029 ms/handshake
 ECDHE-secp256r1             :     684 ms/handshake
 ECDHE-Curve25519            :     585 ms/handshake
 ECDH-secp384r1              :     506 ms/handshake
 ECDH-secp256r1              :     341 ms/handshake
 ECDH-Curve25519             :     289 ms/handshake
```


ARM

What to do now

1. Compile and run the benchmark with the default (software) implementation
   https://github.com/ARMmbed/mbed-os-example-tls/tree/mbedtls-partner-workshop-17Q2/benchmark

2. Clone the partner workshop branch
   https://github.com/ARMmbed/mbed-os/tree/mbed-os-workshop-17q2

3. Run the tests:

```
$ mbed test -m <MCU> -t <TOOLCHAIN> \
    --compile -n tests-mbedtls-ecp,tests-mbedtls-selftest
```

4. Implement your own HW acceleration code, for example AES and create the file for your target:
   `mbed-os/features/mbedtls/targets/TARGET_XXXX/aes_alt.c`

5. Run the tests and verify correctness of integration

```
$ mbed test ...
```

6. Run the benchmark application and check the figures

ARM

Hands-on workshop

- Use this branch:
  https://github.com/ARMmbed/mbed-os/tree/mbedtls-partner-workshop-17Q2

- Workshop materials:
  https://github.com/ARMmbed/mbed-os-workshop-17q2

Questions?

ARM

Backup Slides

ECC vs. RSA

- Over the years confidence in RSA has been shaken
- EC is faster, it uses less memory and power than RSA already when 112 bit security is considered enough
- The edge of EC algorithms over RSA is just going to increase over time:
  - 112 bit: 2048 bit RSA, 224 bit EC
  - 128 bit: 3072 bit RSA, 256 bit EC
  - 192 bit: 8192 bit RSA, 384 bit EC
  - 256 bit: 15360 bit RSA, 512 bit EC
- RSA key transport has been removed from the TLS 1.3 standard