mbed TLS Technical Overview

ARM

Ron Eldor

Security Staff Application Engineer

ARM mbed

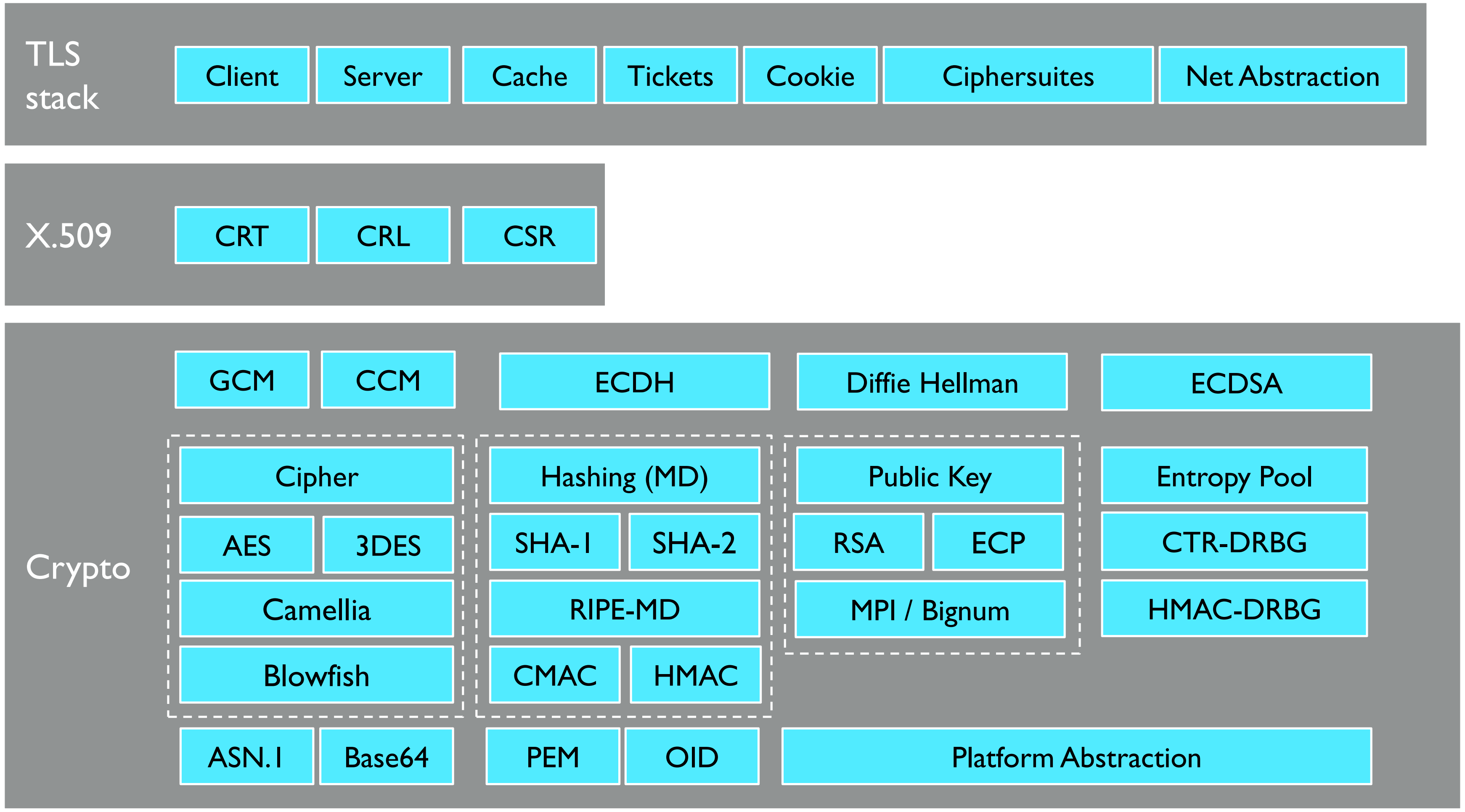
Silicon Partner Workshop - Wyboston Lakes
March 2017

High-level mbed TLS structure

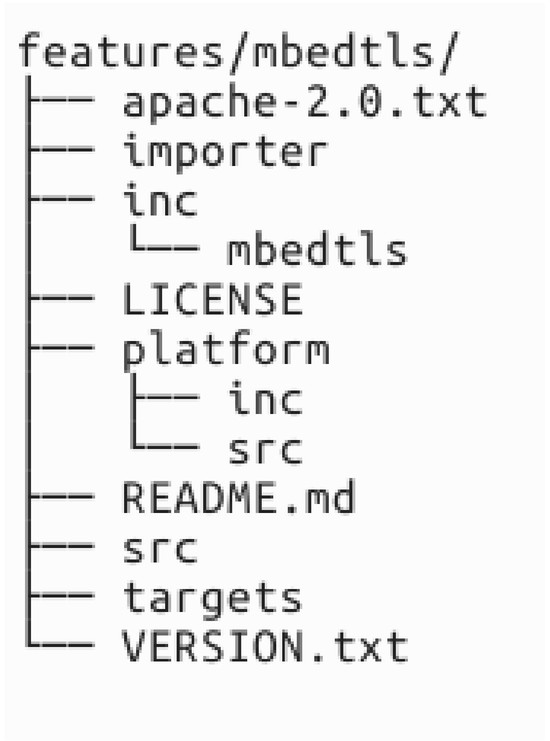
mbed TLS is consisted of the following components:

- TLS stack
- X509 certificate handling
- Cryptographic library

High-level mbed TLS structure



Source tree structure in mbed OS



```
features/mbedtls/
├── apache-2.0.txt
├── importer
├── inc
│   └── mbedtls
├── LICENSE
├── platform
│   ├── inc
│   └── src
├── README.md
├── src
├── targets
└── VERSION.txt
```

- src – mbed TLS source files
- inc – mbed TLS header files
- platform – platform dependent code
- targets – target specific implementations

Flexible / Configurable options

- Ability to enable individual modules, features and protocols versions
  - mbed-os/targets/targets.json
- Ability to add specific alternative features
  - mbed-os/features/mbedtls/target/TARGET_XXX/mbedtls_device.h
- Application can define TLS configuration
  - mbed_app.json: to enable specific features in mbed OS (including mbed TLS)
  - Override mbed TLS configuration in mbed_app.json: MBEDTLS_USER_CONFIG_FILE

- Speed / Memory sizes (RAM / Flash) tradeoffs
- Porting / Abstraction

Enable / Disable modules

- It is possible to disable a specific module only by removing the definition of that module. For example, to disable DES module, add the following line in MBEDTLS_USER_CONFIG_FILE*:

```
#if defined (MBEDTLS_DES_C)
#undef MBEDTLS_DES_C
#endif
```

*This is a user defined configuration file

Enable / Disable features in modules

- Specific features in modules could be disabled, if not needed, such as non used elliptic curves:

```
#if defined (MBEDTLS_ECP_DP_SECP192K1_ENABLED)
#undef MBEDTLS_ECP_DP_SECP192K1_ENABLED
#endif
```

Enable / Disable specific TLS protocol versions

- It is possible to enable or disable specific TLS standard versions:

```
#define MBEDTLS_SSL_PROTO_TLS1_1
```

```
#undef MBEDTLS_SSL_PROTO_TLS1_0
```

Speed / Memory sizes tradeoffs

- Memory footprint is configurable and can be modified. For example:
  - Cost of performance
    - MBEDTLS_MPI_WINDOW_SIZE can be reduced
  - Restrictive limitations
    - MBEDTLS_MPI_MAX_SIZE can be reduced
    - Specific ECP curves can be disabled
  - RAM/ROM tradeoff
    - MBEDTLS_AES_ROM_TABLES (Store AES tables to ROM to save RAM usage)
  - Use HW accelerated cryptographic modules
    - *_ALT

Knowledge based documentation on configuring options:
https://tls.mbed.org/kb/how-to/reduce-mbedtls-memory-and-storage-footprint

Porting / Abstraction

- mbed TLS has a Platform Abstraction Layer that supports the following:
  - printf() / fprintf() / snprintf() *(No port needed in mbed)*
  - calloc() / free() *(No port needed in mbed)*
  - exit() *(No port needed in mbed)*
  - time() *(No port needed in mbed)*
  - NV Seed read/write (We'll get to that later)

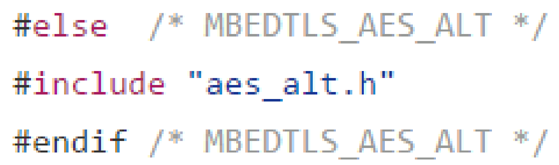
Porting / Abstraction (cont.)

- In addition mbed TLS provides `*_ALT` to provide alternative implementation at compile-time for part or full replacements of:
  - Symmetric ciphers: AES / ARC4 / Blowfish / Camellia / DES / XTEA
  - Asymmetric ciphers: ECC
  - Hashing: MD2 / MD4 / MD5 / SHA-1 / SHA-2 / RIPEMD-160
  - Should be implemented in features/mbedtls/targets/TARGET_XXXX

- Hardware entropy source
  - HW Entropy is to be implemented in targets/TARGET_XXXX

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

- More details will be explained in the Hardware acceleration presentation