WiFi and Secure Socket Offload in Zephyr™

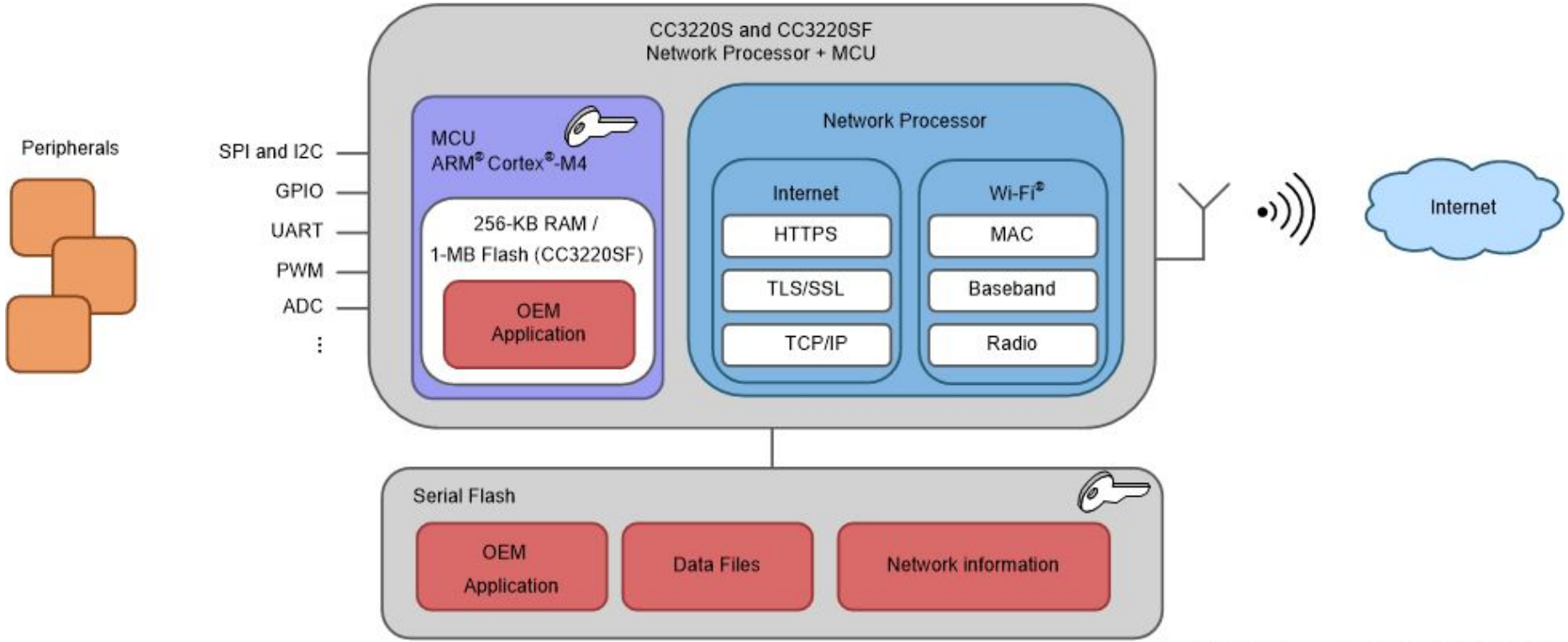
Gil Pitney / Texas Instruments

gpitney@ti.com

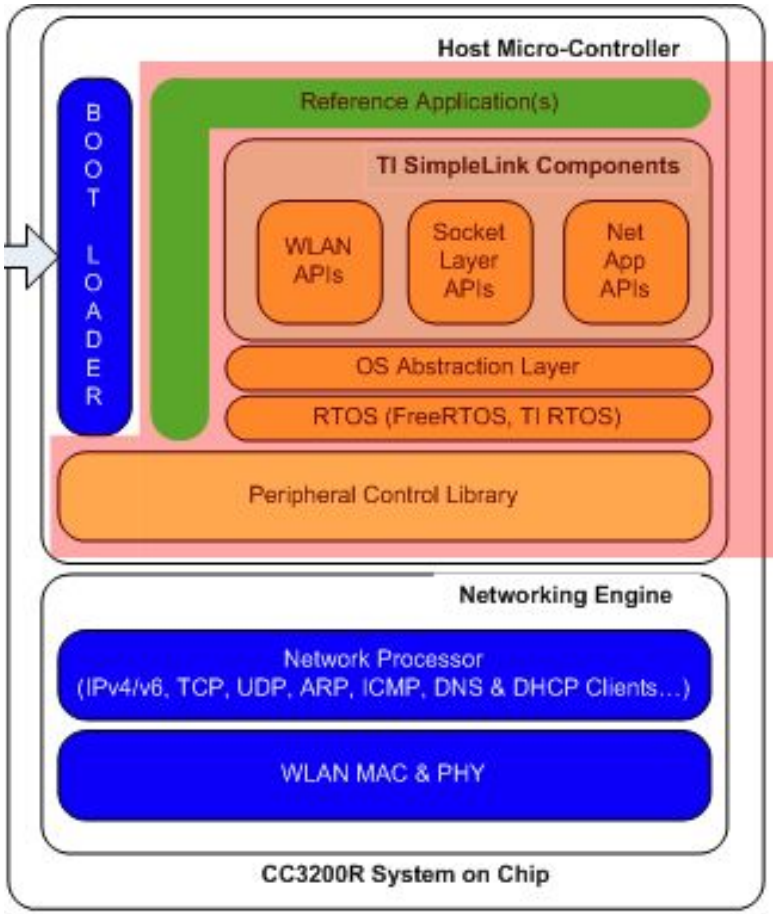
Motivation

- The TI SimpleLink CC32xx family of MCUs provides an SoC and supporting SDK which completely offloads the WiFi stack onto an integrated network coprocessor (NWP).
  - This provides significant memory, CPU, and energy savings.
  - All secure communications, certificate/key storage, crypto and power management is handled on the NWP.
  - The SimpleLink SDK supports TI RTOS and FreeRTOS, but is designed to be portable.
- Zephyr networking stack has support for WiFi via an offload tap (data plane), and some wifi management events (control plane).
- Zephyr has recently added TLS support into the BSD Socket API
  - This meshes well with TI's SimpleLink design
- **The goal is to efficiently integrate the SimpleLink offloaded capabilities into Zephyr, while leveraging Zephyr socket-based networking protocols.**
  - All work was done on the **CC3220SF-LaunchXL** development board.

TI CC3220SF SoC H/W Architecture



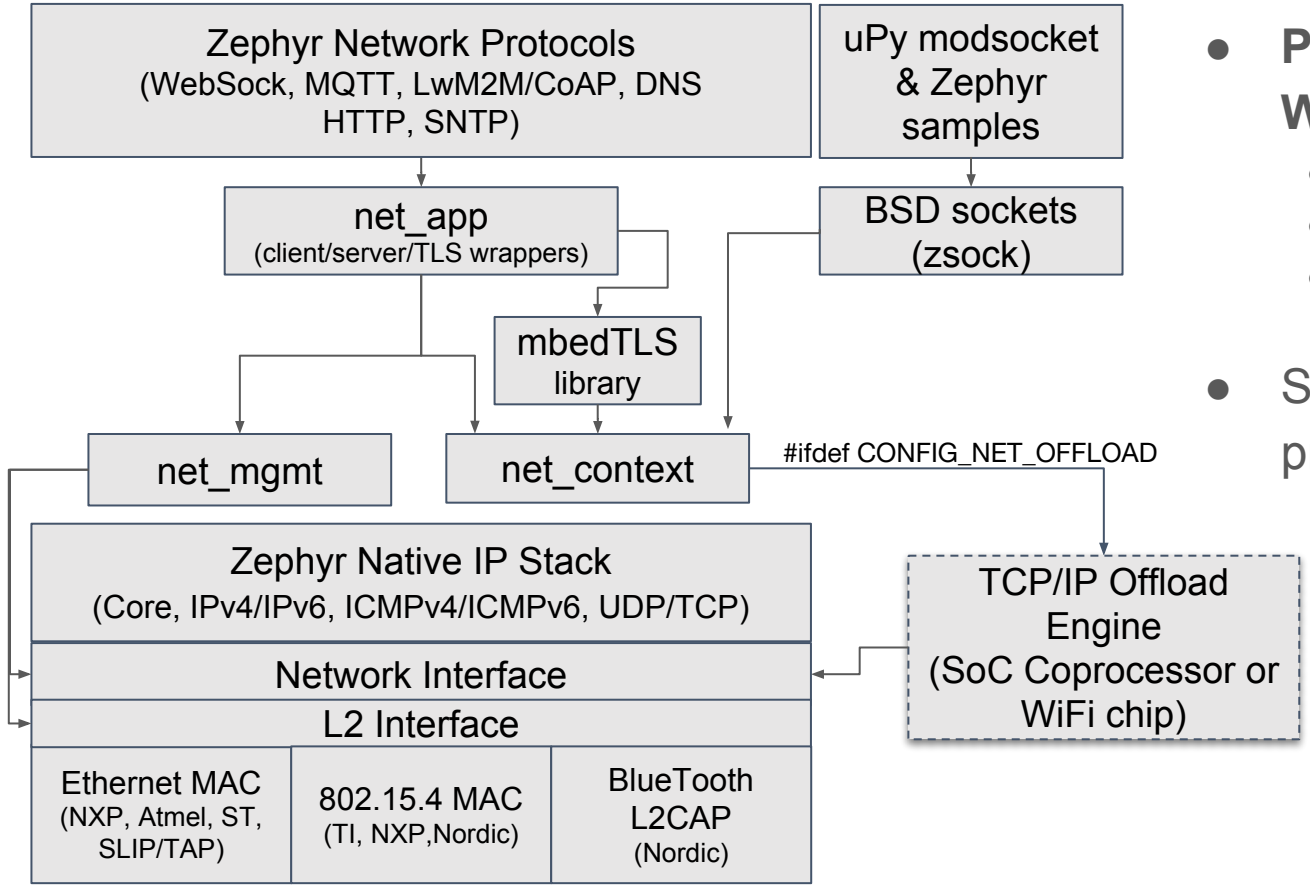
TI SimpleLink CC32xx SDK Architecture & APIs



- Device API: Manages hardware-related functionality such as start, stop, set, and get device configurations.
- WLAN API: Manages WLAN, 802.11 protocol-related functionality such as device mode (station, AP, or P2P), provisioning method, connection profiles, and connection policy.
- BSD Socket API: with TLS handled **under** the BSD API.
- NetApp API: Offloads networking services (HTTP, DHCP, mDNS).
- NetCfg API: Configures network parameters (MAC address, acquiring IP address by DHCP, setting the static IP address).
- Serial Flash API: for networking or user proprietary data.

Sources: swru368, swru369c

Zephyr Network Stack (Previous State)



- **Plan has been to support WiFi via offload chips.**
  - data via **NET_OFFLOAD** tap.
  - No WiFi L2 Drivers
  - No WiFi supplicant, or provisioning support (yet).
- Secure comms (SSL/TLS) provided by mbedTLS library

Options for TCP/IP Offload to the NWP (1/2)

Option 1: Use SimpleLink SDK APIs:

- How:
  - SDK already ported to Zephyr
  - `#include <SL_SDK>/simplelink.h`
  - `#include <SL_SDK>/sys/socket.h`
- Pros:
  - Zephyr apps get full access to SimpleLink WLAN, NetApp, Socket APIs.
  - Can still use Zephyr drivers: I2C, GPIO..
  - Offers fullest H/W entitlement.
- Cons:
  - No integration with Zephyr WiFi event management.
  - Will not leverage Zephyr's socket-based network protocols.

Option 2: Write an L2 Driver:

- How:
  - Use SimpleLink Raw Sockets
    - aka "Transceiver Mode".
  - Implement L2 send(), reserve() fxns.
  - Push received data via net_pkt to Zephyr IP core.
- Pros:
  - Hooks deeply into the Zephyr IP Core.
  - Enables Zephyr use cases like packet routing across network interfaces.
- Cons:
  - Does not fully leverage SimpleLink:
    - network buffer allocation, management
    - DHCP, DNS offloaded
    - Secure socket offloading

Options for TCP/IP Offload to the NWP (2/2)

Option 3: Offload at net_context():

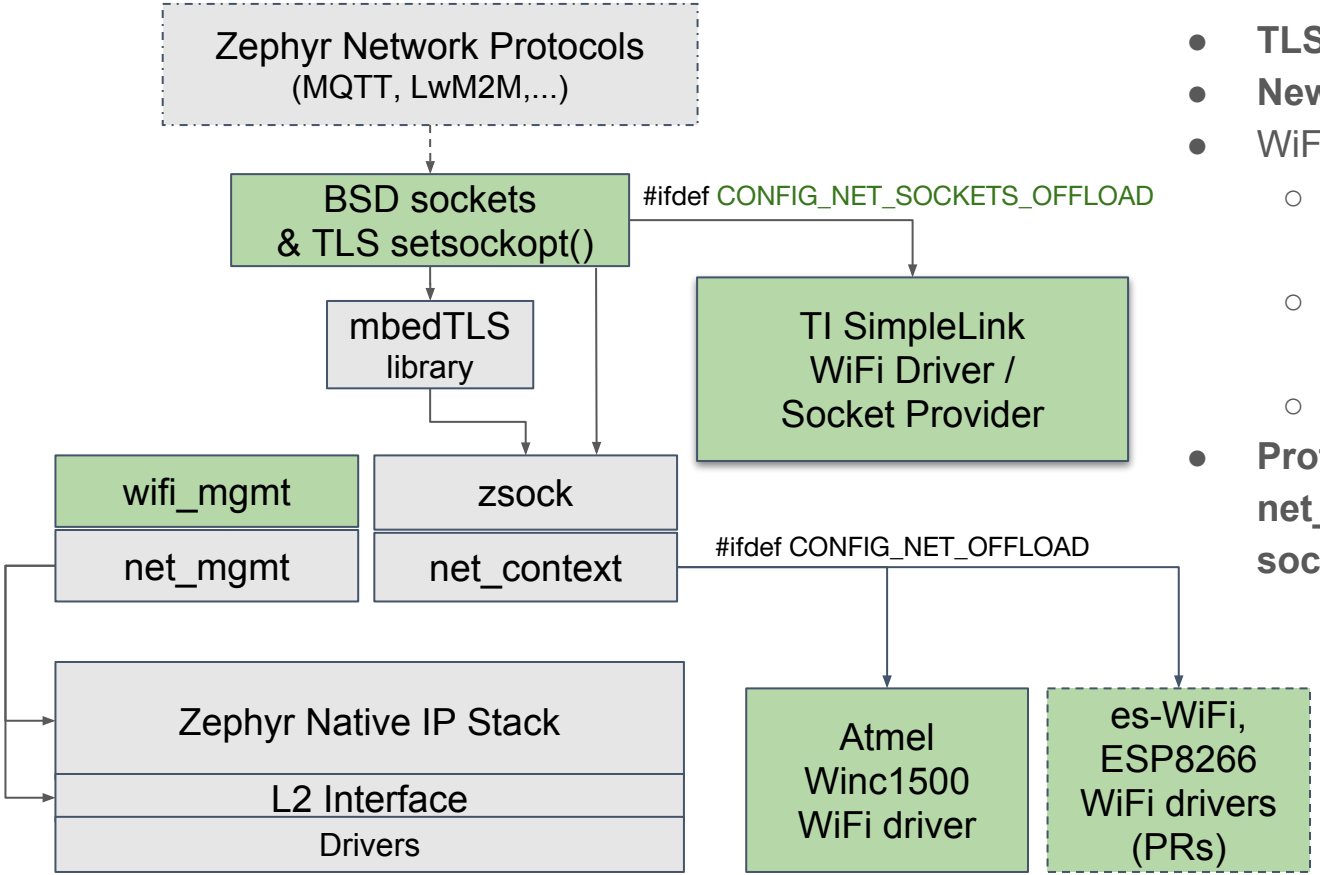
- How:
  - Enable `CONFIG_NET_OFFLOAD`
  - Write a Zephyr WiFi driver (cntrl + data)
- Pros:
  - TCP/IP stack is offloaded to the NWP.
  - Enables Zephyr use cases like packet routing across network interfaces.
- Cons:
  - Overheads:
    - Mapping sync BSD socket APIs to async net_context APIs and back.
    - Received data **copied** into net_bufs and queued.
    - Driver thread to select sockets and trigger callbacks
  - **Security: TLS handshake and crypto are not offloaded**

Option 4: Offload at BSD socket layer:

- How:
  - Enable `CONFIG_NET_SOCKETS_OFFLOAD`
  - Write a Zephyr WiFi driver (cntrl only)
  - Register offloaded socket fxns w/ Zephyr.
- Pros:
  - Avoids overheads of option 3)
  - **Secure socket communications get fully offloaded.**
  - DNS offloaded too (`getaddrinfo()`)
- Cons:
  - Currently, only one socket provider in the system
  - No packet routing across net interfaces.

This Option Chosen for TI SimpleLink

Zephyr Network Stack (New State)



- **TLS handled under socket APIs**
- **New offload tap** at BSD socket layer
- WiFi offload drivers implement:
  - iface_init: NWP init, defaults WLAN & network params.
  - Control: scan(), [dis]connect(), and callbacks to wifi_mgmt
  - Data: net_context() or sockets.
- **Protocols being migrated from net_app/net_context to BSD socket API.**

Zephyr: Adding TLS to Socket APIs

- Why?
  - TLS is hard to get right; many TLS library APIs and configuration options.
  - Let's make it easy to add TLS to non-secure **socket-based** networking apps/protocols.
- Adding TLS to a networking app via mbedTLS involves:
  - Creation/initialization of mbedtls ssl, config contexts, registration of entropy generator.
  - Setup certificates list.
  - Configuration of the TLS/SSL layer.
    - Set server/clilent mode
    - Set certificate authentication mode
    - Specify RNG and DBG functions
    - Set network tx/rx functions via `mbedtls_ssl_set_bio()`
  - Socket creation (standard POSIX); then connection via `mbedtls_net_connect()`
  - Read/Write via `mbedtls_ssl_read()/mbedtls_ssl_write()`
  - Teardown of mbedtls contexts.
- Zephyr wrapped all this with net_app, but we want to leverage standard APIs...

What's involved in establishing a secure channel?

Store Certificates/keys:

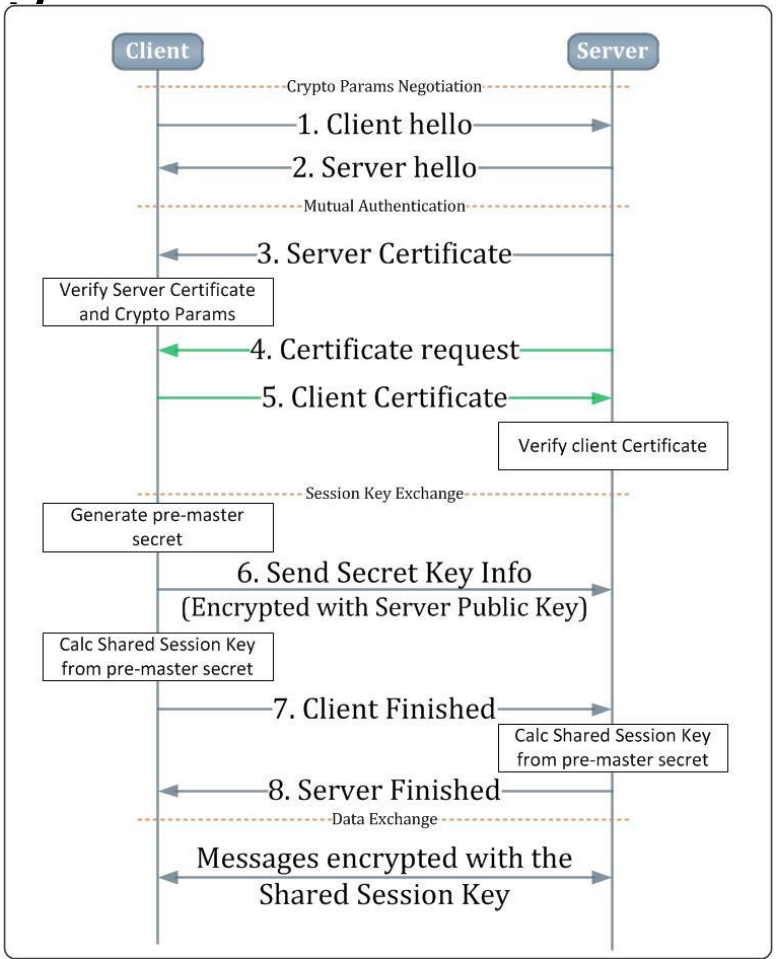
- Certificates/private keys provisioned into secure flash.
- Catalog of known Trusted Root CA Certificates

"TLS Handshake": connect()

- Cipher suite negotiation
- Authentication of the server and (optionally) the client
- Session key exchanged.

Data Exchange: send()/recv()

- Session key used to encrypt data on this channel.



www.it.com/lit/swpu332: Fig. 3

How to provision the certificates/keys to the device?

- The secrets should be kept secure from non-secure apps; eg,
  - On TI CC3220SF:
    - NWP runs the TCP/IP stack and crypto in a separate CPU (address space) from the MCU (running Zephyr). NWP has full access to the keys.
    - MCU **can** write new secrets (eg: via OTA updates). Secrets are signed, encrypted and have R/W access control levels.
  - On an ARMv8-M Device with Trusted Execution Environment:
    - Secrets can be stored in a secure memory partition, accessed by secure code.
    - (See talk by Andy Gross on Tuesday: “Zephyr and Trusted Execution Environments”)
- Storing secrets:
  - Method 1: Write a separate provisioning app to store certs/keys into secure flash filesystem.
  - Method 2: Use vendor production line tool to provision certs/keys to the device’s secure flash.

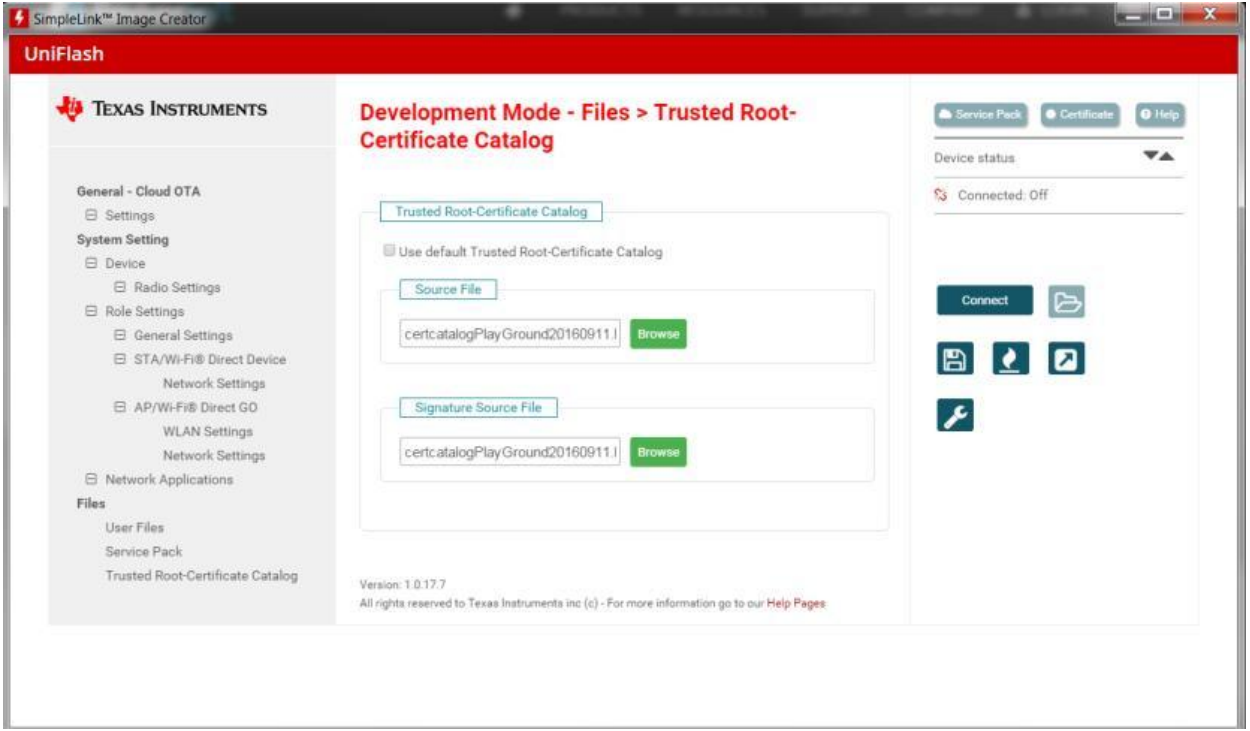
Method 1: Zephyr's tls_credential_add() API

```
/* Ideally, a separate application to store certs/keys into a secure file system: */

#if defined(CONFIG_TLS_CREDENTIALS)
#include <net/tls_credentials.h>

#define CA_CERTIFICATE_TAG 1
/* GlobalSign Root CA - R2 for https://google.com */
static const unsigned char ca_certificate[] = {
#include "globalsign_r2.der.inc"
};

/* Ideally, add credentials to secure flash: */
tls_credential_add(CA_CERTIFICATE_TAG, TLS_CREDENTIAL_CA_CERTIFICATE,
                   ca_certificate, sizeof(ca_certificate));

#endif
```

APIs enabled by a Kconfig variable.

Currently, credentials only saved in RAM, and done as part of network app/protocol initialization.

Method 2: Provisioning Certs/Keys on CC3220SF



TI UniFlash Tool:

- Enable TI catalog of Trusted CA Root Certificates
- Eg: Add google's "GlobalSign R2" DER file to secure flash.

At runtime:

- bind certificate's **filename** via its **sec_tag_t** to client socket using `setsockopt()`

Method 2: at init time, only need provide filenames

```
#include <net/tls_credentials.h>

#define CA_CERTIFICATE_TAG 1

#if defined(CONFIG_NET_SOCKETS_SECURE_OFFLOAD)
/* GlobalSign Root CA - R2 for https://google.com */
static const unsigned char ca_certificate[] = "globalsign_r2.der"
#else
/* Use Method 1: encoding full certificate: */
#endif

/* For method 2: Only the certificate's filename is associated with the tag: */
tls_credential_add(CA_CERTIFICATE_TAG, TLS_CREDENTIAL_CA_CERTIFICATE,
                   ca_certificate, sizeof(ca_certificate));
```

TBD: KConfig name may change before Zephyr LTS

So, now we have this “certificate tag” associated with a certificate or key, how to use it?

http_get: Retrieve google web page over https (1/2)

```
#include <net/socket.h>
#if defined(CONFIG_TLS_CREDENTIALS)
#include <net/tls_credentials.h>
#define HTTP_PORT "443"
#else
#define HTTP_PORT "80"
#endif
#define HTTP_HOST “google.com”
#define REQUEST "GET / HTTP/1.0\r\nHost: " HTTP_HOST "\r\n\r\n"

main() {
     static char response[1024];
     static struct addrinfo hints;
     struct addrinfo *res;
     int sock;

     hints.ai_family = AF_INET;
     hints.ai_socktype = SOCK_STREAM;
     getaddrinfo(HTTP_HOST, HTTP_PORT, &hints, &res);
```

For HTTPS, using port 443

Idea: Encapsulate TLS under POSIX Socket API (2/2)

```
#if defined(CONFIG_TLS_CREDENTIALS)
     sock = socket(res->ai_family, res->ai_socktype, IPPROTO_TLS_1_2);
     sec_tag_t sec_tag_opt[] = {
          CA_CERTIFICATE_TAG,
     };
     setsockopt(sock, SOL_TLS, TLS_SEC_TAG_LIST, sec_tag_opt, sizeof(sec_tag_opt));
     setsockopt(sock, SOL_TLS, TLS_HOSTNAME, HTTP_HOST, sizeof(HTTP_HOST));
#else
     sock = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
#endif

     /* Rest of network app/protocol code remains unchanged: */
     connect(sock, res->ai_addr, res->ai_addrlen);
     send(sock, REQUEST, sizeof(REQUEST)-1, 0);
     do {
          len = recv(sock, response, sizeof(response) - 1, 0);
          response[len] = 0;  printf("%s", response);
     }
     close(sock);
}
```

TLS protocol family

Certificate bound to socket via tag

connect() handles the TLS handshake

send()/recv() now done over secure channel

TLS Security added with a few lines of setsockopt() code. With TI SimpleLink, all secure comms offloaded.

Summary

- The **TI SimpleLink CC3220SF SoC** allows the TCP/IP stack, WiFi, secure communications, encryption, secrets storage and power management to be offloaded from the MCU (Zephyr) to an integrated network coprocessor (NWP).

How?

- The SimpleLink NWP “host driver” is ported to Zephyr via a thin OSAL.
- The SimpleLink Zephyr WiFi driver implements the WiFi control API, and sends [dis]connect/scan notifications back to the network event manager.
- Certificates are provisioned to CC3220SF **secure flash** via TI UniFlash tool.
- The SimpleLink Zephyr WiFi driver registers it BSD socket APIs to the new **Zephyr socket** layer, and
- with the help of Zephyr’s new TLS socket APIs, we can achieve **full secure socket offload**, available to Zephyr’s **socket-based** net protocols.

Thank You!

THE LINUX FOUNDATION