X.509 Client Authentication in Zephyr

Kevin Townsend
Linaro
2022 Zephyr Developer Summit

Who Am I?

Kevin Townsend,

Technical Lead at Linaro (LITE)

Github: microbuilder

Zephyr maintainer for:

- Arm Arch
- TF-M Integration
- Zscilib

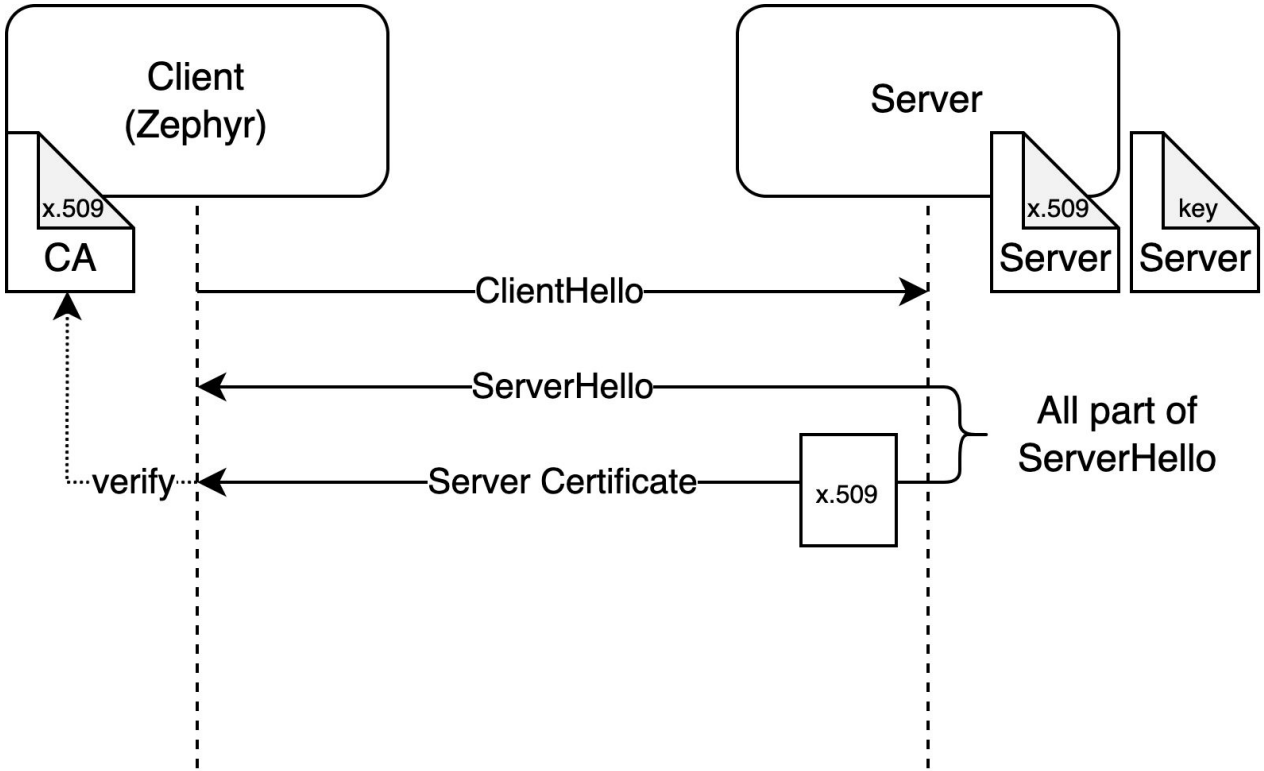
Kudos to my colleague **David Brown** for his technical expertise in the work this presentation is based upon.

Agenda

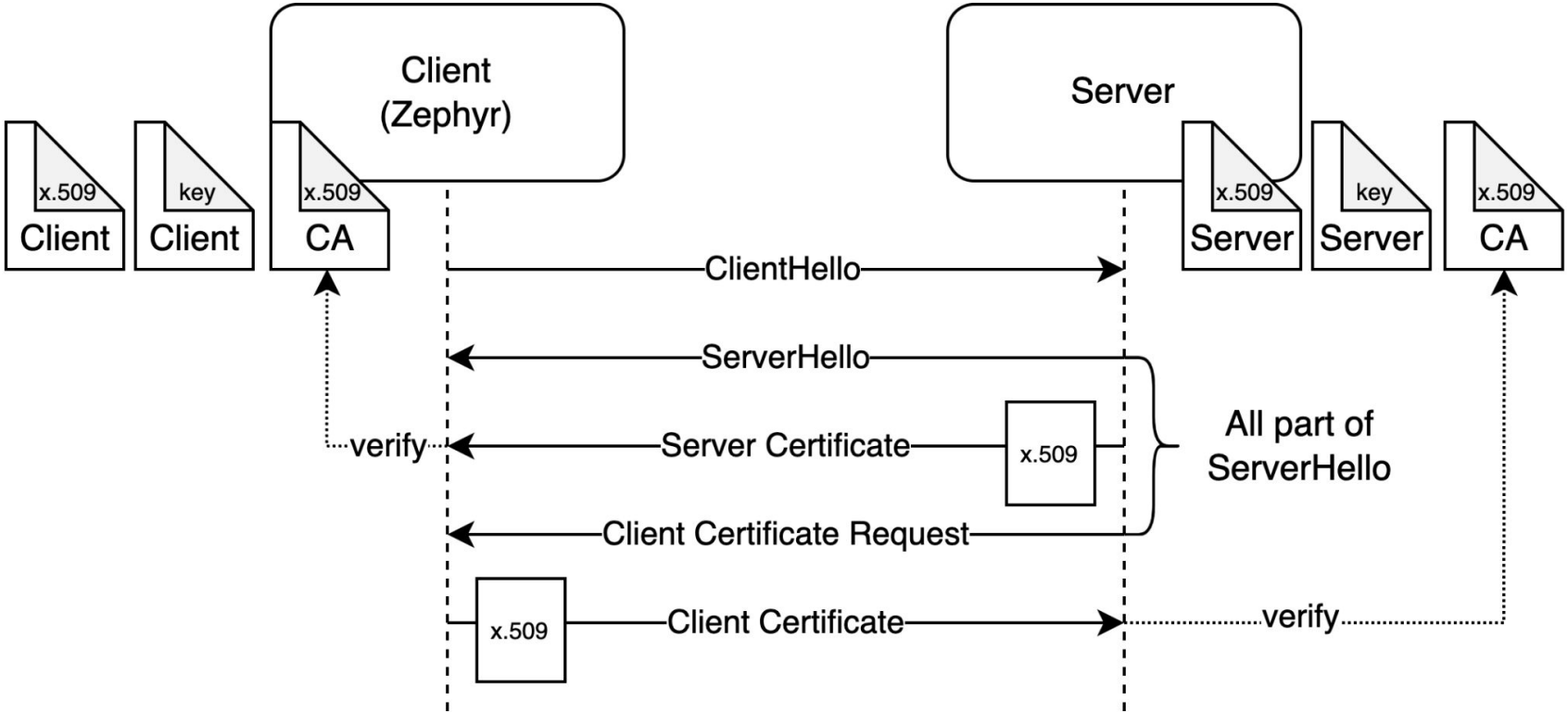
- What do we mean by ‘X.509 Client Authentication’?
- Why bother?
- Generating CA keys/cert
- Generating server keys/certs
- Generating client keys/certs
- Writing a mutual TLS TCP server
- Enabling mutual TLS in Zephyr
- Real World Usage:
  - Trust, but verify (Certificate Revocation)
  - Storage-Free Key Derivation
  - Confidential AI Architecture

What do we mean by ‘X.509 Client Authentication’?

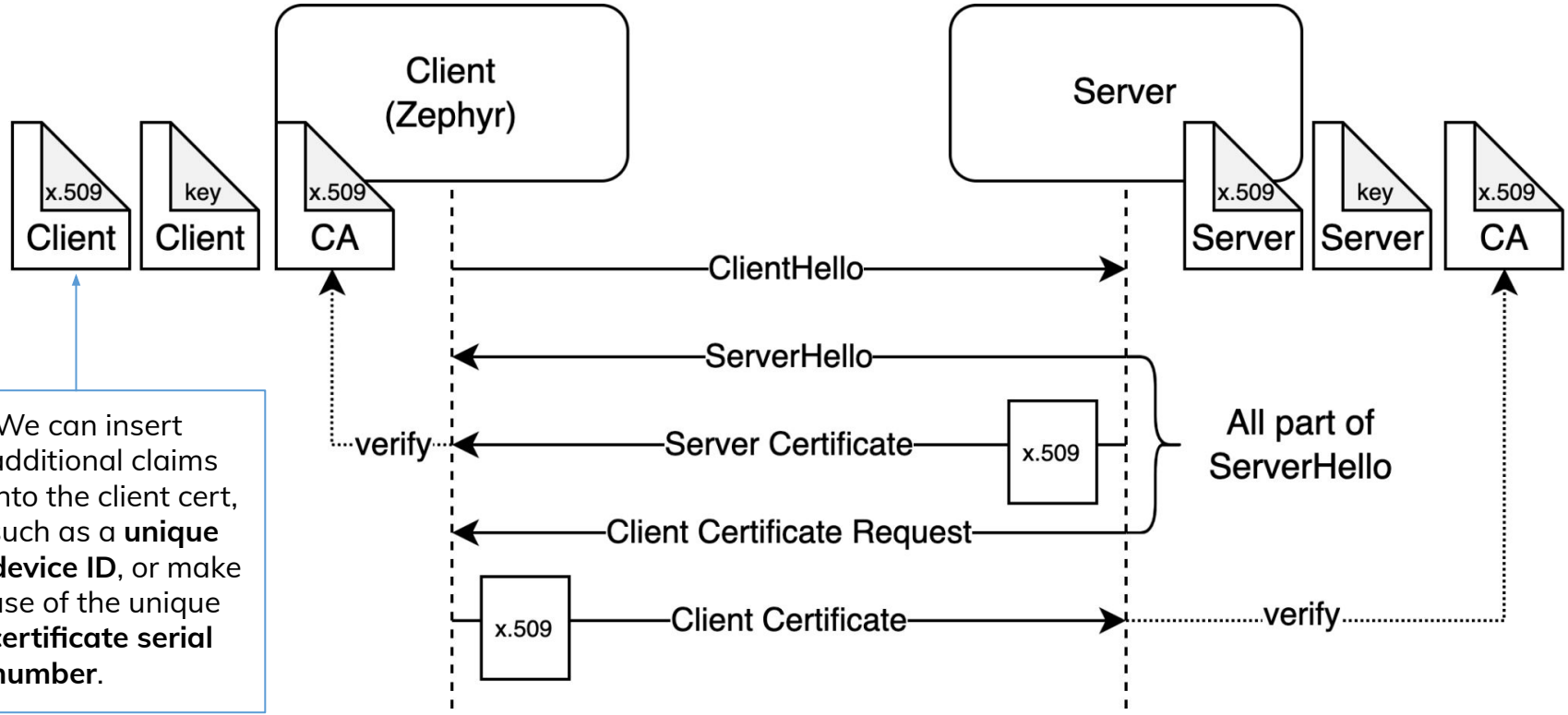
Identity Verification: Basic TLS



Identity Verification: Mutual TLS



Client Authentication?



⚠ We can insert additional claims into the client cert, such as a **unique device ID**, or make use of the unique **certificate serial number**.

Why bother?

Why Bother?

- Based on **public-key cryptography**
- No more transmitting **passwords** or storing static device secrets remotely!
- Private keys can be isolated in a secure enclave
- Easier to bind private key values to specific devices/chips
- Both sides have a high level of confidence in the identity/data of the other party
- **Standards based**: part of the core TLS standard
- Delegates trust management to the certificate authority (CA)
- Enables remote certificate revocation
- Can maintain intermediary CA certs for OEMs, vendors, etc.
- Limited security lifetime (certificate expiry dates)

Downsides:

- Some management overhead, though not necessarily more than alternatives

Generating Keys/Certificates

Generate CA key/cert

```
# Generate a root CA key (keep this safe!)
$ openssl ecparam -name prime256v1 -genkey -out CA.key

# Generate an X.509 certificate from CA.key assigning O and CN subject fields
# NOTE: CN should include a year or distinctive value to ensure a unique subj line
$ openssl req -new -x509 -days 3650 -key CA.key -out CA.crt \
        -subj "/O=Linaro/CN=Root CA"

# Optionally verify the certificate contents
$ openssl x509 -in CA.crt -noout -text
```

Generate Server Key and CSR

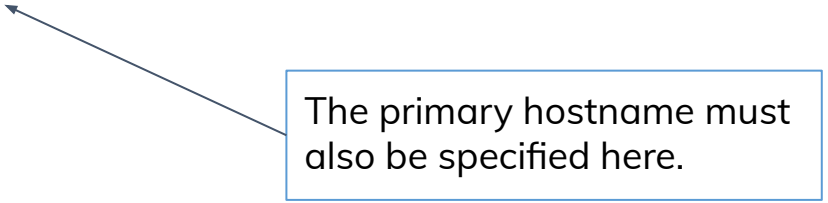
```
# Generate the server's private key for TLS
$ openssl ecparam -name prime256v1 -genkey -out SERVER.key

# Generate a certificate signing request (CSR) for our key
$ openssl req -new -sha256 -key SERVER.key -out SERVER.csr \
    -subj "/O=Linaro, LTD/CN=localhost"
```

NOTE: The 'CN' field ('localhost' here) must be set to the server's hostname. Setting CN to '*.linaro.org', for example, would cover all linaro.org subdomains/servers. Additional hostnames can also be set in the extension data.

Set Server CSR Extensions

```
# Create a config snippet to add proper extensions to this key
# Be sure to set ‘DNS:’ to the server’s actual hostname!
$ echo "subjectKeyIdentifier=hash" > server.ext
$ echo "authorityKeyIdentifier=keyid,issuer" >> server.ext
$ echo "basicConstraints = critical, CA:FALSE" >> server.ext
$ echo "keyUsage = critical, digitalSignature" >> server.ext
$ echo "extendedKeyUsage = serverAuth" >> server.ext
$ echo "subjectAltName = DNS:localhost" >> server.ext
```



Sign Server CSR with the CA

```
$ openssl x509 -req -sha256 \
    -CA CA.crt \
    -CAkey CA.key \
    -days 3560 \
    -CAcreateserial \
    -CAserial CA.srl \
    -in SERVER.csr \
    -out SERVER.crt \
    -extfile server.ext
```

Check Server Certificate

$ openssl x509 -in SERVER.crt -noout -text

```
Data:
Version: 3 (0x2)
Serial Number: 16081112071318811689 (0xdf2b963a3c343429)
Signature Algorithm: ecdsa-with-SHA256
Issuer: O=Linaro, CN=Root CA
Validity
Not Before: May 26 20:43:23 2022 GMT
Not After : Feb 23 20:43:23 2032 GMT
Subject: O=Linaro, LTD, CN=localhost
Subject Public Key Info:
Public Key Algorithm: id-ecPublicKey
            Public-Key: (256 bit)
            pub:
            04:6f:1b:1f:70:c7:1e:6d:78:51:b1:f5:de:cd:55:
            86:ee:2e:51:51:57:cb:63:ab:4e:10:65:07:94:d1:
            e5:73:94:d0:72:64:48:c6:bb:6c:2f:ef:8e:50:55:
            28:54:c1:65:08:f8:db:7e:83:d5:4d:90:7d:16:b6:
            75:5b:07:ee:f5
            ASN1 OID: prime256v1
            NIST CURVE: P-256
X509v3 extensions:
X509v3 Subject Key Identifier:
            A0:C5:88:31:0B:29:06:4E:76:06:88:BA:D7:58:F6:68:F5:45:80:68
X509v3 Authority Key Identifier:
            DirName:/O=Linaro/CN=Root CA
            serial:BB:B2:6A:B2:5D:85:A1:CE

X509v3 Basic Constraints: critical
            CA:FALSE
X509v3 Key Usage: critical
            Digital Signature
X509v3 Extended Key Usage:
            TLS Web Server Authentication
X509v3 Subject Alternative Name:
            DNS:localhost
Signature Algorithm: ecdsa-with-SHA256
30:46:02:21:00:ef:57:05:f8:4a:1f:db:d0:c8:f9:00:a2:c9:
e8:1c:e3:c5:1f:50:19:59:76:7f:34:f0:48:c9:1b:a3:ab:9f:
50:02:21:00:a5:40:63:47:85:71:9b:96:27:e7:19:95:f4:a8:
d4:ca:26:82:39:db:a3:7b:a5:28:70:ff:e5:4a:5a:de:48:c5
```

Subject: O=Linaro, LTD, CN=localhost

Parent Authority (Root CA)

Extensions

Generate Device Key and CSR

```
# Generate a private key for this device
$ openssl ecparam -name prime256v1 -genkey -out DEVICE.key

# Set a unique ID for this device (every certificate MUST have a unique subj!)
$ export DEVID=$(uuidgen | tr '[:upper:]' '[:lower:]') && echo $DEVID
544a263a-49d8-4043-8c50-279f38e4a520

# Generate a CSR for this key
$ openssl req -new \
    -key DEVICE.key \
    -out DEVICE.csr \
    -subj "/O=Linaro/CN=$DEVID/OU=Linaro Device Cert"
```

Useful to identify your device on a server, etc.!

NOTE: '/OU' here is optional. It is added here to differentiate multiple certs on the same device.

Linaro

Sign the Device CSR with the CA

```
$ openssl x509 -req -sha256 \
    -CA CA.crt \
    -CAkey CA.key \
    -days 3560 \
    -in DEVICE.csr \
    -out DEVICE.crt
Signature ok
subject=/O=Linaro/CN=544a263a-49d8-4043-8c50-279f38e4a520/OU=Linaro Device Cert
Getting CA Private Key

# Examine the certificate
$ openssl x509 -in DEVICE.crt -noout -text
```

Make Keys/Certs C-Friendly

```
# Convert the CA certificate to a text file
$ sed 's/.*/"&\\r\\n"/' CA.crt > ca_crt.txt

# Convert the device certificate to a text file
$ sed 's/.*/"&\\r\\n"/' DEVICE.crt > device_crt.txt

# Convert the device private key in DER format to a text file
$ openssl ec -in DEVICE.key -outform DER |
    xxd -i > device_key.txt
```

Key/Certificate Generation Script

Bash script to generate keys and certificates:
gist.github.com/microbuilder/cf928ea5b751e6ea467cc0cd51d2532f#file-certg

Writing a Mutual TLS TCP server

Linaro

Mutual TLS Server Proof of Concept

Boilerplate TCP server in golang:

gist.github.com/microbuilder/cf928ea5b751e6ea467cc0cd51d2532f#file-main-go

Mutual TLS Server (golang)

```
// Load server key pair
cer, err := tls.LoadX509KeyPair("SERVER.crt", "SERVER.key")

// Create a certificate pool with the CA certificate to verify client certificate signatures
certPool := x509.NewCertPool()
caCert, err := ioutil.ReadFile("CA.crt")
certPool.AppendCertsFromPEM(caCert)
```

Mutual TLS Server (golang)

```
// Construct a TLS config with our client CA pool and server certificate/key
config := tls.Config{
    MinVersion: tls.VersionTLS12,           // Set the minimum TLS version
    Certificates: []tls.Certificate{cer},  // Set the server certificate and private key
    ClientAuth: tls.RequireAndVerifyClientCert,
    ClientCAs: certPool,                    // Set CA cert(s) for client cert verification
    VerifyPeerCertificate: validatePeer, // Callback for additional client cert verification
}

func validatePeer(rawCerts [][]byte, verifiedChains [][]*x509.Certificate) error {
        // Additional verification beyond CA signature and date goes here
}
```

Linaro

Mutual TLS Server (golang)

```
// Listen for TCP connections on ‘hostname’
hostname := “localhost”
fmt.Println("Starting mTLS TCP server on " + hostname + ":8443")
listener, err := tls.Listen("tcp", hostname + ":8443"), &config)
defer listener.Close()

for {
    // Accept incoming connections
    conn, err := listener.Accept()
    // Concurrent connection handling
    go handleConnection(conn)
}
```

Make sure this matches ‘/CN’ in the server certificate’s subject line!

Mutual TLS Server (golang)

```
func handleConnection(c net.Conn) {
    fmt.Println("Connection accepted from", c.RemoteAddr())

    ... code to request TLS handshake, which will also trigger validatePeer ...

    state := tlsConn.ConnectionState()
    for _, v := range state.PeerCertificates {
        fmt.Printf("Client certificate:\n")
        fmt.Printf("- Issuer CN: %s\n", v.Issuer.CommonName)
        fmt.Printf("- Subject: %s\n", v.Subject)
    }
    c.Close()
}
```

NOTE: We can get our UUID (etc.) from here to determine the client device's ID

Linaro

Testing (Valid Certificate)

```
$ openssl s_client \
  -cert DEVICE.crt \
  -key DEVICE.key \
  -CAfile CA.crt \
  -connect localhost:8443
```

The golang server should output:

```
Starting mTLS TCP server on localhost:8443
Connection accepted from xxx.xxx.xxx.xxx:xxxxx
Client certificate:
- Issuer CN: Root CA
- Subject: CN=544a263a-49d8-4043-8c50-279f38e4a520,OU=Linaro Device Cert,O=Linaro
```

Testing (Self-Signed Certificate)

```
$ openssl ecparam -name prime256v1 -genkey -out DEVBAD.key
$ openssl req -new -x509 -sha256 -days 365 \
   -key DEVBAD.key -out DEVBAD.crt -subj "/O=Linaro/CN=12345"
$ openssl s_client \
  -cert DEVBAD.crt -key DEVBAD.key -CAfile CA.crt \
  -connect localhost:8443
```

The golang server should output:

```
Starting mTLS TCP server on localhost:8443
Connection accepted from xxx.xxx.xxx.xxx:xxxxx
Client handshake error: tls: failed to verify client certificate: x509: certificate
signed by unknown authority
```

Enabling Mutual TLS in Zephyr

Linaro

Important KConfig Flags

You may need to increase the number of credentials (the default value is 4):

- CONFIG_TLS_MAX_CREDENTIALS_NUMBER=8

* Credentials are individual keys and certificates maintained by the TLS stack. Supported credential types in Zephyr 3.1 are:

- TLS_CREDENTIAL_CA_CERTIFICATE
- TLS_CREDENTIAL_SERVER_CERTIFICATE
- TLS_CREDENTIAL_PRIVATE_KEY
- TLS_CREDENTIAL_PSK
- TLS_CREDENTIAL_PSK_ID

Import CA and Device Cert/Key Credentials*

```
// Load the (DER format) CA certificate into the C project
// Required to verify the remote server's certificate
static const unsigned char raw_caroot_crt[] = {
   #include "ca_crt.txt"
};
const unsigned char *caroot_crt = raw_caroot_crt;
const size_t caroot_crt_len = sizeof(raw_caroot_crt);
```

```
// Load the (DER format) device certificate into the C project
static const unsigned char raw_local_crt[] =
#include "device_crt.txt"
;
const unsigned char *local_crt = raw_local_crt;
const size_t local_crt_len = sizeof(raw_local_crt);
```

```
// Load the device private key in the C project
static const unsigned char raw_local_key[] = {
#include "device_key.txt"
};
const unsigned char *local_key = raw_local_key;
const size_t local_key_len = sizeof(raw_local_key);
```

* .txt files generated earlier with openssl and sed

Define a Tag List

```
#define APP_CA_TAG     1
#define APP_LOCAL_CERT_TAG  2

static sec_tag_t m_sec_tags[] = {
    APP_CA_TAG,
    APP_LOCAL_CERT_TAG,
};
```

- Tags allow you to group credentials together, such as a CA certificate, and your server/client certificate and private key, and associate those with a connection.
- Each tag can have up to 1 of each '`tls_credential_type`' associated with it. Connections can have multiple tags.

Assign Cert/Key Payloads to Tag(s)

```
/* Add CA certificate (used to verify server). */
tls_credential_add(APP_CA_TAG, TLS_CREDENTIAL_CA_CERTIFICATE,
        caroot_crt, caroot_crt_len);

/* Add local (DER-format) certificate, signed by trusted CA. */
tls_credential_add(APP_LOCAL_CERT_TAG, TLS_CREDENTIAL_SERVER_CERTIFICATE,
        local_crt, local_crt_len);

/* Add local private key, associated with above cert. */
tls_credential_add(APP_LOCAL_CERT_TAG, TLS_CREDENTIAL_PRIVATE_KEY,
        local_key, local_key_len);
```

Request Client/Peer Verification

```
/* Set our socket, requesting TLS 1.2 support. */
int sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TLS_1_2);

/* Set remote hostname for socket. */
#define HOST “localhost”
setsockopt(sock, SOL_TLS, TLS_HOSTNAME, HOST, sizeof(HOST));

/* Enable peer verification the during TLS handshake. */
int peer_verify = 2;
zsock_setsockopt(sock, SOL_TLS, TLS_PEER_VERIFY,
        &peer_verify, sizeof(peer_verify));
```

Without this, the TLS connection is insecure, and you have no guarantee who you are talking to, or if there is someone in between you and the remote server!

Linaro

Set the Tag List

```
/* Set the tag list for this socket (2 tags in this case). */
zsock_setsockopt(sock, SOL_TLS, TLS_SEC_TAG_LIST,
        m_sec_tags, sizeof(m_sec_tags));
```

NOTE: It's up to the server to decide if client certificates will be used or not. If the server is configured to require them, a request will be made to the Zephyr TLS stack for our client certs during the normal TLS handshake.

Real World Usage

Trust, but verify (Certificate Revocation)

The TLS stack will only verify the **CA signature** and **date range** of the client cert.

Each certificate has a **unique serial number** assigned during CSR processing.

Remote servers should also check that:

- this certificate has actually been registered (serial + device ID verification)
- this certificate hasn't been revoked by the certificate management service

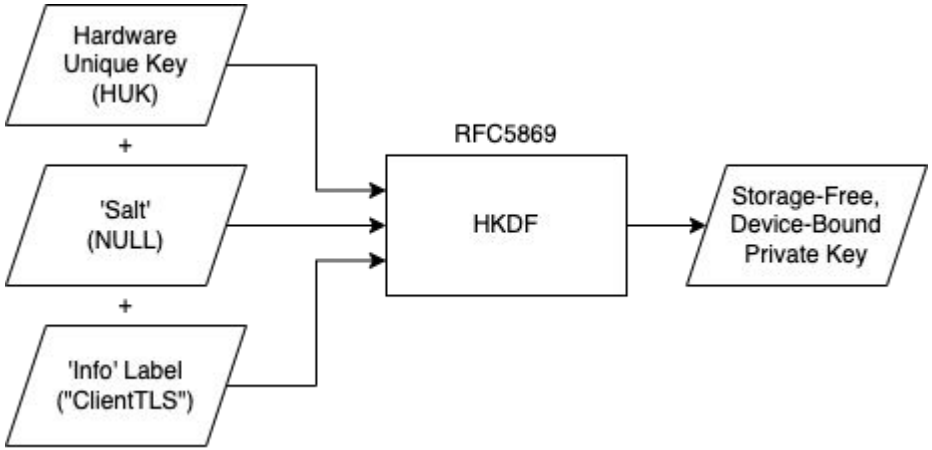
This requires some sort of certificate management system, and the use of:

```
func validatePeer(rawCerts [][]byte, verifiedChains [][]*x509.Certificate) error {
    // Additional verification beyond CA signature and date goes here
}
```

Storage-Free Key Derivation

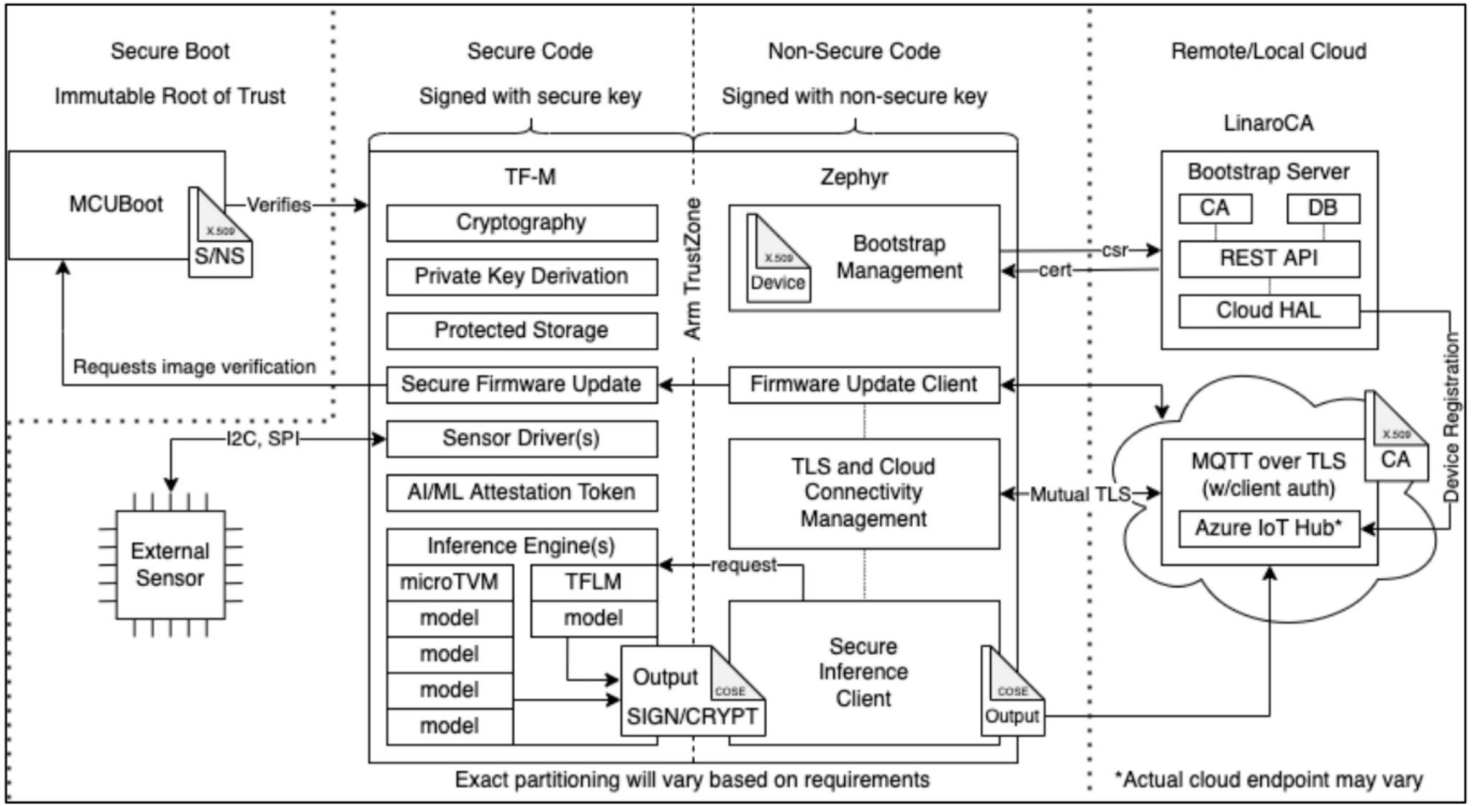
- Private key storage is high risk
- Derive device-bound key w/HUK
- Key regenerated at boot
- Persistent across updates
- Generate a CSR (MbedTLS, etc.)
- Send CSR to CA for signing
- X.509 cert stored in the open

* This same approach can also be used to derive a device UUID



Linaro

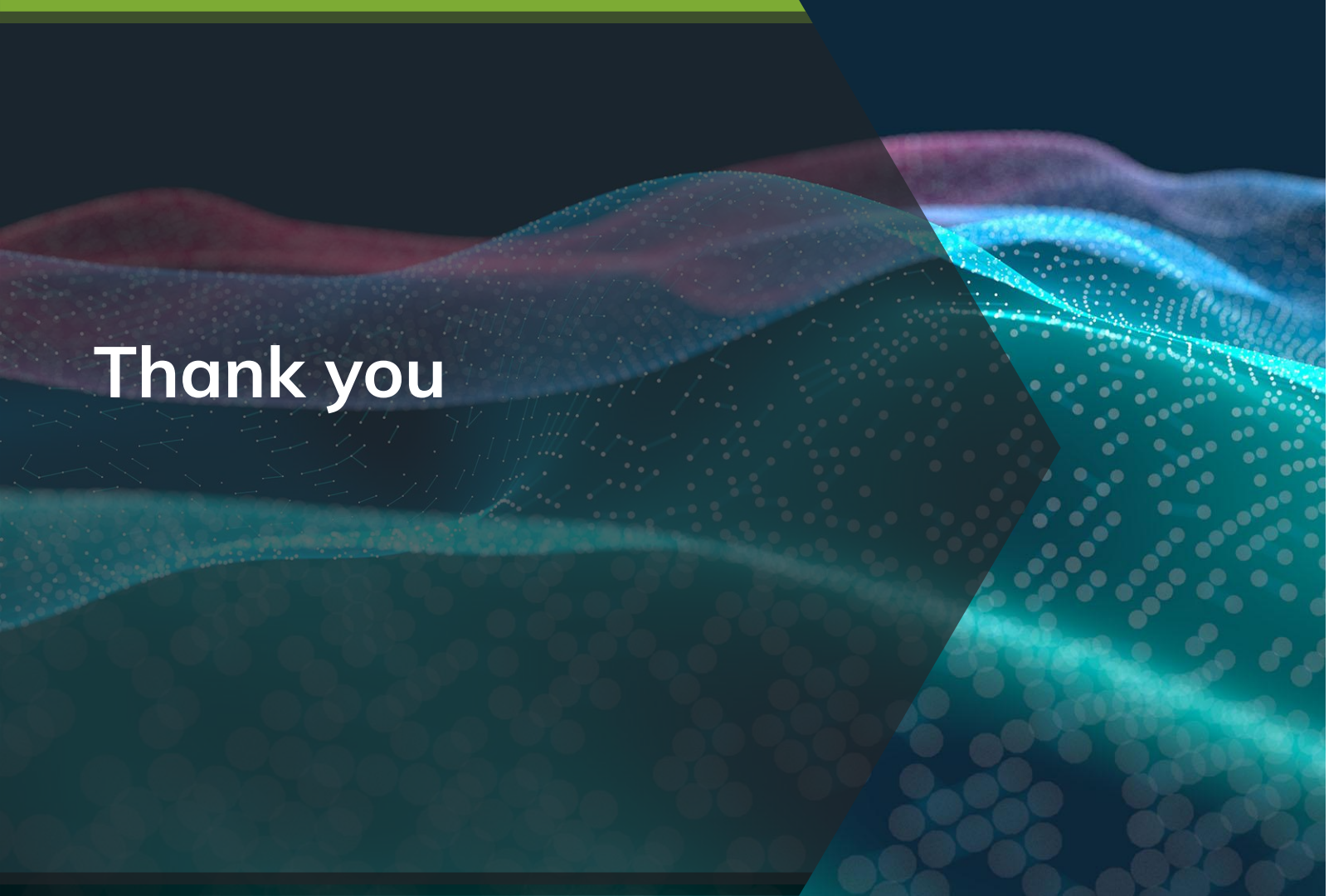
Confidential AI Proof of Concept



Confidential AI: Further Information

28 June, 2022: Linaro and Arm Confidential AI Tech Event
https://www.linaro.org/events/linaro-and-arm-confidential-ai-tech-event/

Linaro



Linaro