Boot to Cloud Security Considerations with IoT

Kevin Townsend
Zephyr Developer Summit
Prague, 28 June 2023

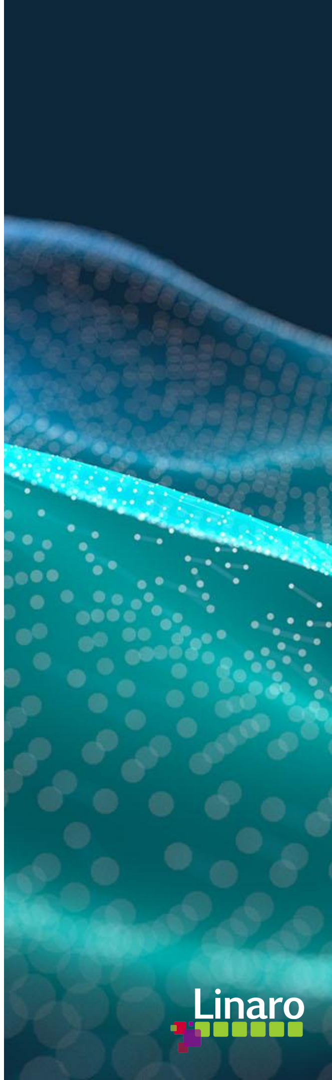
Linaro

About Me

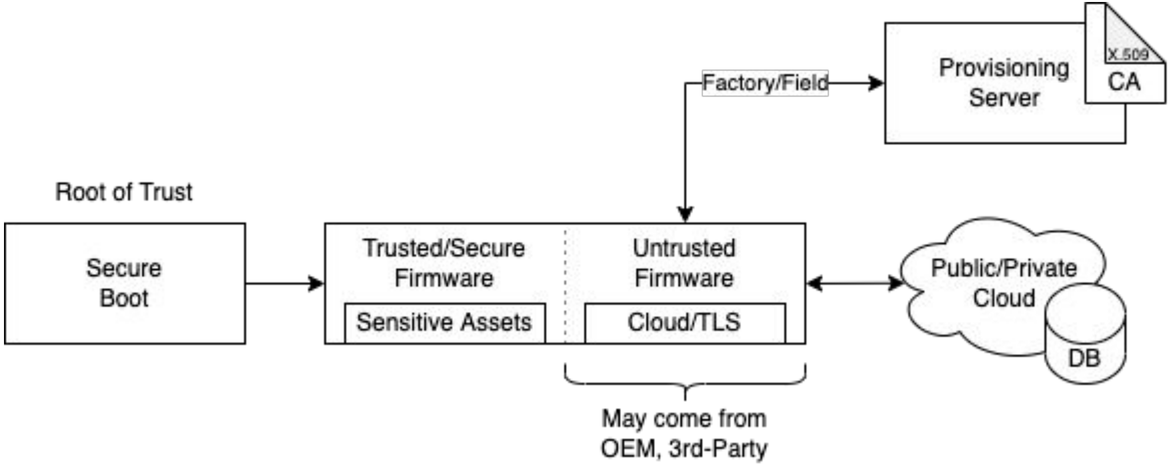
- Tech Lead at Linaro, focusing on Arm, RTOS, and IoT Security
- ~15 years of full time open source development
- Zephyr maintainer for Aarch32, TF-M, zscilib
- Github: @microbuilder

Agenda

- Secure Boot
- Device Provisioning
  - Storage-Free Key Derivation
- Securing Data in Transit
- Securing Data at Rest
- Example: Confidential AI
- Checklist

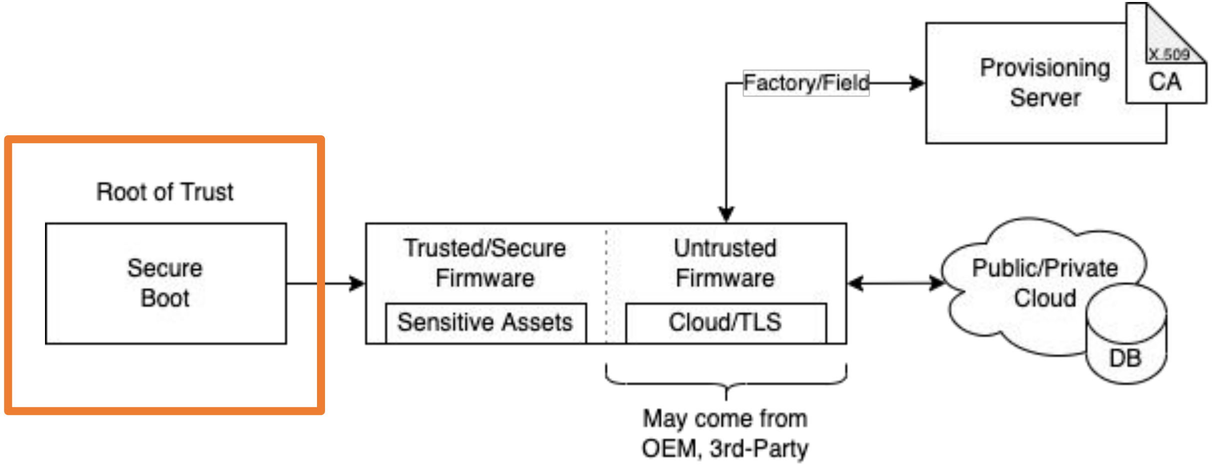


Core Components in a Secure IoT System



Secure Boot

Core Components in a Secure IoT System



Secure Boot

- As the **root of trust** this is the most critical component in a secure system!
  - Shouldn’t be an afterthought!
  - Test early and test often
- In the case of Zephyr, this is often **MCUBoot**, though not always
- Secure means **immutable**
- Should only run valid **signed**, and ideally **versioned** images
- May include **rollback protection** (`MCU_DOWNGRADE_PROTECTION`w/MCUBoot)
- Image contents and signature **must be verified** every reset
- Should support **image encryption** for safer firmware delivery
- May include limited HW recovery option (serial recovery on GPIO pin on MCUBoot)

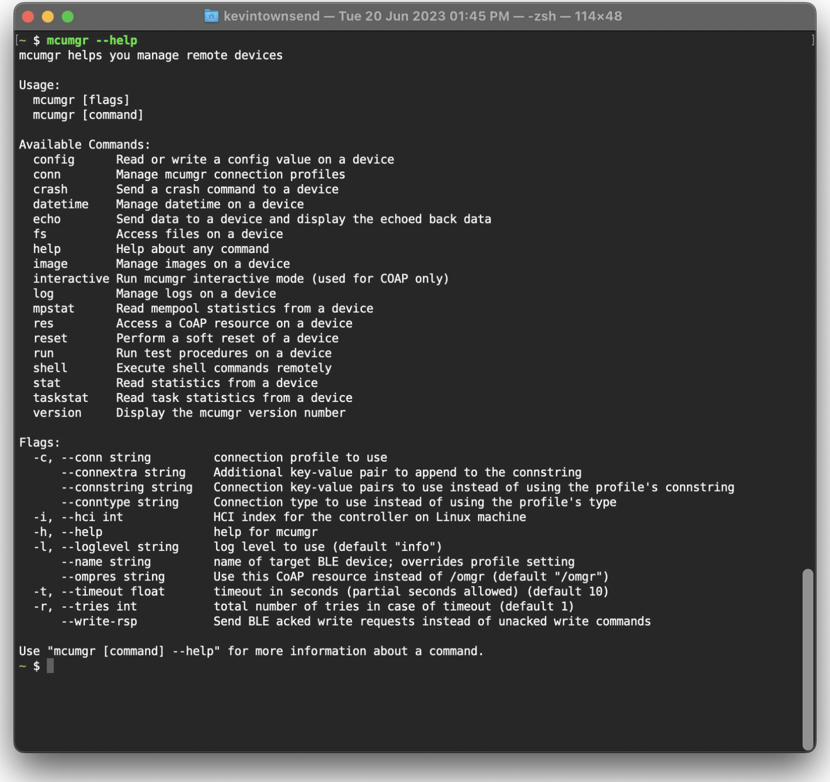
⚠️ Secure boot requires **protecting the bootloader flash region** from overwrites!

⚠️ Must disable **SoC device-recovery** and **debug** interfaces on the MCU!

MCUBoot: mcumgr

- MCUBoot CLI management tool
- Multi transport: Serial, BLE, UDP
- Extensible command set:
  - Set datetime
  - Update file system
  - Get thread/device stats
  - Reset device
  - Shell access

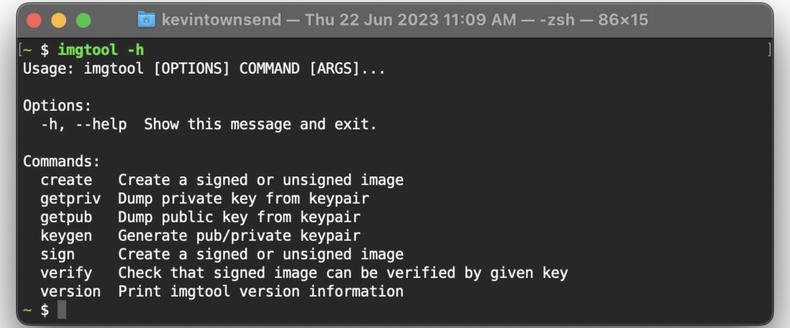
⚠️ The optional commands are a double-edged sword and need to be evaluated against your deployment scenario!



```
~ $ mcumgr --help
mcumgr helps you manage remote devices

Usage:
  mcumgr [flags]
  mcumgr [command]

Available Commands:
  config      Read or write a config value on a device
  conn        Manage mcumgr connection profiles
  crash       Send a crash command to a device
  datetime    Manage datetime on a device
  echo        Send data to a device and display the echoed back data
  fs          Access files on a device
  help        Help about any command
  image       Manage images on a device
  interactive Run mcumgr interactive mode (used for COAP only)
  log         Manage logs on a device
  mpstat      Read mempool statistics from a device
  res         Access a CoAP resource on a device
  reset       Perform a soft reset of a device
  run         Run test procedures on a device
  shell       Execute shell commands remotely
  stat        Read statistics from a device
  taskstat    Read task statistics from a device
  version     Display the mcumgr version number

Flags:
  -c, --conn string          connection profile to use
      --connextra string     Additional key-value pair to append to the connstring
      --connstring string    Connection key-value pairs to use instead of using the profile's connstring
      --conntype string      Connection type to use instead of using the profile's type
  -i, --hci int              HCI index for the controller on Linux machine
  -h, --help                 help for mcumgr
  -l, --loglevel string      log level to use (default "info")
      --name string          name of target BLE device; overrides profile setting
      --ompres string        Use this CoAP resource instead of /omgr (default "/omgr")
  -t, --timeout float        timeout in seconds (partial seconds allowed) (default 10)
  -r, --tries int            total number of tries in case of timeout (default 1)
      --write-rsp            Send BLE acked write requests instead of unacked write commands

Use "mcumgr [command] --help" for more information about a command.
~ $
```

MCUBoot: imgtool

- Generates correctly-formatted keys
  `$ imgtool keygen -k sign_p256.pem -t ecdsa-p256`
- Signs images
- Can be used to verify signatures
- Get C-friendly public/private key data:
  ```
  $ imgtool getpriv -k sign_p256.pem
  $ imgtool getpub -k sign_p256.pem
  ```

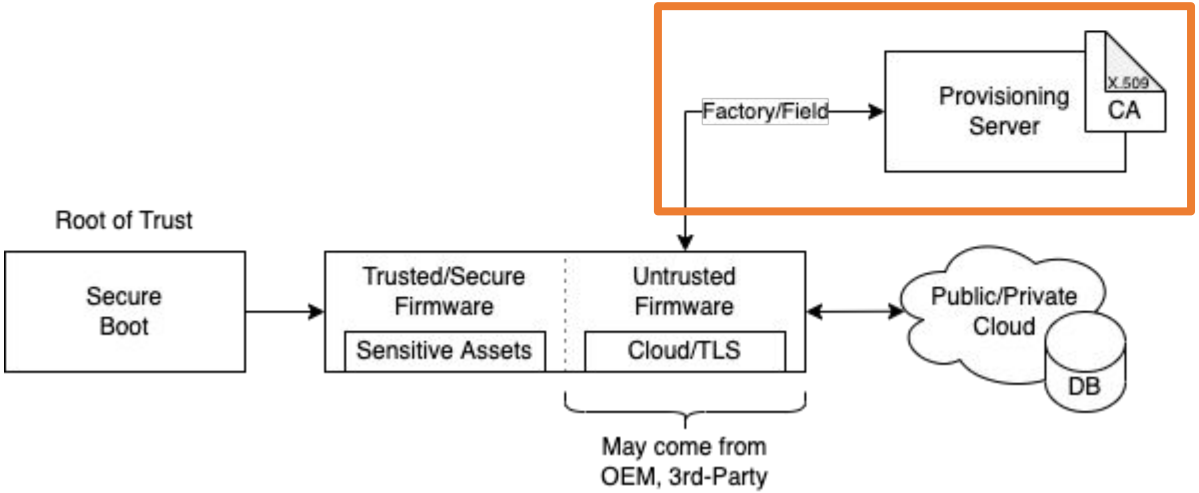


```
~ $ imgtool -h
Usage: imgtool [OPTIONS] COMMAND [ARGS]...

Options:
  -h, --help  Show this message and exit.

Commands:
  create   Create a signed or unsigned image
  getpriv  Dump private key from keypair
  getpub   Dump public key from keypair
  keygen   Generate pub/private keypair
  sign     Create a signed or unsigned image
  verify   Check that signed image can be verified by given key
  version  Print imgtool version information
~ $
```

⚠️ Always generate and safely store **your own private signing key**!
Point the build system to it via `BOOT_SIGNATURE_KEY_FILE`

Device Provisioning

Core Components in a Secure IoT System



Common Provisioning Scenarios

⚠️ There is no **one-size-fits-all** approach here!

❓ Can you use a public cloud provider? EU-only? Where is data stored?

❓ Does everything need to exist behind a company firewall?

❓ What kind and how many keys/certs are required, and how often will they change?

❓ What level of access control is required? Who can provision devices? When/where?

- There are two common scenarios for **provisioning** devices:
  - **Factory provisioning**
    Devices are provisioned in the **factory**, during HW manufacture or when packaging,
  - **Late-binding**
    Devices are provisioned in the **field** by the customer, generally w/an **intermediary tool**

Late-binding is probably the more common scenario in shipping products

Signpost: Open Provisioning Standards

- FIDO Device Onboard 1.1

  “An automatic onboarding protocol for IoT devices. Permits late binding of device credentials, so that one manufactured device may onboard, without modification, to many different IOT platforms.”

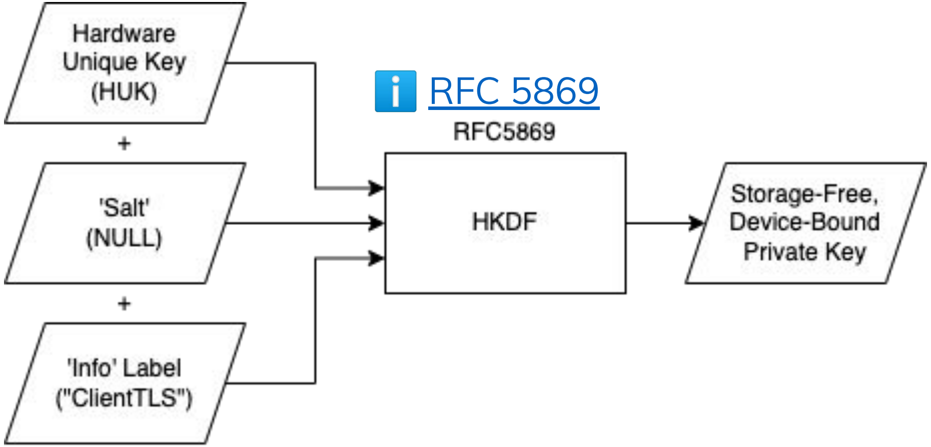
Specification Document

ℹ️ **A simpler solution** may also be appropriate, which we’ll discuss later with the Confidential AI sample.

Best Practice: Storage-Free Key Derivation

- Multiple keys are often required
- Safest way to store a key is never store it!
- **Device-bound** keys derived w/HUK
- Key(s) get regenerated at boot
- Persistent across FW updates
- Ties encryption/auth/etc to MCU

ℹ️ RFC 5869


Hardware Unique Key (HUK)
+
'Salt' (NULL)
+
'Info' Label ("ClientTLS")
RFC5869
HKDF
Storage-Free, Device-Bound Private Key

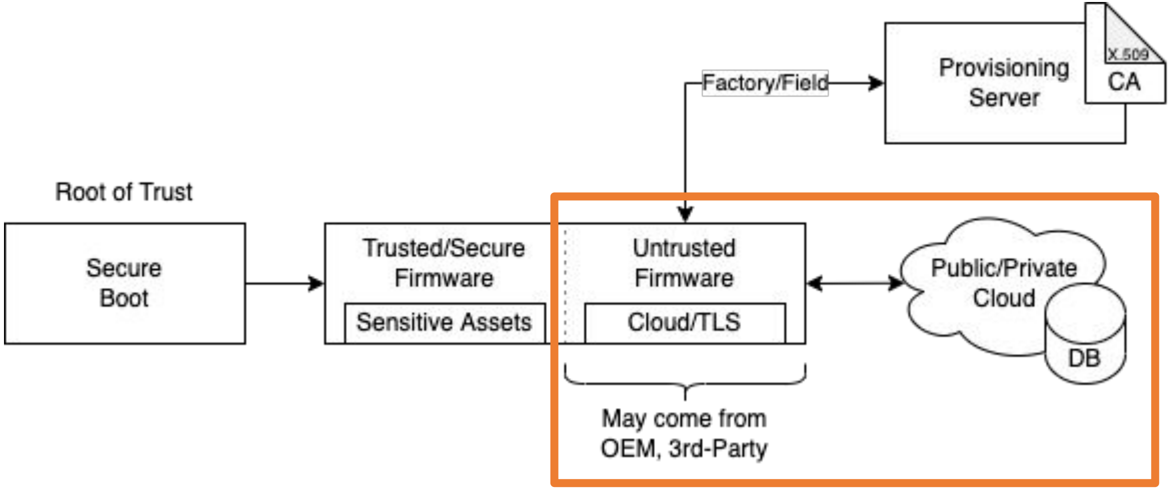
⚠️ This approach requires that the Hardware Unique Key is protected!

⚠️ With S/NS firmware, always prepend a value to the ‘Info’ label in S!

ℹ️ This same approach can also be used to derive a device-bound UUID!

Securing Data in Transit

Core Components in a Secure IoT System



tl;dr: Just use TLS!

- If you can use it, TLS is your best line of defense for **connection-based** data
- Universally adopted, reliable, and based on modern encryption standards
- Explicitly enforce a recent version (>= TLS 1.2) where possible

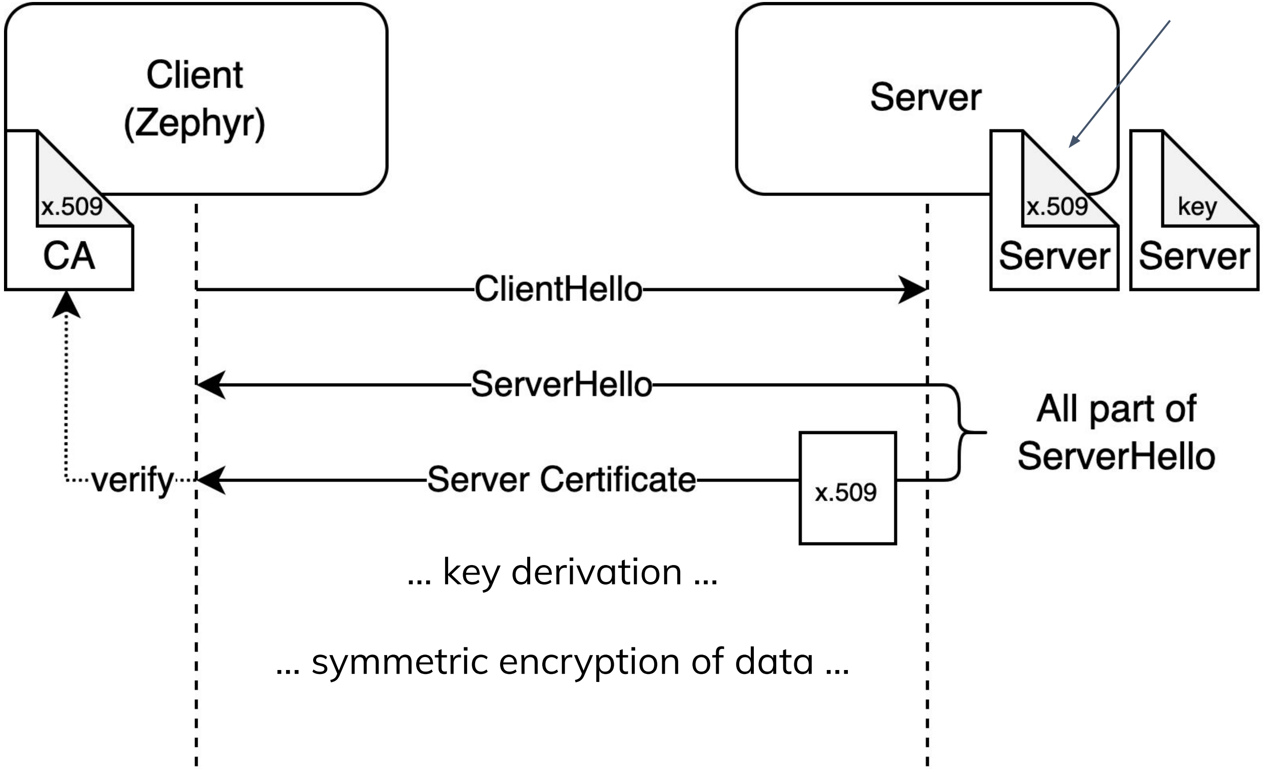
Basic TLS example:
samples/net/sockets/http_get

Basic TLS

- Basic TLS authentication validates the **SERVER** identity
- The certificate exchange gives us a certain degree of confidence in **who we're talking to**, based on the trust we place in the **certificate authority**

Basic TLS

Signed by a Trusted CA



... key derivation ...

... symmetric encryption of data ...

Contains the CA's public key to verify the server certificate's signature

Best Practice: Mutual TLS

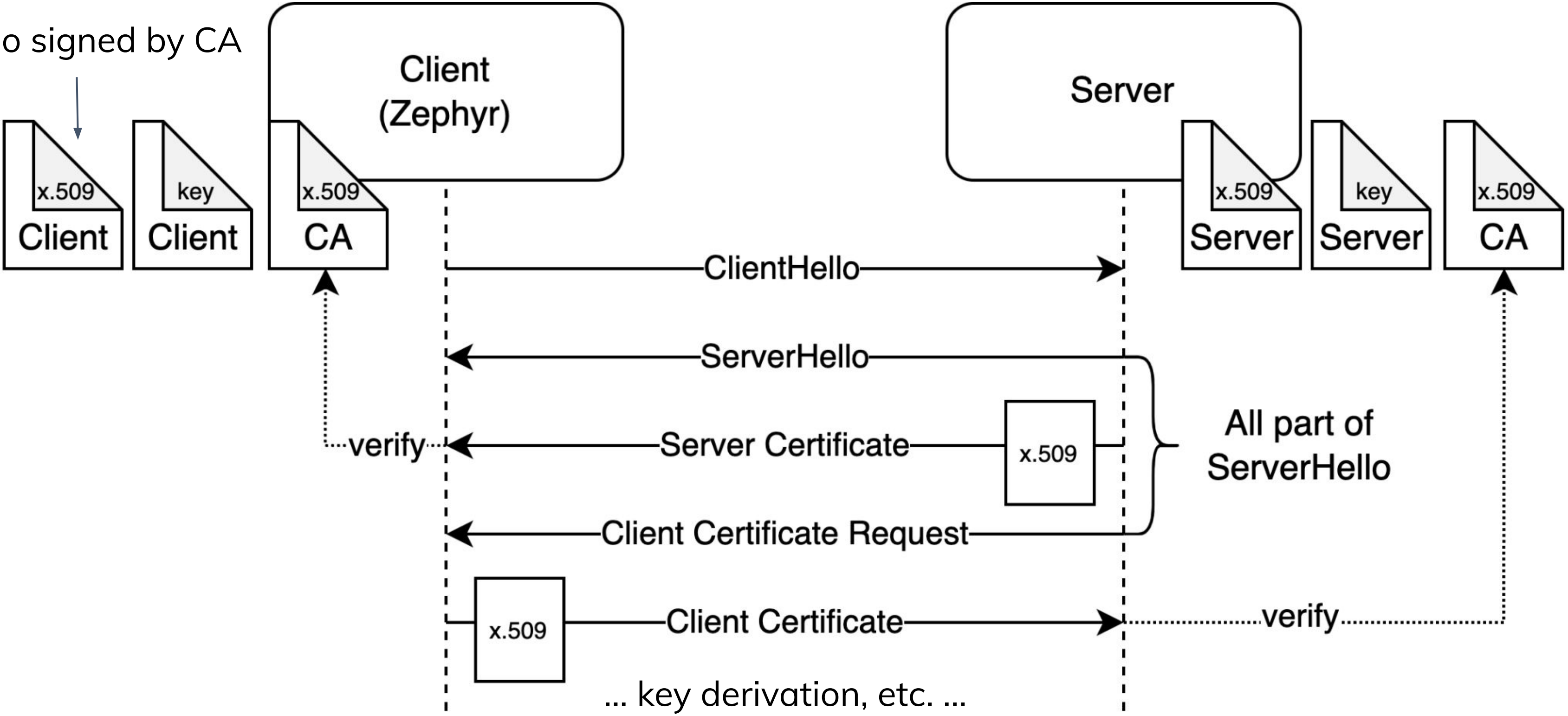
- How does the server know it’s talking to a trusted client device?
- TLS optionally includes **Client Authentication**, where the server also asks the client device to provide proof of it’s identity

⚠️ Mutual TLS is part of the TLS standard, but commercial cloud provider support levels vary!

Some providers (Azure IoT Hub, for example) have better X.509 client certificate authentication support than others for.

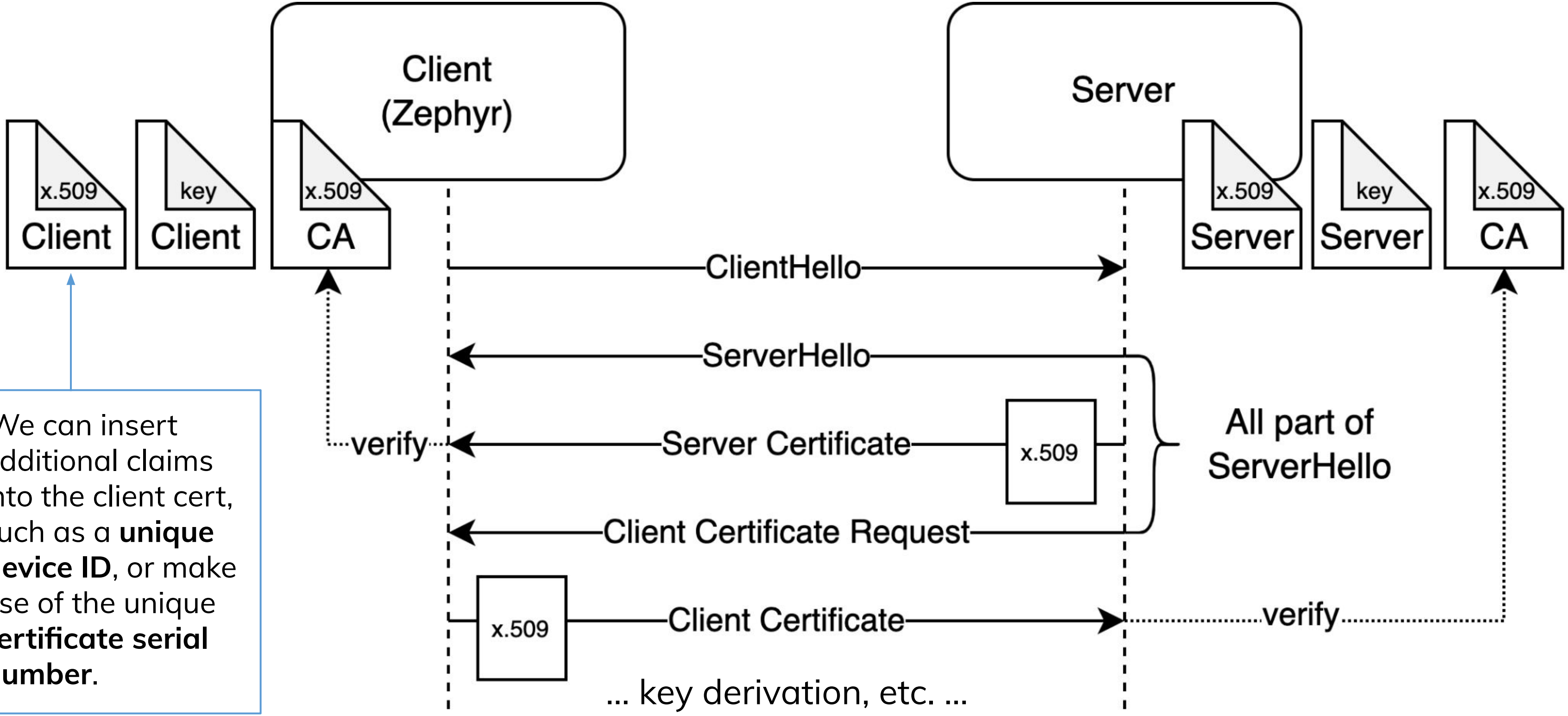
Mutual TLS

Also signed by CA



... key derivation, etc. ...

Basic 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**.

Mutual TLS Sample Code

- **Certificate generation script** and **TCP server** sample code: gist.github.com/microbuilder/cf928ea5b751e6ea467cc0cd51d2532f

ZDS 2022: X.509 Client Authentication in Zephyr
https://www.youtube.com/watch?v=8-PU9_ONSrY

Securing Data at Rest

COSE Payload Encryption

- **What do we do when TLS isn’t available?**
- Examples: Secrets in external flash, intermediary broker app (BLE), etc.
- Securing data at rest is less of a **solved problem** today
- COSE the only open, embedded-appropriate std I’m aware of for data at rest
- COSE is built on top of **CBOR**, which is essentially **binary JSON**
- COSE allows for **signing** and **encryption** of **data at rest** using modern cyphers
- Should be actively promoted as a solution to securing data at rest
- COSE encryption less common than signing today
  - Poor ENCRYPT/ENCRYPT0 library support
  - C libraries like t_cose are making an effort to improve this, but an active WIP
- ⚠️ No ‘profiles’ in COSE, so you need to know what you’re doing piecing things!

COSE Payload Encryption

- Rust PoC of using COSE for encrypting data at rest (‘**flow**’, **CEDAR**):
  https://github.com/Linaro/lite-flow/tree/main

Our initial proposal for **Efficient COSE encryption**:

LHR23-313: Secure IoT Data Flow

David Brown, Linaro Connect 2023

https://resources.linaro.org/en/resource/k9iN8sdePtvXWTN1mP1tKP

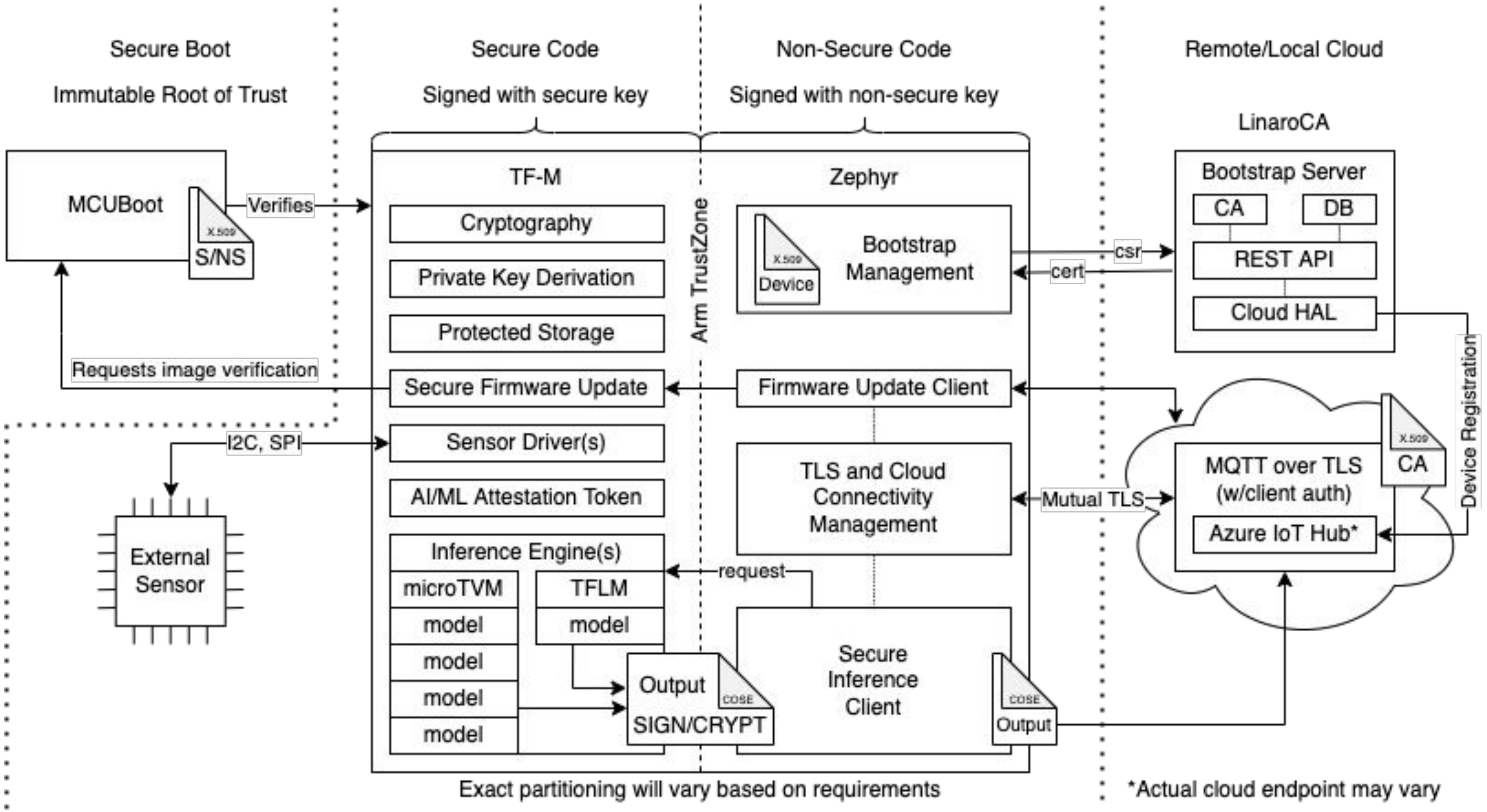
Example: Confidential AI

What is ‘Confidential AI’?

- An attempt to demonstrate **end-to-end security best practices**
- … based on **modern Cortex-M hardware** (v8-M, TrustZone, etc.)
- … using **open source software** and **open standards**
- … with **AI/ML workloads** as a test case

⚠️ There’s nothing magic about the AI/ML component! Engineering outputs just tend to improve when you have a **clear, specific** problem to solve.

Confidential AI System Architecture



Exact partitioning will vary based on requirements

*Actual cloud endpoint may vary

Application Code and Component Repositories

← Confidential AI Proof of Concept Application:
https://github.com/Linaro/zephyr_confidential_ai

Open Source Components:

● LITE Bootstrap	https://github.com/Linaro/lite_bootstrap_server
● MCUBoot	https://github.com/mcu-tools/mcuboot
● TF-M	https://git.trustedfirmware.org/TF-M/trusted-firmware-m.git/
● Zephyr RTOS	https://github.com/zephyrproject-rtos/zephyr
● MicroTVM	https://tvm.apache.org/docs/topic/microtvm/index.html
● TFLM	https://www.tensorflow.org/lite/microcontrollers
● MbedTLS	https://github.com/Mbed-TLS/mbedtls
● COSE	https://github.com/laurencelundblade/t_cose

Project contact details: confidential_ai@linaro.org

Checklist

Checklist

- Integrate bootloader early on!
- Understand your provisioning requirements
- Replace default keys from day one, even during dev!
- Plan for key storage (harder to leak keys you never store!)
- Streaming data? Use TLS 1.2!
- Using TLS? Don’t needlessly reinvent client auth!
- Data at rest still a WIP, but COSE is the standard to watch

Thank you

Linaro