CCA – COMPETENCE CENTRE

HTL Anichstraße

noindent

DIC-serial_crypto

Fabio Plunser

31. März 2021

Inhaltsverzeichnis

Abbildungsverzeichnis

Code

1 Aufgabenstellung

Die Aufgabe ist es, im echtzeit Betriebssystem Zephyr einen Krypto Prozessor zu programmieren, der einen Verschlüsselten Text erhält und mit AES-128 cbc entschlüsselt. Der Prozessor wird mit dem **nativ_posix-Board** programmiert. Dieses kann in eine normal ausführbare Datei kompiliert werden, die man auf einem Linux System ausführen kann. Somit wird ein Mikronroller Board emuliert.

1.1 Aufgaben und Eigenschaften des Krypto Prozessors

Der Krypto Prozessor soll in 4 Threads, **main, uart-in, uart-out, processing** aufgeteilt werden. Weiterhin soll die vorgegebene Statemachine und UART Protokoll implementiert werden. Die Statemachine gibt vor in wann das Programm was machen soll.

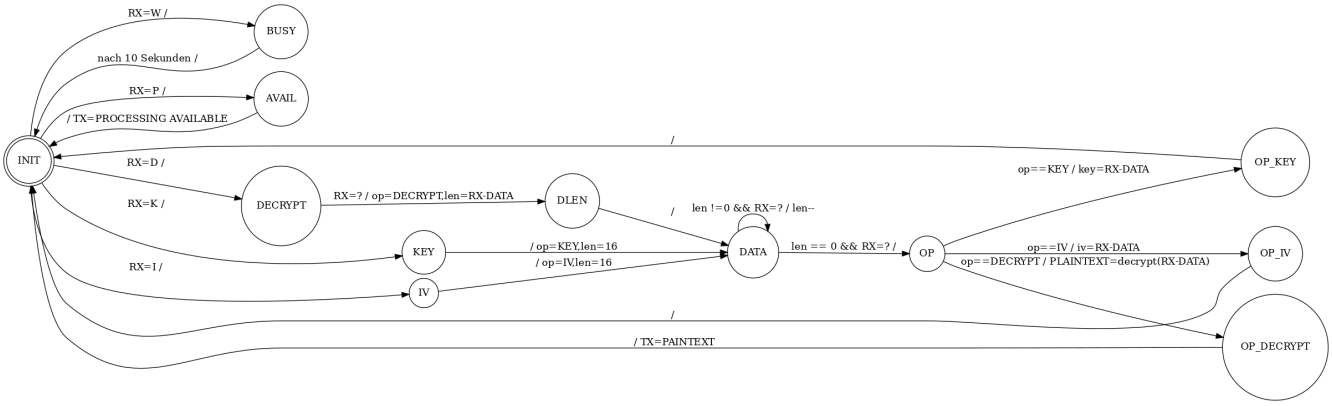

Abbildung 1: Statemachine

Weiterhin wurde ein UART Protokoll vorgegeben:

- alive: Wenn ein ’.’ empfangen wird, soll sofort ein ’.’ zurückgeschickt werden.
- avail: Wenn ein ’P’ empfangen wird, soll vom processing-thread „PROCESSING AVAI“ zurückgeschickt werden.
- key: Wenn ein ’K’ empfangen wird, folgen 16 Byte Des AES-128 Schlüssel, dieser empfangene Schlüssel wird in den Kryptoprozessor geladen.
- iv: Wenn ein ’I’ empfangen wird, folgen 16 Byte des AES-128 IV, , dieser empfangene IV wird in den Kryptoprozessor geladen.
- Decrypt: Wenn ein ’D’ empfangen wird, gefolgt von der Länge des Ciphertextes, gefolgt vom Ciphtertext, wird dieser Ciphertext mit dem entsprechenden Key und IV mit AES128-CBC entschlüsselt und als Plaintext an der UART ausgegeben. Wenn der Ciphtertext nicht durch 16 Teilbar ist, soll eine Fehlermeldung „XERROR“ zurückgesendet werden.

Das Programm soll alle Tests der vorgegebenen test.py Datei erfolgreich absolvieren. Die Tests, testen ob die Statemchine korrekt implementiert wurde und besteht aus folgende Test:

- Test0: Testung der UART Verbindung, indem ein '.' Punkt an den Prozessor geschickt wird.
- Test1: Testung der availibility, indem ein 'P' an den Prozessor geschickt wird.
- Test2: Testung ob der Processor korrekt blockiert
- Test3: Testung ob ein Error vom Prozessor zurückgeschickt wird, wenn ein absichtlich nicht funktionierender Ciphertext an den Prozessor geschickt wird, da dieser nicht durch 16 Teilbar ist.
- Test4: Testung ob die standard Konfiguration der Entschlüsselt korrekt ist.
- Test5: Testung ob ein anderer Key und IV von dem Prozessor übernommen wird.

2 Theorie und Vorwissen

2.1 Zephyr

Zephyr ist ein Open-Source-Echtzeitbetriebssystem welches von der Linux Foundation.[1] Ein Echtzeitbetriebssystem, real-time operating system **RTOS** ist ein Betriebssystem, das Echtzeit-Anforderungen erfüllen kann. Das bedeutet, dass Anfragen eines Anwendungsprogramms innerhalb einer Voraus bestimmbaren Zeit gesichert verarbeitet werden.[2]
Zephyr wurde mit dem Getting-Started-GUID Linux Subsystem von Windows installiert. Um ein Zephyr Projekt zu kompilieren wird Zephyr eigenes **West**[3] verwendet.
West ist ein Kompilierungs-Tool von Zephyr. Es verwendet Ninja und CMake um das Projekt zu kompilieren. West wird folgendermaßen verwendet, um ein Projekt zu kompilieren:

```
1 west build -p auto -b nativ_posix_64
```

Listing 1: West Beispiel

2.1.1 KConfig

Kernel Configuration File[4]ist die prj.conf Datei in einem Zephyr Projekt. In diesem werden bestimmte Konfigurationen, Funktionen und „Geräte", wie z.b. *CONFIG_SERIAL=y* aktiviert.

[1]Quelle: https://de.wikipedia.org/wiki/Zephyr_(Betriebssystem)
[2]Quelle: https://de.wikipedia.org/wiki/Echtzeitbetriebssystem
[3]https://docs.zephyrproject.org/2.4.0/guides/west/index.html
[4]https://docs.zephyrproject.org/latest/application/index.html?#application-kconfig

2.1.2 Device Tree

Der Device Tree[5] ist in einem Zephyr Projekt eine Datei mit der Endung **.dts** dort stehen alle für das ausgewählte Board verfügbare Geräte drinnen. Im Fall des nativ_posix_64 sieht dieses folgendermaßen aus.

```
/dts-v1/;

/ {
    #address-cells = < 0x1 >;
    #size-cells = < 0x1 >;
    model = "Native POSIX Board";
    compatible = "zephyr,posix";
    chosen {
        zephyr,console = &uart0;
        zephyr,shell-uart = &uart0;
        zephyr,uart-mcumgr = &uart0;
        zephyr,flash = &flash0;
        zephyr,entropy = &rng;
        zephyr,flash-controller = &flashcontroller0;
        zephyr,ec-host-interface = &hcp;
    };
    aliases {
        eeprom-0 = &eeprom0;
        i2c-0 = &i2c0;
        spi-0 = &spi0;
        led0 = &led0;
    };
    leds {
        compatible = "gpio-leds";
        led0: led_0 {
            gpios = < &gpio0 0x0 0x0 >;
            label = "Green LED";
        };
    };

    ...

    };
    uart0: uart {
        status = "okay";
        compatible = "zephyr,native-posix-uart";
        label = "UART_0";
        current-speed = < 0x0 >;
    };

    ...

};
```

Listing 2: West Beispiel

[5] https://docs.zephyrproject.org/latest/guides/dts/intro.html
https://docs.zephyrproject.org/latest/reference/devicetree/index.html#devicetree

2.1.3 Tinycrypt

2.2 Linux Pseudo-Terminal

2.3 Threads

2.4 Message-Queue

3 Programm Umsetzung