CCA – COMPETENCE CENTRE

HTL Anichstraße

DIC-serial_crypto

Fabio Plunser

14. April 2021

Inhaltsverzeichnis

Abbildungsverzeichnis

Code

1 Information

Vielen dank an David Reiser, der mir sein Makefile zur Verfügung gestellt hat.
Weiterhin kann das komplette Projekt in folgender GITHUB Repository nachvollzogen werden: https://github.com/FabioPlunser/DIC-Lezuo/

2 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.

2.1 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.

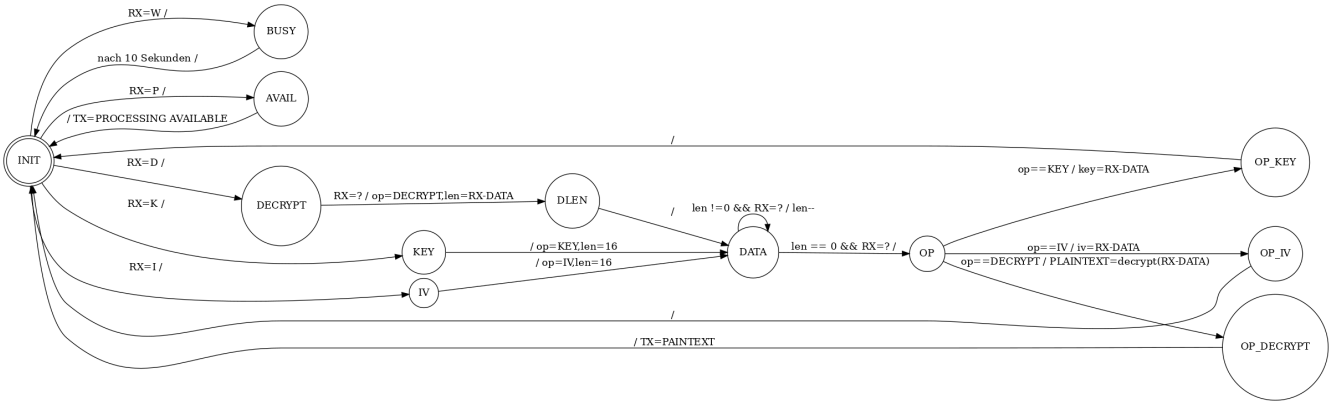

Abbildung 1: Statemachine

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.

3 Theorie und Vorwissen

3.1 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

[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

3.1.1 west

3.1.1.1 West als Repo Verwalter

West ist ein Command-line tool von Zephyr. Es wird unabhängig von Zephyr installiert und wird zur Kompilierung und Flashen verwendet. Das Problem das bei solchen Projekten entsteht ist, dass es sich aus mehreren GIT Repositories mit unterschiedlichen Versionen zusammen setzen kann, somit muss dafür gesorgt werden, dass alle GIT Repositories, mit der korrekter Version im korrekten pfad sich befinden. Diese werden im sogenannten *west manifest* **west.yaml** festgelegt.[4]

```yaml
manifest:
defaults:
  remote: upstream

remotes:
  - name: upstream
  url-base: https://github.com/zephyrproject-rtos

#
# Please add items below based on alphabetical order
projects:
  - name: cmsis
  revision: c3bd2094f92d574377f7af2aec147ae181aa5f8e
  repo-path: mcuboot
  path: modules/tee/tfm-mcuboot
  revision: 1.7.0-rc1
  ...
self:
  path: zephyr
  west-commands: scripts/west-commands.yml

```

Listing 2: west.yaml

Um einen Ordner zu initialisieren wird **west init** und zum downloaden der Daten **west update** verwendet.[5]

3.1.1.2 West als Compiler

West ist der Haupt-Compiler von Zephyr, west ruft cmake, ninja oder make im Hintergrund auf um erfolgreich zu Kompilieren. Im Hintergrund wird hauptsächlich ninja verwendet.[6]

[4] https://docs.zephyrproject.org/latest/guides/west/index.html
[5] https://github.com/zephyrproject-rtos/west
[6] https://docs.zephyrproject.org/latest/application/index.html?highlight=ninja

3.1.2 ninja

Ninja ist ein kleines build System mit Fokus auf Geschwindigkeit. Es ist in assembler geschrieben und ist so designend, dass die Input files von einem höheren Build-System erstellt werden und das es so schnell wie möglich builded. Das Problem, das die Ninja entwickler mit GNU Make hatten, ist das es durch die Higher-Level Programmiersprache für große Projekte recht viel Zeit benötigt. Durch die Implementierung in Assembler steigt die Geschwindigkeit stark und ist somit sehr gut für große Projekte geeignet, jedoch steigt auch die Komplexität.

3.1.3 KConfig

Kernel Configuration File[7]ist die prj.conf Datei in einem Zephyr Projekt. In diesem werden besstimmte Konfigurationen, Funktionen und „Geräte“, wie z.b. *CONFIG_SERIAL=y* aktiviert. Das Ziel ist es alle im Kernel beinhaltete Funktionen, Applikations spezifisch, zur Verfügung zu stellen, ohne den Source Code ändern zu müssen. Somit können Funktionen und Schnittstellen einfach aktiviert werden. Welche Schnittstellen für ein board aktiviert werden können, kann im Device Tree gefunden werden.

[7] `https://docs.zephyrproject.org/latest/application/index.html?#application-kconfig`

3.1.4 Device Tree

Der Device Tree[8] 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 3: West Beispiel

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

3.1.5 Tinycrypt

Die TinyCrypt-Bibliothek bietet eine Implementierung für eingeschränkte Geräte von minimalen Standard-Kryptographie-Grundelementen. Di Bibliothek ist von Intel[9] und wurde in zephyr mit der implementiert.[10] Innerhalb von Zephyr können die eignenen Crypto API Befehle oder die direkten TinyCrypt Befehle verwendet werden.

3.2 Linux Pseudoterminal

Ein Pseudoterminal ist ein Dienst der eine bidirektionale Pipe, aufbaut. Sie werden verwendet um ein physisches Terminal zu emulieren. Im Fall von Zephyr mit dem nativ_posx Board wird ein Pseudoterminal verwendet um mit dem Baord zu kommunizieren. Der Pfad dieses Terminals ist /dev/pts/.

3.3 Threads

Innerhalb eines Betriebssystems werden Applikationen und Abläufe als Prozesse realisiert. Prozesse werden vom Schedular verwaltet. Um nun die Abarbeitung dieser Prozesse zu beschleunigen, besteht ein ein Prozess aus mehreren parallel laufenden Threads. Diese Threads beinhalten einen Teil des Codes des Prozesses, dadurch können große Probleme entstehen. Es muss sichergestellt werden, dass Threads nie auf die gleichen Variablen/Ressourcen zugreifen, da ansonsten *race-Conditions* entstehen.

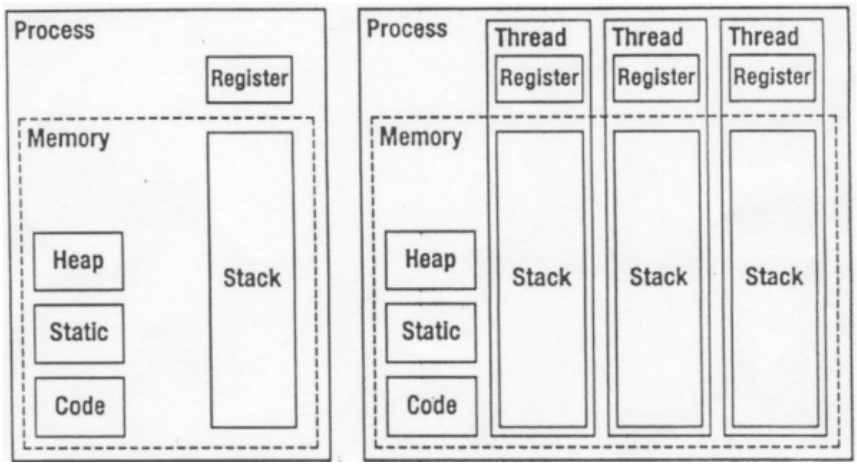

Abbildung 2: Process-Threads

[9] https://github.com/intel/tinycrypt
[10] https://docs.zephyrproject.org/2.3.0/guides/crypto/tinycrypt.html?highlight=tinycrypt

3.4 Message-Queue

Eine Message-Queue ist ein besonderer Buffer, ein FIFO-Buffer (First-In-First-Out-Buffer). Das bedeutet, es werden Nachrichten in einer Reihe in den Buffer geschrieben und es kann nur die erste Nachricht in der Reihe herausgenommen werden, dabei wird diese nachricht im Buffer gelöscht und die nächste Nachricht rückt nach. Solche Message-Queues werden verwendet, um zwischen Threads die Daten korrekt auszutauschen. Da eine Nachricht beim auslesen gelöscht wird, können nicht mehrere Threads gleichzeitig auf die Nachricht zugreifen.

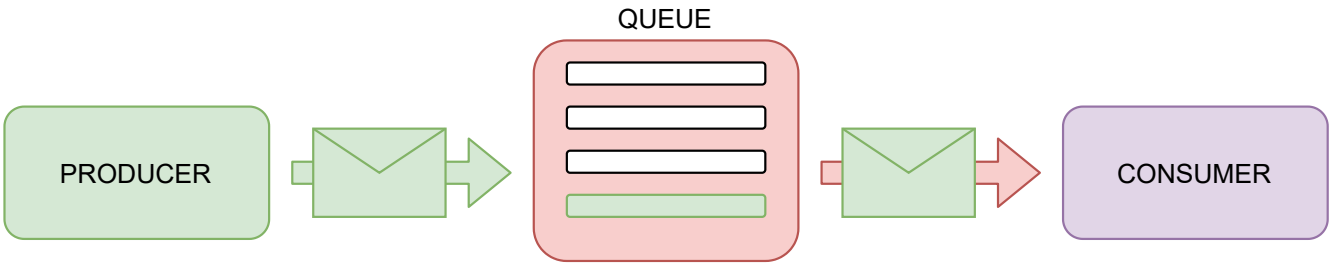

Abbildung 3: Message-Queue-Darstellung

4 Programm Umsetzung

Das Programm wurde wie angegeben in 4 Threads eingeteilt.

- Main-Thread
  - Initialisierung der anderen Threads
  - UART und Crypto Device Initialisierung
  - Validate Hardware Compatibility
  - alle 5 Sekunden ein Lebenszeichen von sich geben.
- UART-IN-Thread
  - Einlesen der UART
  - Statemachine Implementierung
- UART-OUT-Thread
  - Ausgabe der in die Queue geschriebenen Messages
- PROCESS-Thread
  - Verwaltung der Entschlüsselung

4.1 Blockschaltbild

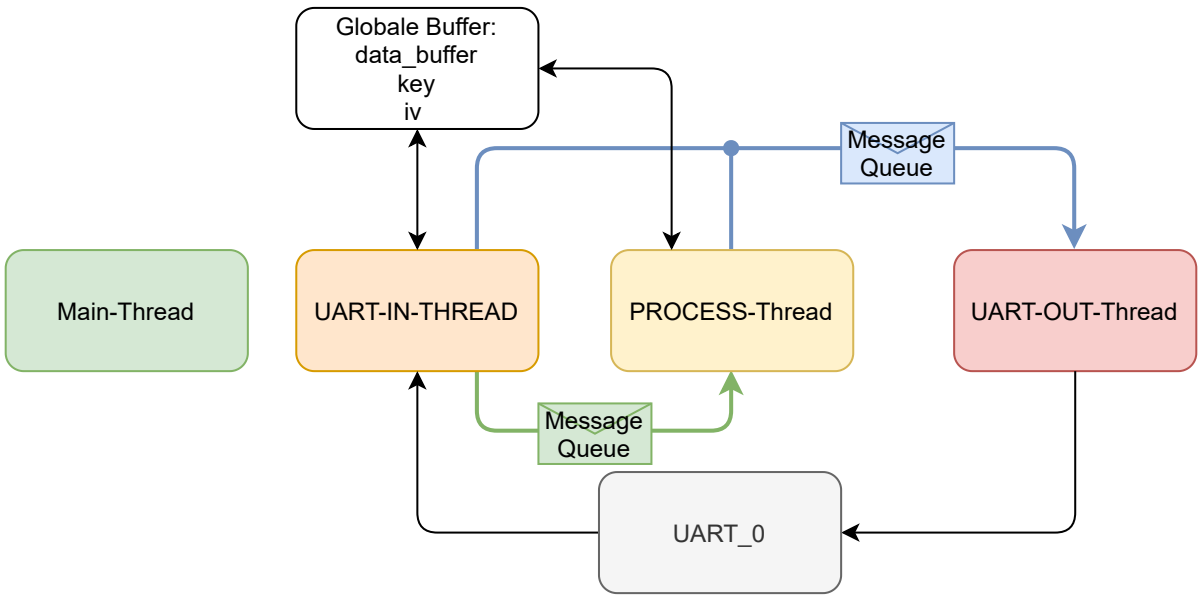

Abbildung 4: Blockschaltbild

4.2 Projekt Konfiguration

Sodass, das nativ_posix Board die benötigten Geräte verwenden kann müssen dieser in der **prj.conf** Datei aktivieren werden.

```
#Configure Serial-Connection
CONFIG_SERIAL=y
CONFIG_UART_NATIVE_POSIX=y
CONFIG_NATIVE_UART_0_ON_OWN_PTY=y

#Crypto
CONFIG_TINYCRYPT=y
CONFIG_TINYCRYPT_AES=y
CONFIG_TINYCRYPT_AES_CBC=y
CONFIG_CRYPTO=y
CONFIG_CRYPTO_TINYCRYPT_SHIM=y
```

Listing 4: prj.conf

4.3 Initialisierung

Es müssen:

- Message-Queue
- UART_0
- Verschlüsselung
- Threads

initialisiert werden.

4.3.1 Message-Queue

Wie in der Zephyr Dokumentation[11] beschrieben wird, kann eine Message-Queue mit einem Macro initialisiert werden. Für den Kryptoprozessor werden zwei Message-Queues verwendet, eine um die Nachrichten an der UART auszugeben und eine weitere um Befehle an den Process-Thread weiterzugeben.

```
K_MSGQ_DEFINE(uart_queue, sizeof(struct uart_message *), q_max_msgs, q_align
    );
K_MSGQ_DEFINE(crypto_queue, sizeof(char* ), q_max_msgs, q_align);
```

Listing 5: Message-Queue-Initialisierung

[11] https://docs.zephyrproject.org/2.3.0/reference/kernel/data_passing/message_queues.html?highlight=queue

4.3.2 UART_0

Um die UART verwenden zu können muss zuerst ein Device „erstellt“ werden. Dieses Gerät muss dan zur richtigen UART_0 *gebinded*/verbunden werden. Danach kann die UART wie in der Dokumentation[12] beschrieben konfiguriert werden. Da die UART in diesem Fall nur mit Pseudo-Terminal verwendet wird, ist die Konfiguration der Baudrate etc. nicht notwendig.

```
1  const struct device * uart_dev;
2  uart_dev = device_get_binding(UART_NAME);
3  if(!uart_dev){
4      printk("UART-binding-error\n");
5  }
6  const struct uart_config uart_cfg = {
7      .baudrate = 115200,
8      .parity = UART_CFG_PARITY_NONE,
9      .stop_bits = UART_CFG_STOP_BITS_1,
10     .data_bits = UART_CFG_DATA_BITS_8,
11     .flow_ctrl = UART_CFG_FLOW_CTRL_NONE
12 };
13
14 if(!uart_configure(uart_dev, &uart_cfg)){
15     printk("UART-config-error\n");
16 }
```

Listing 6: UART-Initialisierung

[12] https://docs.zephyrproject.org/2.3.0/reference/peripherals/uart.html

4.3.3 Verschlüsselung

Die Initialisierung der Verschlüsselund bzw. des Crypto Device funktioniert sehr ähnlich wie bei der UART. Nur statt eine eigenen Konfiguration wird eine Validate_Hardware_Funktion verwendet um zu Überprüfen wie das Crypto Device verwendet werden kann. Diese Funktion wurde vom CBC Beispiel von Zephyr kopiert.

```
const struct device * crypto_dev;
static uint32_t cap_flags;

//bind crpyto
crypto_dev = device_get_binding(CRYPTO_DRV_NAME);
if (!crypto_dev) {
    printk("Crypto-binding-error\n");
    return;
}
//validate hardware for crypto device
validate_hw_compatibility();


int validate_hw_compatibility()
{
  uint32_t flags = 0U;
  flags = cipher_query_hwcaps(crypto_dev);
    if ((flags & CAP_RAW_KEY) == 0U) {
            printk("Please provision the key separately "
                   "as the module doesnt support a raw key\n");
            return -1;
    }

    if ((flags & CAP_SYNC_OPS) == 0U) {
            printk("The app assumes sync semantics. "
              "Please rewrite the app accordingly before proceeding\n");
            return -1;
    }

    if ((flags & CAP_SEPARATE_IO_BUFS) == 0U) {
            printk("The app assumes distinct IO buffers. "
            "Please rewrite the app accordingly before proceeding\n");
            return -1;
    }
  cap_flags = CAP_RAW_KEY | CAP_SYNC_OPS | CAP_SEPARATE_IO_BUFS;
  return 0;
}
```

Listing 7: UART-Initialisierung

4.3.4 Threads

Die Threads werden innerhalb des Main-Thread initialisiert. Dabei sind die Threads in einem Array und werden nacheinander gestartet.

```
1  pthread_t thread_id[Number_of_threads];
2  init_threads(thread_id);
3
4  void init_threads(pthread_t* thread_id)
5  {
6    //init threads
7    int i, thread_ok;
8    void*(*threads[])(void*) = {uart_in_thread, uart_out_thread,
       process_thread};
9    for(i=0; i<Number_of_threads; i++)
10   {
11     thread_ok = pthread_create(&thread_id[i], NULL, threads[i], NULL);
12     if(thread_ok != 0)
13     {
14       printk("Thred creation Error\n");
15     }
16   }
17 }
```

Listing 8: UART-Initialisierung

4.4 UART-IN-Thread

Im UART-IN-Thread ist die Statemachine implementiert. Somit steuert dieser Thread alle Vorgänge im Prozessor. Die States werden mittels einer Switch-Case abgefragt und gesetzt. Um die States übersichtlich zu setzen wurde eine Enumeration verwendet.

```
1  enum state{ INIT, ALIVE, AVAIL, KEY, IV, DECRYPT, DLEN, DATA,
     SELECT_OPERATION};
2  enum op{OP_KEY,OP_IV,OP_DECRYPT};
```

Listing 9: Statemachine-Enumerations

```
1  void* uart_in_thread(void * x){
2      state_machine();
3      return x;
4  }
```

Listing 10: UART-IN-Thread

```
void state_machine()
{
    uint8_t i = 0;
    uint8_t uart_in;
    uint8_t* data_buffer = "";
    printk("In State Machine\n");
    while(1)
    {
        switch(st_state){
            case INIT:
                if(!uart_poll_in(uart_dev, &uart_in)){
                    switch(uart_in){
                        case 'w':
                        case 'W':
                            put_message_in_crypto_queue("W\n");
                            break;
                        ...
                    }}
                break;
            ...
            Pseudo-Code
            case DECRYPT
            //set busy flag, set op=OP_KEY, goto DLEN
            case IV
            //allocate data_buffer, set op=OP_IV, goto DATA
            case KEY
            //allocate data_buffer, set op=OP_KEY, goto DATA
            case DLEN
            //allocate data_buffer for data + iv, copy iv into buffer, move
               pointer from buffer where data should start, goto data
            CASE DATA
            //get data from uart and put it into data_buffer, goto
               SELECT_OPERATION
            case SELECT_OPERATION:
                switch(operation)
                {
                    case OP_KEY:
                    //copy key from data_buffer into global key variable,
                       goto state = INIT
                    case OP_IV:
                    //copy iv from data_buffer into global iv variable, goto
                        state = INIT
                    case OP_DECRYPT
                    //reset pointer from data_buffer, copy data_buffer into
                       a global buffer, goto state = INIT
                }
        }
    }
}
```

Listing 11: Statemachine

4.5 UART-Out-Thread

Der UART-OUT-Thread sendet die Daten, die in der UART-Message-Queue stehen. Da die Daten für die finalen Tests teilweise mit Nullterminierung geschickt werden müssen wurde für die UART-Message-Queue ein Struct erstellt um die länge der Message festzulegen, da *strlen()* nur bis zur Nullterminierung zählt.

```
1 struct uart_message{
2     unsigned char* message;
3     uint32_t len;
4 };
```

Listing 12: uart_message-struct label

```
1  //put string into uart queue
2  int put_message_in_uart_queue(unsigned char* str, uint32_t len)
3  {
4      static struct uart_message message;
5      message.message = str;
6      message.len = len;
7      struct uart_message * message_pointer = &message;
8
9      if(k_msgq_put(&uart_queue, &message_pointer, K_FOREVER)!=0){
10         printk("Couldn't put message in queue!!\n");
11     }
12     return 0;
13 }
14 //send uart messages from queue
15 void* uart_out_thread(void * x)
16 {
17   int i=0;
18   uint32_t len;
19   struct uart_message * message;
20   unsigned char* message_temp;
21   while(1)
22   {
23     if(!k_msgq_get(&uart_queue, &message, K_NO_WAIT)) {
24       len = message->len;
25       message_temp = message->message;
26       while(i < len)
27       {
28         uart_poll_out(uart_dev, message_temp[i++]);
29       }
30       i = 0;
31     }
32   }
33   return x;
34 }
```

Listing 13: UART-OUT-Thread

4.6 Processing Thread

Der Processing-Thread started die Entschlüsselung und verwaltet, wie in der Angabe beschrieben, das Senden vom "Processing-Availableünd Blocken des Threads.

```
1  void * process_thread(void * x)
2  {
3    unsigned char* message;
4    while(1)
5    {
6      if(!k_msgq_get(&crypto_queue, &message, K_NO_WAIT)) {
7
8        switch (message[0])
9        {
10       case 'D':
11         printk("Process_thread: Decrypting\n");
12         printk("Ciphertext: %02X\n", cbc_buffer);
13         processing_busy = true;
14         if(cbc_mode())
15         {
16           format_plaintext_for_comparison(out_buffer);
17         }
18         processing_busy = false;
19         break;
20       case 'P':
21         if(processing_busy == false){
22           put_message_in_uart_queue("PROCESSING AVAILABLE\n", strlen("
                PROCESSING AVAILABLE\n"));
23         }
24         break;
25       case 'W':
26         processing_busy = true;
27         sleep(10);
28         processing_busy = false;
29         break;
30
31
32       default:
33         break;
34       }
35     }
36   }
37   return x;
38 }
```

Listing 14: Processing-Thread

4.6.1 Entschlüsselung

Für die Entschlüsselung sind standard IV und Key und ein Input und Output Buffer nötig. Diese werden global und in der Statemachine entsprechend der Angabe gesetzt.

```
1  //set default ciphertext BBBBBBBBBBBBBBBB
2  static uint8_t iv[AES_IV_LEN] ={
3      0x42,0x42,0x42,0x42,
4      0x42,0x42,0x42,0x42,
5      0x42,0x42,0x42,0x42,
6      0x42,0x42,0x42,0x42,
7  };
8  //set default key, with care so that iv and key are not overwriting each
       other
9  uint8_t* key = iv;
10 static uint8_t* cbc_buffer;
11 static uint8_t* out_buffer;
12
13 int cbc_mode()
14 {
15   uint32_t  in_buffer_len = len + AES_IV_LEN;
16   uint32_t out_buffer_len = len;
17   out_buffer = malloc(out_buffer_len);
18   struct cipher_ctx ini = {
19     .keylen = AES_KEY_LEN,
20     .key.bit_stream = key,
21     .flags = cap_flags,
22   };
23   struct cipher_pkt decrypt = {
24     .in_buf = cbc_buffer,
25     .in_len = in_buffer_len,
26     .out_buf = out_buffer,
27     .out_buf_max = out_buffer_len,
28   };
29   if(cipher_begin_session(crypto_dev, &ini, CRYPTO_CIPHER_ALGO_AES,
         CRYPTO_CIPHER_MODE_CBC, CRYPTO_CIPHER_OP_DECRYPT)){
30     cipher_free_session(crypto_dev, &ini); put_message_in_uart_queue("XERROR
         \n", strlen("XERROR\n")); return 0;
31   }
32   if (cipher_cbc_op(&ini, &decrypt, cbc_buffer)) {
33     cipher_free_session(crypto_dev, &ini); put_message_in_uart_queue("XERROR
         \n", strlen("XERROR\n")); return 0;
34   }
35   cipher_free_session(crypto_dev, &ini);
36   return 1;
37 }
```

Listing 15: UART-Initialisierung

4.7 Test-Ausführung

```
~crypto-Program$ make run
Running Project
UART_0 connected to pseudotty: /dev/pts/5
*** Booting Zephyr OS build zephyr-v2.5.0-1214-gddf5f607e163  ***
In State Machine
Main-Thread is alive
Main-Thread is alive
Main-Thread is alive
State Machine UART_IN: .
Putting into uart_queue: .

Message in queue: .

State Machine UART_IN: .
Putting into uart_queue: .

Message in queue: .

State Machine UART_IN: .
Putting into uart_queue: .

Message in queue: .

State Machine UART_IN: .
Putting into uart_queue: .

Message in queue: .

State Machine UART_IN: .
Putting into uart_queue: .

Message in queue: .

State Machine UART_IN: .
Putting into uart_queue: .

Message in queue: .

State Machine UART_IN: .
Putting into uart_queue: .

Message in queue: .

State Machine UART_IN: .
Putting into uart_queue: .

Message in queue: .

State Machine UART_IN: P
Putting into uart_queue: PROCESSING AVAILABLE

Message in queue: PROCESSING AVAILABLE

Main-Thread is alive
State Machine UART_IN: W
```

```
peppi@Peppi-Dekstop:/mnt/c/Users/fplun/GoogleDrive/Schule/2020_21/DIC/DIC-Lezuo/2.Semester-serial_crypto/DI
test_00 (__main__.MyTests) ... 000000.000 Q-RX reset_input_buffer
000000.000 TX   0000  2E                                              .
000000.000 RX   0000  2E 0A                                           ..
Replyb'.\n'
000000.000 TX   0000  2E                                              .
000000.001 RX   0000  2E 0A                                           ..
Replyb'.\n'
000000.001 TX   0000  2E                                              .
000000.001 RX   0000  2E 0A                                           ..
Replyb'.\n'
000000.001 TX   0000  2E                                              .
000000.001 RX   0000  2E 0A                                           ..
Replyb'.\n'
000000.001 TX   0000  2E                                              .
000000.001 RX   0000  2E 0A                                           ..
Replyb'.\n'
000000.001 TX   0000  2E                                              .
000000.002 RX   0000  2E 0A                                           ..
Replyb'.\n'
000000.002 TX   0000  2E                                              .
000000.002 RX   0000  2E 0A                                           ..
Replyb'.\n'
000000.002 TX   0000  2E                                              .
000000.002 RX   0000  2E 0A                                           ..
Replyb'.\n'
000000.002 TX   0000  2E                                              .
000000.002 RX   0000  2E 0A                                           ..
Replyb'.\n'
000000.002 TX   0000  2E                                              .
000000.003 RX   0000  2E 0A                                           ..
Replyb'.\n'
ok
test_01 (__main__.MyTests) ... 000000.000 TX   0000  50                                                    P
000001.001 RX   0000  50 52 4F 43 45 53 53 49  4E 47 20 41 56 41 49 4C  PROCESSING AVAIL
000001.001 RX   0010  41 42 4C 45 0A                                    ABLE.
Replyb'PROCESSING AVAILABLE\n'
ok
```

Abbildung 5: Test00-01

```
Main-Thread is alive
State Machine UART_IN: W
State Machine UART_IN: P
State Machine UART_IN: .
Putting into uart_queue: .

Message in queue: .

Putting into uart_queue: BUSY

Message in queue: BUSY

State Machine UART_IN: .
Putting into uart_queue: .

Message in queue: .

Putting into uart_queue: BUSY

Message in queue: BUSY
```

```
test_02 (__main__.MyTests) ... 000000.000 TX   0000  57                                                    W
000000.000 TX   0000  50                                              P
Replyb''
000001.001 TX   0000  2E                                              .
000002.003 RX   0000  2E 0A 42 55 53 59 0A                            ..BUSY.
000002.003 TX   0000  2E                                              .
000003.004 RX   0000  2E 0A 42 55 53 59 0A                            ..BUSY.
000003.004 TX   0000  2E                                              .
000004.005 RX   0000  2E 0A 42 55 53 59 0A                            ..BUSY.
000004.005 TX   0000  2E                                              .
000005.006 RX   0000  2E 0A 42 55 53 59 0A                            ..BUSY.
000005.007 TX   0000  2E                                              .
000006.008 RX   0000  2E 0A 42 55 53 59 0A                            ..BUSY.
000006.008 TX   0000  2E                                              .
000007.009 RX   0000  2E 0A 42 55 53 59 0A                            ..BUSY.
000007.009 TX   0000  2E                                              .
000008.010 RX   0000  2E 0A 42 55 53 59 0A                            ..BUSY.
000008.010 TX   0000  2E                                              .
000009.011 RX   0000  2E 0A 42 55 53 59 0A                            ..BUSY.
000009.012 TX   0000  2E                                              .
000010.012 RX   0000  2E 0A 42 55 53 59 0A 50  52 4F 43 45 53 53 49 4E  ..BUSY.PROCESSIN
000010.012 RX   0010  47 20 41 56 41 49 4C 41  42 4C 45 0A              G AVAILABLE.
ok
```

Abbildung 6: Test02

```
State Machine UART_IN: D
DECRYPT
DATA
Data: 0xAA
Data: 0xE3
OP_DECRYPT
Process_thread: Decrypting
Ciphertext: 4000B60
CBC_Mode:
Key: BBBBBBBBBBBBBBBB@◆Q◆dU
IV: BBBBBBBBBBBBBBBB@◆Q◆dU
Buffer: BBBBBBBBBBBBBBBB◆◆
Buffer 0x4000b60
Keylen: 16, keybitstream: BBBBBBBBBBBBBBBB@◆Q◆dU , flags 50
Putting into uart_queue: XERROR

Message in queue: XERROR
```

```
000010.012 RX   0000  2E 0A 42 55 53 59 0A 50  52 4F 43 45 53 53 49 4E  ..BUSY.PROCESSIN
000010.012 RX   0010  47 20 41 56 41 49 4C 41  42 4C 45 0A              G AVAILABLE.
ok
test_03 (__main__.MyTests) ... Data in Bytes: b'D' Data: 68
000000.000 TX   0000  44                                              D
Data in Bytes: b'\x02' Data: 2
000000.100 TX   0000  02                                              .
Data in Bytes: b'\xaa' Data: 170
000000.201 TX   0000  AA                                              .
Data in Bytes: b'\xe3' Data: 227
000000.301 TX   0000  E3                                              .
Data in Bytes: b'X' Data: 88
000000.402 TX   0000  58                                              X
000001.503 RX   0000  58 45 52 52 4F 52 0A                            XERROR.
Replyb'XERROR\n'
ok
```

Abbildung 7: Test03

```
Key: BBBBBBBBBBBBBBBB@◆Q◆dU
IV: BBBBBBBBBBBBBBBB@◆Q◆dU
Buffer: BBBBBBBBBBBBBBBB◆◆
Buffer 0x4000b60
Keylen: 16, keybitstream: BBBBBBBBBBBBBBBB@◆Q◆dU , flags 50
Putting into uart_queue: XERROR

Message in queue: XERROR

State Machine UART_IN: X
State Machine UART_IN: D
DECRYPT
DATA
Data: 0xAA
Data: 0xE3
Data: 0x65
Data: 0x27
Data: 0x2C
Data: 0x81
Data: 0x07
Data: 0x8A
Data: 0xB6
Data: 0x11
Data: 0x6B
Data: 0x36
Data: 0x18
Data: 0x31
Data: 0xD0
Data: 0xF6
Data: 0xA5
Data: 0xD3
Data: 0xC8
Data: 0x58
Data: 0x7E
Data: 0x94
Data: 0x6B
Data: 0x53
Data: 0x0B
Data: 0x79
Data: 0x57
Data: 0x54
Data: 0x31
Data: 0x07
Data: 0xF1
Data: 0x5E
OP_DECRYPT
Process_thread: Decrypting
Ciphertext: 4000B80
CBC_Mode:
Key: BBBBBBBBBBBBBBBB@◆Q◆dU
IV: BBBBBBBBBBBBBBBB@◆Q◆dU
Buffer: BBBBBBBBBBBBBBBB◆◆e',◆◆◆□k6□1◆◆◆◆◆X~◆kS
                                                yWT1◆^
Buffer 0x4000b80
Keylen: 16, keybitstream: BBBBBBBBBBBBBBBB@◆Q◆dU , flags 50
Plaintext: Schoene Crypto Welt
Putting into uart_queue: D Schoene Crypto Welt
Message in queue: D Schoene Crypto Welt
State Machine UART_IN: X
Main-Thread is alive
State Machine UART_IN: K
DATA
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
```

```
ok
test_04 (__main__.MyTests) ... Data in Bytes: b'D' Data: 68
000000.000 TX   0000  44                                                D
Data in Bytes: b' ' Data: 32
000000.100 TX   0000  20
Data in Bytes: b'\xaa' Data: 170
000000.201 TX   0000  AA                                                .
Data in Bytes: b'\xe3' Data: 227
000000.301 TX   0000  E3                                                .
Data in Bytes: b'e' Data: 101
000000.401 TX   0000  65                                                e
Data in Bytes: b"'" Data: 39
000000.502 TX   0000  27                                                '
Data in Bytes: b',' Data: 44
000000.602 TX   0000  2C                                                ,
Data in Bytes: b'\x81' Data: 129
000000.702 TX   0000  81                                                .
Data in Bytes: b'\x07' Data: 7
000000.803 TX   0000  07                                                .
Data in Bytes: b'\x8a' Data: 138
000000.903 TX   0000  8A                                                .
Data in Bytes: b'\xb6' Data: 182
000001.003 TX   0000  B6                                                .
Data in Bytes: b'\x11' Data: 17
000001.104 TX   0000  11                                                .
Data in Bytes: b'k' Data: 107
000001.204 TX   0000  6B                                                k
Data in Bytes: b'6' Data: 54
000001.304 TX   0000  36                                                6
Data in Bytes: b'\x18' Data: 24
000001.404 TX   0000  18                                                .
Data in Bytes: b'1' Data: 49
000001.505 TX   0000  31                                                1
Data in Bytes: b'\xd0' Data: 208
000001.605 TX   0000  D0                                                .
Data in Bytes: b'\xf6' Data: 246
000001.705 TX   0000  F6                                                .
Data in Bytes: b'\xa5' Data: 165
000001.806 TX   0000  A5                                                .
Data in Bytes: b'\xd3' Data: 211
000001.906 TX   0000  D3                                                .
Data in Bytes: b'\xc8' Data: 200
000002.006 TX   0000  C8                                                .
Data in Bytes: b'X' Data: 88
000002.107 TX   0000  58                                                X
Data in Bytes: b'~' Data: 126
000002.207 TX   0000  7E                                                ~
Data in Bytes: b'\x94' Data: 148
000002.307 TX   0000  94                                                .
Data in Bytes: b'k' Data: 107
000002.407 TX   0000  6B                                                k
Data in Bytes: b'S' Data: 83
000002.508 TX   0000  53                                                S
Data in Bytes: b'\x0b' Data: 11
000002.608 TX   0000  0B                                                .
Data in Bytes: b'y' Data: 121
000002.708 TX   0000  79                                                y
Data in Bytes: b'W' Data: 87
000002.809 TX   0000  57                                                W
Data in Bytes: b'T' Data: 84
000002.909 TX   0000  54                                                T
Data in Bytes: b'1' Data: 49
000003.009 TX   0000  31                                                1
Data in Bytes: b'\x07' Data: 7
000003.110 TX   0000  07                                                .
Data in Bytes: b'\xf1' Data: 241
000003.210 TX   0000  F1                                                .
Data in Bytes: b'^' Data: 94
000003.310 TX   0000  5E                                                ^
Data in Bytes: b'X' Data: 88
000003.411 TX   0000  58                                                X
000004.512 RX   0000  44 20 53 63 68 6F 65 6E  65 20 43 72 79 70 74 6F  D Schoene Crypto
000004.512 RX   0010  20 57 65 6C 74 0D 0D 0D  0D 0D 0D 0D 0D 0D 0D 0D   Welt............
000004.512 RX   0020  0D 0D 00                                          ...
Replyb'D Schoene Crypto Welt\r\r\r\r\r\r\r\r\r\r\r\r\r\x00'
ok
```

Abbildung 8: Test04

```
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
OP_IV
State Machine UART_IN: X
State Machine UART_IN: I
DATA
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
Data: 0x41
OP_IV
State Machine UART_IN: X
State Machine UART_IN: D
DECRYPT
DATA
Data: 0x55
Main-Thread is alive
Data: 0x8F
Data: 0x85
Data: 0x68
Data: 0x96
Data: 0x87
Data: 0x31
Data: 0x42
Data: 0xB1
Data: 0x6D
Data: 0xC8
Data: 0xF2
Data: 0xEA
Data: 0x8F
Data: 0x33
Data: 0x4E
Data: 0xDA
Data: 0x7E
Data: 0x8F
Data: 0x71
Data: 0x37
Data: 0x87
Data: 0x7E
Data: 0xC2
Data: 0x50
Data: 0xAD
Data: 0x73
Data: 0x3A
Data: 0x74
Data: 0x03
Data: 0xCF
Data: 0xC0
OP_DECRYPT
Process_thread: Decrypting
Ciphertext: 4000C00
CBC_Mode:
Key: AAAAAAAAAAAAAAA@◆Q◆dU
IV: AAAAAAAAAAAAAAA@◆Q◆dU
Buffer: AAAAAAAAAAAAAAAU◆◆h◆◆1B◆m◆◆◆3N◆~◆q7◆~◆P◆s:t□◆◆
Buffer 0x4000c00
Keylen: 16, keybitstream: AAAAAAAAAAAAAAA@◆Q◆dU , flags 50
Plaintext: Schoene Crypto Welt
Putting into uart_queue: D Schoene Crypto Welt
Message in queue: D Schoene Crypto Welt
```

```
test_05 (__main__.MyTests) ... 000000.000 TX   0000  4B
000000.100 TX   0000  41                                              A
000000.201 TX   0000  41                                              A
000000.301 TX   0000  41                                              A
000000.401 TX   0000  41                                              A
000000.501 TX   0000  41                                              A
000000.602 TX   0000  41                                              A
000000.702 TX   0000  41                                              A
000000.802 TX   0000  41                                              A
000000.903 TX   0000  41                                              A
000001.003 TX   0000  41                                              A
000001.103 TX   0000  41                                              A
000001.203 TX   0000  41                                              A
000001.304 TX   0000  41                                              A
000001.404 TX   0000  41                                              A
000001.504 TX   0000  41                                              A
000001.605 TX   0000  41                                              A
000001.705 TX   0000  58                                              X
000001.905 TX   0000  49                                              I
000002.006 TX   0000  41                                              A
000002.106 TX   0000  41                                              A
000002.206 TX   0000  41                                              A
000002.306 TX   0000  41                                              A
000002.407 TX   0000  41                                              A
000002.507 TX   0000  41                                              A
000002.607 TX   0000  41                                              A
000002.707 TX   0000  41                                              A
000002.808 TX   0000  41                                              A
000002.908 TX   0000  41                                              A
000003.008 TX   0000  41                                              A
000003.109 TX   0000  41                                              A
000003.209 TX   0000  41                                              A
000003.309 TX   0000  41                                              A
000003.409 TX   0000  41                                              A
000003.510 TX   0000  41                                              A
000003.610 TX   0000  58                                              X
000003.710 TX   0000  44                                              D
000003.811 TX   0000  20                                               
000003.911 TX   0000  55                                              U
000004.011 TX   0000  8F                                              .
000004.111 TX   0000  85                                              .
000004.212 TX   0000  68                                              h
000004.312 TX   0000  96                                              .
000004.412 TX   0000  87                                              .
000004.513 TX   0000  31                                              1
000004.613 TX   0000  42                                              B
000004.713 TX   0000  B1                                              .
000004.813 TX   0000  6D                                              m
000004.914 TX   0000  C8                                              .
000005.014 TX   0000  F2                                              .
000005.114 TX   0000  EA                                              .
000005.215 TX   0000  8F                                              .
000005.315 TX   0000  33                                              3
000005.415 TX   0000  4E                                              N
000005.515 TX   0000  DA                                              .
000005.616 TX   0000  7E                                              ~
000005.716 TX   0000  8F                                              .
000005.816 TX   0000  71                                              q
000005.917 TX   0000  37                                              7
000006.017 TX   0000  87                                              .
000006.117 TX   0000  7E                                              ~
000006.218 TX   0000  C2                                              .
000006.318 TX   0000  50                                              P
000006.418 TX   0000  AD                                              .
000006.518 TX   0000  73                                              s
000006.619 TX   0000  3A                                              :
000006.719 TX   0000  74                                              t
000006.819 TX   0000  03                                              .
000006.920 TX   0000  CF                                              .
000007.020 TX   0000  C0                                              .
000007.120 TX   0000  58                                              X
000008.222 RX   0000  44 20 53 63 68 6F 65 6E  65 20 43 72 79 70 74 6F  D Schoene Crypto
000008.222 RX   0010  20 57 65 6C 74 0D 0D 0D  0D 0D 0D 0D 0D 0D 0D 0D   Welt............
000008.222 RX   0020  0D 0D 00                                          ...
Replyb'D Schoene Crypto Welt\r\r\r\r\r\r\r\r\r\r\r\r\r\x00'
ok
```

Abbildung 9: Test05