04_AES_ECB.md

AES-ECB

Anpassung der Makefile

In dem hier besprochenen Programm werden wir die Module `shell`, `shell_commands`, `crypto_aes` sowie `od` verwenden

Fertige Makefile

```
# Name der Anwendung
APPLICATION = aes_ecb_example

# Standardboard
BOARD ?= native

+  USEMODULE += shell_commands  # RIOT Shell Commands
+  USEMODULE += shell           # RIOT Shell Modul
+  USEMODULE += crypto_aes      # Verschlüsselung mithilfe von AES
+  USEMODULE += od              # Object Dump
+  USEMODULE += od_string       # Object Dump String representation

# Pfad zur RIOT installation
RIOTBASE ?= ${HOME}/RIOT

include $(RIOTBASE)/Makefile.include
```

Die Header-Dateien `crpyto/ciphers.h` und `crypto/aes.h`

In dem Programm werden wir die Header Dateien `crypto/ciphers.h` sowie `crypto/aes.h` verwenden.

Die `ciphers.h` Headerdatei enthält essentielle Strukturen und Funktionen um mit RIOT Daten zu verschlüsseln. Die wichtigste Struktur dabei ist die `cipher_t` Struktur:

```
/**
 * @brief basic struct for using block ciphers
 *      contains the cipher interface and the context
 */
typedef struct {
        const cipher_interface_t *interface;    /**< BlockCipher-Interface for the Cipher-Algorithms */
        cipher_context_t context;               /**< The encryption context (buffer) for the algorithm */
} cipher_t;
```

Wir werden die Member dieser Struktur nicht selber verwenden, aber `interface` ist ein Pointer auf eine `cipher_interface_t` Struktur, welche Informationen über die Blockgröße, Maximale Schlüsselgröße und Function Pointer zu den Init/Encrypt sowie Decrypt Funktionen des Algorithmus enthält. `cipher_context_t` ist ein Buffer, der von den Algorithmen intern verwendet wird.

Wir werden aus der `ciphers.h` Headerdatei außerdem noch die Funktionen `cipher_init`, `cipher_encrypt` sowie `cipher_decrypt` verwenden:

cipher_init

```
/**
 * @brief Initialize new cipher state
 *
 * @param cipher        cipher struct to init (already allocated memory)
 * @param cipher_id     cipher algorithm id
 * @param key           encryption key to use
 * @param key_size      length of the encryption key
 *
 * @return CIPHER_INIT_SUCCESS if the initialization was successful.
 * @return CIPHER_ERR_BAD_CONTEXT_SIZE if CIPHER_MAX_CONTEXT_SIZE has
 *          not been defined (which means that the cipher has not been
 *          included in the build)
 * @return  The command may return CIPHER_ERR_INVALID_KEY_SIZE if the
 *              key size is not valid.
 */
int cipher_init(cipher_t *cipher, cipher_id_t cipher_id,
                const uint8_t *key, uint8_t key_size);
```

Die cipher_init Funktion nimmt einen Pointer zu einer `cipher_t` Struktur, die Struktur darf uninitialisierter Speicher sein, die Funktion initialisiert diesen Speicher dann mit dem richtigen Kontext und Buffer. Das zweite Argument ist ein Pointer zu einer `cipher_interface_t` Struktur. Außerdem nimmt die Funktion den Schlüssel zur Verschlüsselung an, sowie dessen Größe.

Die Funktion gibt bei erfolgreicher Initialisiation `CIPHER_INIT_SUCCESS` zurück, ansonsten einer der Fehlercodes `CIPHER_ERR_BAD_CONTEXT_SIZE` oder `CIPHER_ERR_INVALID_KEY_SIZE`

Beispiel:

```
uint8_t key[AES_KEY_SIZE] = { /* ... */ };
cipher_t cipher;

int err = cipher_init(&cipher, CIPHER_AES_128, key, AES_KEY_SIZE);

if (err != CIPHER_INIT_SUCCESS)
{
    printf("Cipher Init failed: %d\n", err);
    exit(err);
}
```

Der Code definiert einen Schlüssel der Größe `AES_KEY_SIZE` (16). Erstellt danach eine uninitialisierte `cipher_t` Struktur auf dem Stack. Ruft die `cipher_init` Funktion auf mit dem Pointer zu der `cipher_t` Struktur, mit der cipher id `CIPHER_AES_128` , dem Pointer zum Schlüssel und der Größe des Schlüssels.

Der Code speichert dann das Ergebnis des `cipher_init` aufrufs in einer Variablen `err` ab und überprüft, ob es beim Initialisieren der `cipher_t` Struktur zu Fehlern kam.

cipher_encrypt

```
/**
 * @brief Encrypt data of BLOCK_SIZE length
 * *
 *
 * @param cipher    Already initialized cipher struct
 * @param input     pointer to input data to encrypt
 * @param output    pointer to allocated memory for encrypted data.
 *                  It has to be of size BLOCK_SIZE
 *
 * @return          The result of the encrypt operation of the underlying
 *                  cipher, which is always 1 in case of success
 * @return          A negative value for an error
 */
```

```
int cipher_encrypt(const cipher_t *cipher, const uint8_t *input,
                uint8_t *output);
```

Die cipher_encrypt Funktion verschlüsselt einen Datenblock der im `cipher_interface_t` gespeicherten Block size und nimmt als Argument einen Pointer zu einer initialisierten `cipher_t` Struktur, einen Pointer zu dem Klartextbuffer sowie einen Pointer zum Ciphertextbuffer (Also wohin die Ausgabe geschrieben werden soll). Die Klartext und Ciphertextbuffer sollten einen Block des verwendeten Algorithmus enthalten (Bei AES also mindestens 16 Bytes).

Die Funktion gibt bei erfolgreichem Verschlüsseln 1 zurück

cipher_decrypt

```
/**
 * @brief Decrypt data of BLOCK_SIZE length
 * *
 *
 * @param cipher    Already initialized cipher struct
 * @param input     pointer to input data (of size BLOCKS_SIZE) to decrypt
 * @param output    pointer to allocated memory for decrypted data.
 *                  It has to be of size BLOCK_SIZE
 *
 * @return          The result of the decrypt operation of the underlying
 *                  cipher, which is always 1 in case of success
 * @return          A negative value for an error
 */
int cipher_decrypt(const cipher_t *cipher, const uint8_t *input,
                uint8_t *output);
```

Die `cipher_decrypt` Funktion entschlüsselt einen Datenblock der für den Algorithmus geltenden Block size und nimmt als Argument einen Pointer zu einer initialisierten `cipher_t` Struktur, einen Pointer zu dem Ciphertext, der entschlüsselt werden soll sowie einen Pointer zu einem Buffer, in dem der entschlüsselte Klartext geschrieben werden soll.

Die Funktion gibt bei erfolgreichem Entschlüsseln 1 zurück

Programm zur Verschlüsselung einer Kurzen (bis zu 15 Zeichen) Nachricht

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#include "crypto/ciphers.h"
#include "crypto/aes.h"
#include "od.h"

int main(void)
{
        // Schlüssel für den AES Algorithmus
        uint8_t key[AES_KEY_SIZE] = {
        0x64, 0x52, 0x67, 0x55,
        0x6B, 0x58, 0x70, 0x32,
        0x73, 0x35, 0x75, 0x38,
        0x78, 0x2F, 0x41, 0x3F};

        cipher_t cipher;
        int err;

        // Initialisierung der cipher_t Struktur
        if ((err = cipher_init(&cipher, CIPHER_AES_128, key, AES_KEY_SIZE)) != CIPHER_INIT_SUCCESS)
        {
             printf("Failed to initialize cipher_t: %d\n", err);
```

```
            exit(err);
        }

        uint8_t input[AES_BLOCK_SIZE] = {0};  // Initialisiere den Eingabebuffer mit Nullen
        uint8_t output[AES_BLOCK_SIZE] = {0}; // Initialisiere den Ausgabebuffer mit Nullen

        sprintf((char *)input, "Testnachricht"); // Schreibe die Nachricht in den Eingabebuffer

        // Verschlüsseln der Eingabe
        if ((err = cipher_encrypt(&cipher, input, output)) != 1)
        {
            printf("Failed to encrypt data: %d\n", err);
            exit(err);
        }


       // Ausgabe des Buffers in Hexadecimal sowie der Druckbaren Zeichen in ASCII
        printf("Klartext: \t");
        od_hex_dump_ext(input, AES_BLOCK_SIZE, 0, 0);
        printf("Ciphertext: \t");
        od_hex_dump_ext(output, AES_BLOCK_SIZE, 0, 0);

        // Entschlüsseln der Verschlüsselten Eingabe,
       // durch das vertauschen von input und output,
       // wird der Verschlüsselte Text in den Inputbuffer wieder geschrieben,
       // welcher sich dadurch nicht ändern sollte.
        if ((err = cipher_decrypt(&cipher, output, input)) != 1)
        {
            printf("Failed to decrypt data: %d\n", err);
            exit(err);
        }

        printf("Entschlüsselt: \t");
        od_hex_dump_ext(input, AES_BLOCK_SIZE, 0, 0);

        exit(0);
}
```

Als Ausgabe erhalten wir:

```
Klartext:      54 65 73 74 6E 61 63 68 72 69 63 68 74 00 00 00  Testnachricht...
Ciphertext:    BC 4E DC 18 20 A9 EB 57 59 0F 76 C0 DC 9D 5A B9  .N.. ..WY.v...Z.
Entschlüsselt: 54 65 73 74 6E 61 63 68 72 69 63 68 74 00 00 00  Testnachricht...
```

Zurück zum Index

Zurück zu Kapitel 1: Grundlagen, Shell und Commands

Weiter zu Teil 5: AES im Cipher Block Chaining (CBC) Modus