ATS\_Crypto\_IotT\_Week1\_WS2023

**Gruppe: H**

**Teilnehmer:**

Mehmet Odabas

Baran Evsen

Omar Haj Abdulaziz

Bünyamin Berber

**Blatt 1**

**Aufgabe 1-4:**

Hardware/Arduino IDE und Bibliotheken installiert. Für NMCU V2 musste ein Treiber installiert werden.

**Aufgabe 5:** Egal welche Baudrate man benutzt hat, es gab immer wieder unvollständige Ausgaben, darum wird in unserem Code nach Serial.begin(<baudrate>), ein delay(time) gesetzt. Das Problem mit der Ausgabe gibt es dann nicht mehr.

**Aufgabe 6**

Es wurde derselbe Code von Test\_AES benutzt und die “testRandomTextCipher” Funktion hinzugefügt.

NMCU V2

|  |
| --- |
| void perfRandomTextCipher(BlockCipher \*cipher, const char \*name)  {      unsigned long totalEncryptTime = 0;      unsigned long totalDecryptTime = 0;      int numIterations = 100;      Serial.println(name);      for (int i = 0; i < numIterations; ++i)      {          for (int j = 0; j < 16; ++j)          {              buffer[j] = random(256);          }          unsigned long startEncrypt = micros();          cipher->encryptBlock(buffer, buffer);          unsigned long elapsedEncrypt = micros() - startEncrypt;          totalEncryptTime += elapsedEncrypt;            unsigned long startDecrypt = micros();          cipher->decryptBlock(buffer, buffer);          unsigned long elapsedDecrypt = micros() - startDecrypt;          totalDecryptTime += elapsedDecrypt;      }      Serial.print("Average Encryption Time: ");      Serial.print(totalEncryptTime / numIterations);      Serial.println("microseconds");      Serial.print("Average Decryption Time: ");      Serial.print(totalDecryptTime / numIterations);      Serial.println(" microseconds");      Serial.println();  }  void setup() {      Serial.begin(4800);      delay(6000);      Serial.println("Test Vectors:");      testCipher(&aes128, &testVectorAES128);      testCipher(&aes192, &testVectorAES192);      testCipher(&aes256, &testVectorAES256);      Serial.println(“Performance Test with random plaintext:”);      perfRandomTextCipher(&aes128, "AES-128-Random");      perfRandomTextCipher(&aes192, "AES-192-Random");      perfRandomTextCipher(&aes256, "AES-256-Random");  }  void loop()  {  } |

**Ausgabe NMCU V2:**

|  |
| --- |
| AES-128-Random  Average Encryption Time: 95 microseconds  Average Decryption Time: 142 microseconds  AES-192-Random  Average Encryption Time: 114 microseconds  Average Decryption Time: 171 microseconds  AES-256-Random    Average Encryption Time: 133 microseconds  Average Decryption Time: 200 microseconds |

Hinweis:  
5 Sekunden “delay” da, im Serial Monitor sonst die ersten Ausgaben nicht erscheinen.

**Aufgabe 7**

NMCU V2

|  |
| --- |
| **void testRandomTextAndKeyCipher(BlockCipher \*cipher, int keySize, const char \*name) {**  **unsigned long totalEncryptTime = 0;**  **unsigned long totalDecryptTime = 0;**  **int numIterations = 100;**  **Serial.println(name);**  **for (int i = 0; i < numIterations; ++i) {**  **for (int j = 0; j < keySize; ++j) {**  **key[j] = random(256);**  **}**  **for (int j = 0; j < 16; ++j) {**  **buffer[j] = random(256);**  **Serial.println(buffer[j]);**  **}**  **cipher->setKey(key, keySize);**  **unsigned long startEncrypt = micros();**  **cipher->encryptBlock(buffer, buffer);**  **unsigned long elapsedEncrypt = micros() - startEncrypt;**  **totalEncryptTime += elapsedEncrypt;**  **unsigned long startDecrypt = micros();**  **cipher->decryptBlock(buffer, buffer);**  **unsigned long elapsedDecrypt = micros() - startDecrypt;**  **totalDecryptTime += elapsedDecrypt;**  **}**  **Serial.print("Average Encryption Time: ");**  **Serial.print(totalEncryptTime / numIterations);**  **Serial.println(" microseconds");**  **Serial.print("Average Decryption Time: ");**  **Serial.print(totalDecryptTime / numIterations);**  **Serial.println(" microseconds");**  **Serial.println();**  **}**  void setup() {      Serial.begin(4800);      delay(6000);      Serial.println("Test Vectors:");      testCipher(&aes128, &testVectorAES128);      testCipher(&aes192, &testVectorAES192);      testCipher(&aes256, &testVectorAES256);      Serial.println(“Performance Test with random plaintext and key:”);      perfRandomTextandKeyCipher(&aes128, "AES-128-Random");      perfRandomTextandKeyCipher(&aes192, "AES-192-Random");      perfRandomTextandKeyCipher(&aes256, "AES-256-Random");  }  void loop()  {  } |

**Ausgabe NMCU V2:**

|  |
| --- |
| ES-128-Random  Average Encryption Time: 96 microseconds  Average Decryption Time: 143 microseconds  AES-192-Random  Average Encryption Time: 114 microseconds  Average Decryption Time: 171 microseconds  AES-256-Random  Average Encryption Time: 133 microseconds  Average Decryption Time: 200 microseconds |

**Aufgabe 8**

NMCU V2 CTR-Mode

|  |
| --- |
| **#include <Crypto.h>**  **#include <AES.h>**  **#include <CTR.h>**  **#include <string.h>**  **#define MAX\_PLAINTEXT\_SIZE  36**  **#define MAX\_CIPHERTEXT\_SIZE 36**  **struct TestVector**  **{**  **const char \*name;**  **byte key[32];**  **byte plaintext[MAX\_PLAINTEXT\_SIZE];**  **byte ciphertext[MAX\_CIPHERTEXT\_SIZE];**  **byte iv[16];**  **size\_t size;**  **};**  **// Test vectors for AES-128 in CTR mode from RFC 3686.**  **static TestVector const testVectorAES128CTR = {**  **.name        = "AES-128-CTR",**  **.key         = {0xAE, 0x68, 0x52, 0xF8, 0x12, 0x10, 0x67, 0xCC,**  **0x4B, 0xF7, 0xA5, 0x76, 0x55, 0x77, 0xF3, 0x9E},**  **.plaintext   = {0x53, 0x69, 0x6E, 0x67, 0x6C, 0x65, 0x20, 0x62,**  **0x6C, 0x6F, 0x63, 0x6B, 0x20, 0x6D, 0x73, 0x67},**  **.ciphertext  = {0xE4, 0x09, 0x5D, 0x4F, 0xB7, 0xA7, 0xB3, 0x79,**  **0x2D, 0x61, 0x75, 0xA3, 0x26, 0x13, 0x11, 0xB8},**  **.iv          = {0x00, 0x00, 0x00, 0x30, 0x00, 0x00, 0x00, 0x00,**  **0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01},**  **.size        = 16**  **};**  **static TestVector const testVectorAES192CTR = {**  **.name        = "AES-192-CTR",**  **.key         = {0x8E, 0x73, 0xB0, 0xF7, 0xDA, 0x0E, 0x64, 0x52,**  **0xC8, 0x10, 0xF3, 0x2B, 0x80, 0x90, 0x79, 0xE5,**  **0x62, 0xF8, 0xEA, 0xD2, 0x52, 0x2C, 0x6B, 0x7B},**  **.plaintext   = {0x6B, 0xC1, 0xBE, 0xE2, 0x2E, 0x40, 0x9F, 0x96,**  **0xE9, 0x3D, 0x7E, 0x11, 0x73, 0x93, 0x17, 0x2A},**  **.ciphertext  = {0x1A, 0xBC, 0x93, 0x24, 0x17, 0x52, 0x1C, 0xA2,**  **0x4F, 0x2B, 0x04, 0x59, 0xFE, 0x7E, 0x6E, 0x0B},**  **.iv          = {0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7,**  **0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, 0xFF},**  **.size        = 16**  **};**  **static TestVector const testVectorAES256CTR = {**  **.name        = "AES-256-CTR",**  **.key         = {0x60, 0x3D, 0xEB, 0x10, 0x15, 0xCA, 0x71, 0xBE,**  **0x2B, 0x73, 0xAE, 0xF0, 0x85, 0x7D, 0x77, 0x81,**  **0x1F, 0x35, 0x2C, 0x07, 0x3B, 0x61, 0x08, 0xD7,**  **0x2D, 0x98, 0x10, 0xA3, 0x09, 0x14, 0xDF, 0xF4},**  **.plaintext   = {0x6B, 0xC1, 0xBE, 0xE2, 0x2E, 0x40, 0x9F, 0x96,**  **0xE9, 0x3D, 0x7E, 0x11, 0x73, 0x93, 0x17, 0x2A},**  **.ciphertext  = {0x60, 0x1E, 0xC3, 0x13, 0x77, 0x57, 0x89, 0xA5,**  **0xB7, 0xA7, 0xF5, 0x04, 0xBB, 0xF3, 0xD2, 0x28},**  **.iv          = {0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7,**  **0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, 0xFF},**  **.size        = 16**  **};**  **CTR<AES128> ctraes128;**  **CTR<AES192> ctraes192;**  **CTR<AES256> ctraes256;**  **byte buffer[128];**  **bool testCipher\_N(Cipher \*cipher, const struct TestVector \*test, size\_t inc)**  **{**  **byte output[MAX\_CIPHERTEXT\_SIZE];**  **size\_t posn, len;**  **cipher->clear();**  **if (!cipher->setKey(test->key, cipher->keySize())) {**  **Serial.print("setKey ");**  **return false;**  **}**  **if (!cipher->setIV(test->iv, cipher->ivSize())) {**  **Serial.print("setIV ");**  **return false;**  **}**  **memset(output, 0xBA, sizeof(output));**  **for (posn = 0; posn < test->size; posn += inc) {**  **len = test->size - posn;**  **if (len > inc)**  **len = inc;**  **cipher->encrypt(output + posn, test->plaintext + posn, len);**  **}**  **if (memcmp(output, test->ciphertext, test->size) != 0) {**  **Serial.print(output[0], HEX);**  **Serial.print("->");**  **Serial.print(test->ciphertext[0], HEX);**  **return false;**  **}**  **cipher->setKey(test->key, cipher->keySize());**  **cipher->setIV(test->iv, cipher->ivSize());**  **for (posn = 0; posn < test->size; posn += inc) {**  **len = test->size - posn;**  **if (len > inc)**  **len = inc;**  **cipher->decrypt(output + posn, test->ciphertext + posn, len);**  **}**  **if (memcmp(output, test->plaintext, test->size) != 0)**  **return false;**  **return true;**  **}**  **void testCipher(Cipher \*cipher, const struct TestVector \*test)**  **{**  **bool ok;**  **Serial.print(test->name);**  **Serial.print(" ... ");**  **ok  = testCipher\_N(cipher, test, test->size);**  **ok &= testCipher\_N(cipher, test, 1);**  **ok &= testCipher\_N(cipher, test, 2);**  **ok &= testCipher\_N(cipher, test, 5);**  **ok &= testCipher\_N(cipher, test, 8);**  **ok &= testCipher\_N(cipher, test,  13);**  **ok &= testCipher\_N(cipher, test, 16);**  **if (ok)**  **Serial.println("Passed");**  **else**  **Serial.println("Failed");**  **}**  **void perfCipherEncrypt(const char \*name, Cipher \*cipher, const struct TestVector \*test)**  **{**  **unsigned long start;**  **unsigned long elapsed;**  **int count;**  **Serial.print(name);**  **Serial.print(" ... ");**  **cipher->setKey(test->key, cipher->keySize());**  **cipher->setIV(test->iv, cipher->ivSize());**  **start = micros();**  **for (count = 0; count < 500; ++count) {**  **cipher->encrypt(buffer, buffer, sizeof(buffer));**  **}**  **elapsed = micros() - start;**  **Serial.print("Average Encryption Time: ");**  **Serial.print(elapsed / 500);**  **Serial.println(" microseconds");**  **Serial.println();**  **}**  **void perfCipherDecrypt(const char \*name, Cipher \*cipher, const struct TestVector \*test)**  **{**  **unsigned long start;**  **unsigned long elapsed;**  **int count;**  **Serial.print(name);**  **Serial.print(" ... ");**  **cipher->setKey(test->key, cipher->keySize());**  **cipher->setIV(test->iv, cipher->ivSize());**  **start = micros();**  **for (count = 0; count < 500; ++count) {**  **cipher->decrypt(buffer, buffer, sizeof(buffer));**  **}**  **elapsed = micros() - start;**  **Serial.print("Average Decryption Time: ");**  **Serial.print(elapsed / 500);**  **Serial.println(" microseconds");**  **Serial.println();**  **}**  **void setup()**  **{**  **Serial.begin(9600);**  **delay(10000);**  **Serial.println();**  **Serial.println("Test Vectors:");**  **testCipher(&ctraes128, &testVectorAES128CTR);**  **testCipher(&ctraes192, &testVectorAES192CTR);**  **testCipher(&ctraes256, &testVectorAES256CTR);**  **Serial.println();**  **Serial.println("Performance Tests:");**  **perfCipherEncrypt("AES-128-CTR Encrypt", &ctraes128, &testVectorAES128CTR);**  **perfCipherDecrypt("AES-128-CTR Decrypt", &ctraes128, &testVectorAES128CTR);**  **perfCipherEncrypt("AES-192-CTR Encrypt", &ctraes128, &testVectorAES192CTR);**  **perfCipherDecrypt("AES-192-CTR Decrypt", &ctraes128, &testVectorAES192CTR);**  **perfCipherEncrypt("AES-256-CTR Encrypt", &ctraes128, &testVectorAES256CTR);**  **perfCipherDecrypt("AES-256-CTR Decrypt", &ctraes128, &testVectorAES256CTR);**  **}**  **void loop()**  **{**  **}** |

**Ausgabe NMCU V2:**

|  |
| --- |
| **Test Vectors:**  **AES-128-CTR ... Passed**  **AES-192-CTR ... Passed**  **AES-256-CTR ... Passed**  **Performance Tests:**  **AES-128-CTR Encrypt ... Average Encryption Time: 815 microseconds**  **AES-128-CTR Decrypt ... Average Decryption Time: 816 microseconds**  **AES-192-CTR Encrypt ... Average Encryption Time: 815 microseconds**  **AES-192-CTR Decrypt ... Average Decryption Time: 816 microseconds**  **AES-256-CTR Encrypt ... Average Encryption Time: 815 microseconds**  **AES-256-CTR Decrypt ... Average Decryption Time: 816 microseconds** |