h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Darmstadt University of Applied Sciences

– Faculty of Computer Science –

cryptography lab report 2

by
Lennart Eichhorn
Matriculation number: 759253

Part I

REPORT

1 INTRODUCTION

The supplied simple AES implementation shown Listing 2 was supplied as a reference. However it is missing the inverse S-Box required for decryption. Our first task is to implement the inverse S-Box and then use it to implement the decryption function.

We wanted to have a known working encryption and decryption implementation to compare our implementation against. For this we build simple implementation using the openssl library, as shown in Listing 3. We then verified that the openssl implementation produces the same output as the simple implementation.

Listing 1 shows our function for calculation the inverted SBOX table. It swaps the indices and values of the SBOX table, to create the inverted table.

Listing 1. `openssl` *based AES tool*

```cpp
int main() {
  unsigned char invertedSbox[256];
  for (unsigned int i = 0; i != 256; ++i) {
    invertedSbox[SBOX[i]] = i;
  }

  std::printf("static const unsigned char INVERTED_SBOX[256] = {");
  for (auto element : invertedSbox) {
    std::printf("0x%x,", element);
  }
  std::printf("};\n");
}
```

2 APPENDIX

Listing 2. Simple reference AES implementation

CPP

```cpp
#include <iostream>

const unsigned int NUM_ROUNDS = 4 + 6;

static const unsigned char SBOX[256] = {
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b,
    0xfe, 0xd7, 0xab, 0x76, 0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0,
    0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, 0xb7, 0xfd, 0x93, 0x26,
    0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2,
    0xeb, 0x27, 0xb2, 0x75, 0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0,
    0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84, 0x53, 0xd1, 0x00, 0xed,
    0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f,
    0x50, 0x3c, 0x9f, 0xa8, 0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5,
    0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2, 0xcd, 0x0c, 0x13, 0xec,
    0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14,
    0xde, 0x5e, 0x0b, 0xdb, 0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c,
    0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79, 0xe7, 0xc8, 0x37, 0x6d,
    0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f,
    0x4b, 0xbd, 0x8b, 0x8a, 0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e,
    0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e, 0xe1, 0xf8, 0x98, 0x11,
    0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f,
    0xb0, 0x54, 0xbb, 0x16};

static const unsigned char INV_SBOX[256] = {
    0x52, 0x9,  0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e,
    0x81, 0xf3, 0xd7, 0xfb, 0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87,
    0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb, 0x54, 0x7b, 0x94, 0x32,
    0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0xb,  0x42, 0xfa, 0xc3, 0x4e,
    0x8,  0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49,
    0x6d, 0x8b, 0xd1, 0x25, 0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16,
    0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92, 0x6c, 0x70, 0x48, 0x50,
    0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
    0x90, 0xd8, 0xab, 0x0,  0x8c, 0xbc, 0xd3, 0xa,  0xf7, 0xe4, 0x58, 0x5,
    0xb8, 0xb3, 0x45, 0x6,  0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0xf,  0x2,
    0xc1, 0xaf, 0xbd, 0x3,  0x1,  0x13, 0x8a, 0x6b, 0x3a, 0x91, 0x11, 0x41,
    0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
    0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8,
    0x1c, 0x75, 0xdf, 0x6e, 0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89,
    0x6f, 0xb7, 0x62, 0xe,  0xaa, 0x18, 0xbe, 0x1b, 0xfc, 0x56, 0x3e, 0x4b,
    0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
    0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x7,  0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59,
    0x27, 0x80, 0xec, 0x5f, 0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0xd,
```

```
    0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef, 0xa0, 0xe0, 0x3b, 0x4d,
    0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
    0x17, 0x2b, 0x4,  0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63,
    0x55, 0x21, 0xc,  0x7d};

static const unsigned char RCON[255] = {
    0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c,
    0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a,
    0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
    0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
    0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
    0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6,
    0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72,
    0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
    0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10,
    0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e,
    0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5,
    0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
    0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02,
    0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d,
    0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,
    0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
    0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb,
    0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c,
    0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a,
    0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
    0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
    0x74, 0xe8, 0xcb};

void keyExpansion(const unsigned char *key, unsigned char *roundKey) {
  unsigned char temp[4], k;

  for (unsigned int i = 0; i < 4; ++i) {
    roundKey[i * 4 + 0] = key[i * 4 + 0];
    roundKey[i * 4 + 1] = key[i * 4 + 1];
    roundKey[i * 4 + 2] = key[i * 4 + 2];
    roundKey[i * 4 + 3] = key[i * 4 + 3];
  }

  for (unsigned int i = 4; i < 4 * (NUM_ROUNDS + 1); ++i) {
    for (unsigned int j = 0; j != 4; ++j)
      temp[j] = roundKey[(i - 1) * 4 + j];
    if (i % 4 == 0) {
      k = SBOX[temp[0]];
      temp[0] = SBOX[temp[1]];
      temp[1] = SBOX[temp[2]];
      temp[2] = SBOX[temp[3]];
      temp[3] = k;
```

```
96
97        temp[0] = temp[0] ^ RCON[i / 4];
98      }
99      roundKey[i * 4 + 0] = roundKey[(i - 4) * 4 + 0] ^ temp[0];
100     roundKey[i * 4 + 1] = roundKey[(i - 4) * 4 + 1] ^ temp[1];
101     roundKey[i * 4 + 2] = roundKey[(i - 4) * 4 + 2] ^ temp[2];
102     roundKey[i * 4 + 3] = roundKey[(i - 4) * 4 + 3] ^ temp[3];
103   }
104 }
105
106 void addRoundKey(unsigned char *state, const unsigned char *roundKey,
107                  int round) {
108   for (unsigned int i = 0; i != 4; ++i) {
109     for (unsigned int j = 0; j != 4; ++j)
110       state[j * 4 + i] ^= roundKey[round * 4 * 4 + i * 4 + j];
111   }
112 }
113
114 void subBytes(unsigned char *state) {
115   for (unsigned int i = 0; i != 4; ++i) {
116     for (unsigned int j = 0; j != 4; ++j)
117       state[i * 4 + j] = SBOX[state[i * 4 + j]];
118   }
119 }
120
121 void invSubBytes(unsigned char *state) {
122   for (unsigned int i = 0; i != 4; ++i) {
123     for (unsigned int j = 0; j != 4; ++j)
124       state[i * 4 + j] = INV_SBOX[state[i * 4 + j]];
125   }
126 }
127
128 void shiftRows(unsigned char *state) {
129   unsigned char temp;
130
131   // Rotate first row 1 columns to left
132   temp = state[1 * 4 + 0];
133   state[1 * 4 + 0] = state[1 * 4 + 1];
134   state[1 * 4 + 1] = state[1 * 4 + 2];
135   state[1 * 4 + 2] = state[1 * 4 + 3];
136   state[1 * 4 + 3] = temp;
137
138   // Rotate second row 2 columns to left
139   temp = state[2 * 4 + 0];
140   state[2 * 4 + 0] = state[2 * 4 + 2];
141   state[2 * 4 + 2] = temp;
142
143   temp = state[2 * 4 + 1];
```

```
144   state[2 * 4 + 1] = state[2 * 4 + 3];
145   state[2 * 4 + 3] = temp;
146 
147   // Rotate third row 3 columns to left
148   temp = state[3 * 4 + 0];
149   state[3 * 4 + 0] = state[3 * 4 + 3];
150   state[3 * 4 + 3] = state[3 * 4 + 2];
151   state[3 * 4 + 2] = state[3 * 4 + 1];
152   state[3 * 4 + 1] = temp;
153 }
154 
155 void invShiftRows(unsigned char *state) {
156   unsigned char temp;
157 
158   // Rotate first row 1 columns to right
159   temp = state[1 * 4 + 3];
160   state[1 * 4 + 3] = state[1 * 4 + 2];
161   state[1 * 4 + 2] = state[1 * 4 + 1];
162   state[1 * 4 + 1] = state[1 * 4 + 0];
163   state[1 * 4 + 0] = temp;
164 
165   // Rotate second row 2 columns to right
166   temp = state[2 * 4 + 0];
167   state[2 * 4 + 0] = state[2 * 4 + 2];
168   state[2 * 4 + 2] = temp;
169 
170   temp = state[2 * 4 + 1];
171   state[2 * 4 + 1] = state[2 * 4 + 3];
172   state[2 * 4 + 3] = temp;
173 
174   // Rotate third row 3 columns to right
175   temp = state[3 * 4 + 0];
176   state[3 * 4 + 0] = state[3 * 4 + 1];
177   state[3 * 4 + 1] = state[3 * 4 + 2];
178   state[3 * 4 + 2] = state[3 * 4 + 3];
179   state[3 * 4 + 3] = temp;
180 }
181 
182 // XTIME is a macro that finds the product of {02} and the argument to XTIME
183 // modulo {1b}
184 #define XTIME(x) (((x) << 1) ^ ((((x) >> 7) & 1) * 0x1b))
185 
186 // Multiplty is a macro used to multiply numbers in the field GF(2^8)
187 #define MULTIPLY(x, y)                                                \
188   ((((y)&1) * (x)) ^ (((y) >> 1 & 1) * XTIME(x)) ^                    \
189    (((y) >> 2 & 1) * XTIME(XTIME(x))) ^                               \
190    (((y) >> 3 & 1) * XTIME(XTIME(XTIME(x)))) ^                        \
191    (((y) >> 4 & 1) * XTIME(XTIME(XTIME(XTIME(x))))))
```

```
192
193 void mixColumns(unsigned char *state) {
194   unsigned char Tmp, t;
195   for (unsigned int i = 0; i != 4; ++i) {
196     t = state[0 * 4 + i];
197     Tmp = state[0 * 4 + i] ^ state[1 * 4 + i] ^ state[2 * 4 + i] ^
198           state[3 * 4 + i];
199     state[0 * 4 + i] ^= XTIME(state[0 * 4 + i] ^ state[1 * 4 + i]) ^ Tmp;
200     state[1 * 4 + i] ^= XTIME(state[1 * 4 + i] ^ state[2 * 4 + i]) ^ Tmp;
201     state[2 * 4 + i] ^= XTIME(state[2 * 4 + i] ^ state[3 * 4 + i]) ^ Tmp;
202     state[3 * 4 + i] ^= XTIME(state[3 * 4 + i] ^ t) ^ Tmp;
203   }
204 }
205
206 void invMixColumns(unsigned char *state) {
207   unsigned char a, b, c, d;
208   for (unsigned int i = 0; i != 4; ++i) {
209     a = state[0 * 4 + i];
210     b = state[1 * 4 + i];
211     c = state[2 * 4 + i];
212     d = state[3 * 4 + i];
213
214     state[0 * 4 + i] = MULTIPLY(a, 0x0e) ^ MULTIPLY(b, 0x0b) ^
215                        MULTIPLY(c, 0x0d) ^ MULTIPLY(d, 0x09);
216     state[1 * 4 + i] = MULTIPLY(a, 0x09) ^ MULTIPLY(b, 0x0e) ^
217                        MULTIPLY(c, 0x0b) ^ MULTIPLY(d, 0x0d);
218     state[2 * 4 + i] = MULTIPLY(a, 0x0d) ^ MULTIPLY(b, 0x09) ^
219                        MULTIPLY(c, 0x0e) ^ MULTIPLY(d, 0x0b);
220     state[3 * 4 + i] = MULTIPLY(a, 0x0b) ^ MULTIPLY(b, 0x0d) ^
221                        MULTIPLY(c, 0x09) ^ MULTIPLY(d, 0x0e);
222   }
223 }
224
225 void cipher(const unsigned char *in, const unsigned char *roundKey,
226             unsigned char *out) {
227   unsigned char state[4 * 4];
228
229   for (unsigned int i = 0; i != 4; ++i) {
230     for (unsigned int j = 0; j != 4; ++j)
231       state[j * 4 + i] = in[i * 4 + j];
232   }
233
234   addRoundKey(state, roundKey, 0);
235   for (unsigned int round = 1; round < NUM_ROUNDS; ++round) {
236     subBytes(state);
237     shiftRows(state);
238     mixColumns(state);
239     addRoundKey(state, roundKey, round);
```

```
240   }
241   subBytes(state);
242   shiftRows(state);
243   addRoundKey(state, roundKey, NUM_ROUNDS);
244
245   for (unsigned int i = 0; i != 4; ++i) {
246     for (unsigned int j = 0; j != 4; ++j)
247       out[i * 4 + j] = state[j * 4 + i];
248   }
249 }
250
251 void decipher(const unsigned char *in, const unsigned char *roundKey,
252               unsigned char *out) {
253   unsigned char state[4 * 4];
254
255   for (unsigned int i = 0; i != 4; ++i) {
256     for (unsigned int j = 0; j != 4; ++j)
257       state[j * 4 + i] = in[i * 4 + j];
258   }
259
260   addRoundKey(state, roundKey, NUM_ROUNDS);
261   for (unsigned int round = NUM_ROUNDS - 1; round > 0; --round) {
262     invShiftRows(state);
263     invSubBytes(state);
264     addRoundKey(state, roundKey, round);
265     invMixColumns(state);
266   }
267   invShiftRows(state);
268   invSubBytes(state);
269   addRoundKey(state, roundKey, 0);
270
271   for (unsigned int i = 0; i != 4; ++i) {
272     for (unsigned int j = 0; j != 4; ++j)
273       out[i * 4 + j] = state[j * 4 + i];
274   }
275 }
276
277 int main(int argc, char *argv[]) {
278   unsigned char roundKey[240];
279   unsigned char out[16];
280
281   // Sample
282   {
283     const unsigned char in[16] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
284                                   'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p'};
285     const unsigned char key[16] = {0xa3, 0x28, 0x4e, 0x09, 0xc6, 0xfe,
286                                    0x53, 0x29, 0x97, 0xef, 0x6d, 0x10,
287                                    0x74, 0xc3, 0xde, 0xad};
```

```
288
289     std::cout << std::endl
290               << "Text before encryption:" << std::hex << std::endl;
291     for (unsigned int i = 0; i != 4 * 4; ++i)
292       std::cout << "0x" << (unsigned int)in[i] << ", ";
293     std::cout << std::endl;
294
295     keyExpansion(key, roundKey);
296     cipher(in, roundKey, out);
297
298     std::cout << std::endl << "Text after encryption:" << std::hex << std::endl;
299     for (unsigned int i = 0; i != 4 * 4; ++i)
300       std::cout << "0x" << (unsigned int)out[i] << ", ";
301     std::cout << std::endl;
302   }
303
304   return 0;
305 }
```

Listing 3. `openssl` *based AES tool*

```
CPP
1  #include <iostream>
2  #include <openssl/aes.h>
3  #include <openssl/evp.h>
4  #include <stdlib.h>
5
6  int main(int argc, char *argv[]) {
7
8    const unsigned char in[16] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
9                                  'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p'};
10   const unsigned char key[16] = {0xa3, 0x28, 0x4e, 0x09, 0xc6, 0xfe,
11                                  0x53, 0x29, 0x97, 0xef, 0x6d, 0x10,
12                                  0x74, 0xc3, 0xde, 0xad};
13
14   std::cout << std::endl << "Text before encryption:" << std::hex << std::endl;
15   for (unsigned int i = 0; i != 4 * 4; ++i)
16     std::cout << "0x" << (unsigned int)in[i] << ", ";
17   std::cout << std::endl;
18
19   unsigned char out[16] = {
20       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
21   };
22   AES_KEY aes_key;
23   AES_set_encrypt_key(key, 128, &aes_key);
24   AES_encrypt(in, out, &aes_key);
25
26   std::cout << std::endl << "Text after encryption:" << std::hex << std::endl;
```

```
27   for (unsigned int i = 0; i != 4 * 4; ++i)
28     std::cout << "0x" << (unsigned int)out[i] << ", ";
29   std::cout << std::endl;
30
31   return 0;
32 }
```