Implementation of the Advanced Encryption Standard (AES) on 8-bit AVR MCU

F. De Santis
desantis@tum.de

Sichere Implementierung kryptographischer Verfahren WS 2015-2016
Lehrstuhl für Sicherheit in der Informationstechnik
Technische Universität München.

21.10.2015

Lehrstuhl für Sicherheit
in der Informationstechnik

Technische Universität München

Outline

Section 1

Administrative Topics

People

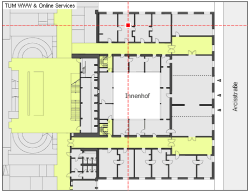
Teaching Assistant:

- www.sec.ei.tum.de/mitarbeiter/fabrizio-de-santis/
- Office hours by appointment only

Tutors:

- Silvan Streit silvan.streit@tum.de
- Andreas Schauer a.schauer@tum.de
- Tutor hours by **appointment only**
- Reserve your 30 minutes slot per E-mail

- Room N1005:
  - ➥ Wed. 12:30 – 16:30 (Andreas)
  - ➥ Thu. 13:00 – 17:00 (Silvan)



Overview on Exercises

1. 21.10.2015 (N2128, 16:45 - 18:15)
   ➥ AES Implementation on 8-bit AVR MCU (C/Asm)
2. 28.10.2015 (**N5325**, 16:45 - 18:15)
   ➥ RSA Implementation (Python)
3. 11.11.2015 (**N5325**, 16:45 - 18:15)
   ➥ Timing Attack on RSA Implementation (Python)
4. 02.12.2015 (N2128, **10:45 - 12:15**)
   ➥ DPA Attack on AES Implementation (Python)
5. 13.01.2016 (N2128, **10:45 - 12:15**)
   ➥ DFA Attack on AES Implementation (Python)

Section 2

Task Description

Task Description

Software Implementation of the AES on the ATMega644V MCU

- Programming language: C/Asm
- Programming environment: Atmel Studio 6.1
  http://www.atmel.com/tools/atmelstudio.aspx

Optimization goals:

1. Maximize throughput
2. Minimize RAM usage
3. Minimize code size

- Target only one optimization goal, then optimize the rest
- Your implementation will be used for DPA in the next assignment

Stairway to Heaven

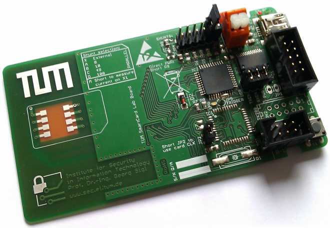
1. Register to the lecture
2. Download the framework
3. Implement the **encryption** algorithm of AES
4. Write a design report of your implementation (**important**!)
5. Submit your source code and report **before** 18.11.2015 23:59:59 CET

Section 3

The Advanced Encryption Standard (AES)

The Atmel ATMega644V Microcontroller

CPU	8-bit AVR
Op. Freq.	10 MHz
Flash	64 KBytes
SRAM	4 KBytes
EEPROM	2 KBytes
Registers	32
DRAM Memory	No
Crypto Engine	No
FPU / MPU / MMU	No / No / No



Datasheet: http://www.atmel.com/Images/2593s.pdf

Block Ciphers

- Block ciphers are the building blocks of **symmetric cryptography**
- A block cipher is a pair of **algorithms** $< \texttt{Enc}(k, m), \texttt{Dec}(k, c) >$ s.t. $\texttt{Dec}(k, \texttt{Enc}(k, m)) = m$ and k is "hard" to compute given $\{(m_i, c_i)\}$
- Modern block ciphers iterate a **round function** to update the current state:
  - ➥ Substitution layer (non-linear operation operating on small bundles)
  - ➥ Diffusion layer (distributes the input over different bundles)
  - ➥ Key addition (round keys are mixed to the state)
- Mainly two constructions: Feistel-networks and SP-networks
- A **key expansion** function is applied every round to generate so-called round keys

The Advanced Encryption Standard (AES)

- AES is the successor of DES determined by an international competition started on June 1998 and standardized by NIST on November 26, 2001 as FIPS number 197[1]

  $$\texttt{Enc/Dec} : \{0,1\}^{128} \times \{0,1\}^{n} \rightarrow \{0,1\}^{128}, \quad n \in \{128, 192, 256\},$$

  with 10/12/14 rounds for 128/192/256-bit key, respectively.

- AES is a SP-network derived from the Rijndael block cipher designed by J. Daemen and V. Rijmen

- Selection criteria were security but also implementation aspects *e.g.*, throughput, code size, area, ...

[1] http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Notation

- The plaintext, ciphertext, secret key and all intermediate states are 128-bit values arranged by columns 4×4-byte matrices
- Let $r^i = R_i(r^{i-1}, k^i)$ denote the state in the i^{th} round obtained by applying the round function $R_i(\cdot, \cdot)$

$$\begin{cases} r^{-1} & = & m \\ r^i & = & R_i(r^{i-1}, k^i) \\ c & = & r^{10} \end{cases} , \quad i = 0, \ldots, 10$$

and $k^i = E_i(k^{i-1})$ is the i^{th} round key obtained by applying the key expansion function $E_i(\cdot)$:

$$\begin{cases} k^0 & = & k \\ k^i & = & E_i(k^{i-1}) \end{cases} , \quad i = 1, \ldots, 10$$

The AES Operations

1. SubBytes (SB)
   - ➥ Non-linear mapping between input/output bits at **byte** level
2. ShiftRows (SR)
   - ➥ Intercolumn diffusion operating on the **rows** of the state
3. MixColumns (MC)
   - ➥ Interbyte diffusion operating on the **columns** of the state
4. AddRoundKey (ARK)
   - ➥ Key mixing at **bit** level

The AES Operations

1. SubBytes (SB)
   - ➥ $\mathtt{S}(x) = Ax^{-1} + b$ in $GF(2^8)$
   - ➥ Highly non-linear mapping between input-output **bytes** of the state

$$\begin{bmatrix} sb_0^i & sb_4^i & sb_8^i & sb_{12}^i \\ sb_1^i & sb_5^i & sb_9^i & sb_{13}^i \\ sb_2^i & sb_6^i & sb_{10}^i & sb_{14}^i \\ sb_3^i & sb_7^i & sb_{11}^i & sb_{15}^i \end{bmatrix} = \begin{bmatrix} \mathtt{S}(r_0^{i-1}) & \mathtt{S}(r_4^{i-1}) & \mathtt{S}(r_8^{i-1}) & \mathtt{S}(r_{12}^{i-1}) \\ \mathtt{S}(r_1^{i-1}) & \mathtt{S}(r_5^{i-1}) & \mathtt{S}(r_9^{i-1}) & \mathtt{S}(r_{13}^{i-1}) \\ \mathtt{S}(r_2^{i-1}) & \mathtt{S}(r_6^{i-1}) & \mathtt{S}(r_{10}^{i-1}) & \mathtt{S}(r_{14}^{i-1}) \\ \mathtt{S}(r_3^{i-1}) & \mathtt{S}(r_7^{i-1}) & \mathtt{S}(r_{11}^{i-1}) & \mathtt{S}(r_{15}^{i-1}) \end{bmatrix}$$

- Designed to be resistant against known cryptanalytic attacks
- No (opposite) fixed points ($\forall a \neq 0,\ \mathtt{S}(a) \neq a \wedge \mathtt{S}(a) \neq \bar{a}$)
- Many different implementations possible

The AES Operations

2. ShiftRows (SR)
   - ➥ Rotates the **rows** of the current state to the left according to $(i, j) \rightarrow (i, j - i \mod 4)$

$$\begin{bmatrix} sb^i_0 & sb^i_4 & sb^i_8 & sb^i_{12} \\ sb^i_5 & sb^i_9 & sb^i_{13} & sb^i_1 \\ sb^i_{10} & sb^i_{14} & sb^i_2 & sb^i_6 \\ sb^i_{15} & sb^i_3 & sb^i_7 & sb^i_{11} \end{bmatrix} = SR\left(\begin{bmatrix} sb^i_0 & sb^i_4 & sb^i_8 & sb^i_{12} \\ sb^i_1 & sb^i_5 & sb^i_9 & sb^i_{13} \\ sb^i_2 & sb^i_6 & sb^i_{10} & sb^i_{14} \\ sb^i_3 & sb^i_7 & sb^i_{11} & sb^i_{15} \end{bmatrix}\right)$$

- Spread bytes over columns: after the transformation every column has a byte from each other column

The AES Operations

3. MixColumns (MC)
   - ➥ For each column $c \in [0,3]$ the current state is a polynomial multiplication over GF(2^8) by a fixed polynomial modulo x^4+1:

$$\left[MC\left(\begin{bmatrix} sr^i_{4c+0} \\ sr^i_{4c+1} \\ sr^i_{4c+2} \\ sr^i_{4c+3} \end{bmatrix}\right), c \in [0,3] \right]$$

- Most complex operation in software implementations
- Coefficients accurately chosen to reduce implementation effort
- Produce diffusion through the column

The AES Operations

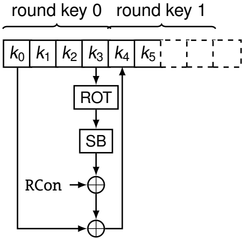
4. AddRoundKey (ARK)
   - ➥ **Bitwise** XOR between the current state and the corresponding round key

$$ARK\left(\begin{bmatrix} s_0^i & s_4^i & s_8^i & s_{12}^i \\ s_1^i & s_5^i & s_9^i & s_{13}^i \\ s_2^i & s_6^i & s_{10}^i & s_{14}^i \\ s_3^i & s_7^i & s_{11}^i & s_{15}^i \end{bmatrix}, \begin{bmatrix} k_0^i & k_4^i & k_8^i & k_{12}^i \\ k_1^i & k_5^i & k_9^i & k_{13}^i \\ k_2^i & k_6^i & k_{10}^i & k_{14}^i \\ k_3^i & k_7^i & k_{11}^i & k_{15}^i \end{bmatrix}\right)$$

$$= \begin{bmatrix} s_0^i \oplus k_0^i & s_4^i \oplus k_4^i & s_8^i \oplus k_8^i & s_{12}^i \oplus k_{12}^i \\ s_1^i \oplus k_1^i & s_5^i \oplus k_5^i & s_9^i \oplus k_9^i & s_{13}^i \oplus k_{13}^i \\ s_2^i \oplus k_2^i & s_6^i \oplus k_6^i & s_{10}^i \oplus k_{10}^i & s_{14}^i \oplus k_{14}^i \\ s_3^i \oplus k_3^i & s_7^i \oplus k_7^i & s_{11}^i \oplus k_{11}^i & s_{15}^i \oplus k_{15}^i \end{bmatrix}$$

The AES Key Schedule

- The secret key is expanded to $N_r + 1$ round keys
- Each **column** of the key state is viewed as a 32-bit word k_i
- If $i \mod 4 = 0$:
  1. Rotation of k_i by one byte to the left
  2. S-box lookups of each byte
  3. Bitwise addition with a constant `RCon`
- else $k_i = k_{i-1} \oplus k_{i-4}$



- If a round key and the corresponding round number are known, then the secret key can be computed back

The AES Encryption Algorithm

Algorithm 1 Pseudocode for the AES $\texttt{Enc}$ Algorithm

Input: m, k
Output: $c = \texttt{Enc}(k, m)$

$rk[0 \dots 10] \leftarrow \mathit{KeySched}(k)$ # generate round keys
$s \leftarrow \text{ARK}(p, rk[0])$ # key whitening
for $i = 1$ to 9 **do**
    $s \leftarrow \text{ARK}(\text{MC}(\text{SR}(\text{SB}(s))), rk[i])$ # iterate the round function
end for
$c \leftarrow \text{ARK}(\text{SR}(\text{SB}(s)), rk[10])$ # last round without MC
return c

The $\texttt{Enc}$ algorithm is typically enough for confidentiality (*e.g.* CFB and OFB modes), authentication (*e.g.* CBC-MAC) and integrity (*e.g.* Davies-Meyer)

The AES Decryption Algorithm

Algorithm 2 Pseudocode for the AES `Dec` Algorithm

Input: c, k
Output: $m = \texttt{Dec}(k, c)$

$rk[0 \dots 10] \leftarrow \mathit{KeySched}(k)$
$s \leftarrow \text{InvSR}(\text{InvSB}(\text{ARK}(c)), rk[10])$
for $i = 9$ downto 1 **do**
    $s \leftarrow \text{ARK}(\text{InvMC}(\text{InvSR}(\text{InvSB}(s))), rk[i])$
end for
$m \leftarrow \text{ARK}(s, rk[0])$
return m

The inverse operations are executed in reverse order using the round keys in reverse order.

8-bit Software Implementation

1. SubBytes (SB)
   - ➥ Typically implemented as Look-Up Table (LUT) in software, but many implementations / trade-offs are possible
2. ShiftRows (SR)
   - ➥ Just Re-Indexing
3. MixColumns (MC)
   - ➥ Bit shifts and conditional bitwise XORs (XTIME operation)
4. AddRoundKey (ARK)
   - ➥ Bitwise XORs

The SubBytes Operation

- The SubBytes is typically implemented as LUT in software
- It can also be computed by composing two transformations:
  1. Inversion: the multiplicative inverse is computed over the field GF(2^8) modulo $x^8 + x^4 + x^3 + x + 1$, where 0 is mapped to itself.
  2. Affine transformation: every element is viewed as a bit vector and the following matrix transformation is applied

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

The MixColumns Operation

- Each column is viewed as a polynomial $a(x) = \sum_{i=0}^{3} a_i x^i$ with $a_i \in \mathrm{GF}(2^8)$ and multiplied by the fixed polynomial $b(x) = 3x^3 + x^2 + x + 2$ modulo $x^4 + 1$. In matrix form the modular multiplication can be represented as:

$$\begin{bmatrix} a'_0 \\ a'_1 \\ a'_2 \\ a'_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

- The fixed polynomial was chosen s.t. only the multiplication by 02 must be implemented.
- The multiplication by 03 is computed by a multiplication by 02 followed by an addition.

The MixColumns Operation

- In GF(2^8), the addition corresponds to a bitwise XOR
- In GF(2^8), the multiplication by 02 corresponds to a left shift and a conditional addition
- The multiplication by 02 is implemented by the so-called XTIME operation

Algorithm 3 Pseudocode for the XTIME Operation

Input: $a(x)$
Output: $b = a(x) \cdot x$

1: $b \leftarrow a_7 \| a_6 \| a_5 \| a_4 \| a_3 \| a_2 \| a_1 \| a_0 \ll 1$ # left shift
2: **if** $a_7 == 1$ **then**
3: $\quad b \leftarrow b \oplus 0x1B$ # reduction
4: **end if**
5: **return** b

The XTIME Operation: Example

Reduction polynomial:
$r(x) = x^8 + x^4 + x^3 + x + 1 = (0001\ 0001\ 1011)_2 = 0x11B$

- Multiplication by x without reduction:

$$
\begin{array}{lcll}
a(x) & = & x^6 + x + 1 & (0100\ 0011)_2 \\
a(x) \cdot x & = & x^7 + x^2 + x & (1000\ 0110)_2 \\
a(x) \cdot x \mod r(x) & = & x^7 + x^2 + x & (1000\ 0110)_2
\end{array}
$$

- Multiplication by x with reduction:

$$
\begin{array}{lcll}
a(x) & = & x^7 + x + 1 & (1000\ 0011)_2 \\
a(x) \cdot x & = & x^8 + x^2 + x & (0001\ 0000\ 0110)_2 \\
a(x) \cdot x \mod r(x) & = & x^4 + x^3 + x^2 + 1 & (0001\ 1101)_2
\end{array}
$$

Section 4

Framework

Framework

- Online resources available @ https://tueisec-sica.sec.ei.tum.de/
  - ➥ Framework
    - ✯ Installation scripts for Ubuntu Linux 10.04, 13.04 and 14.04
    - ✯ Toolchain
    - ✯ Project file for Atmel Studio 6.1 (Windows)
    - ✯ Skeleton files
    - ✯ Report template
  - ➥ VM with pre-installed Framework (VirtualBox)
- Website access only from the TUM Intranet and over VPN
  - ➥ https://www.lrz.de/services/netz/mobil/vpn/

Skeleton Files and Template

Files:

- `student.c`
  - ➥ Put the implementation here and adjust the compilation options
- `SIKA_AES_Report.tex`
  - ➥ Write a report of the implementation justifying your design choices
- `main.c` has **NOT** to be modified
- `student.h` has **NOT** to be modified

Two functions available:

- `aes128_init()` for pre-computations, possibly
- `aes128_encrypt()` for the AES encryption

Lesson Learned

- Please mind the ordering of plaintext/ciphertext bytes in the state matrix during loadings/unloadings
- The `buf` variable must contain the plaintext before the AES encryption and the ciphertext after the AES encryption
- The implementation must be able to perform multiple encryptions using the same key

- Do **NOT** change the prototype of provided functions
- Do **NOT** add header files

- The `aes128_encrypt()` function must compute the full AES encryption, not just “some” rounds ...

Section 5

Submitting Results

Handing in Results

- Hand in `student.c` and the **PDF** report @ `https://tueisec-sica.sec.ei.tum.de/handin/`
  - ➥ Authentication using the LRZ Kennung required

- Multiple submissions are possible
  - ➥ Only the last submission is considered (files are overwritten)

- Deadline for submission fixed in 4 weeks
  - ➥ **18.11.2015 23:59:59 CET**

- Only one optimization goal should be pursued
  - ➥ Physical security has **NOT** to be considered for this assignment

Evaluation

- The implementation will be **automatically** compiled and tested against some freshly generated pairs of (plaintexts, keys)
- The following parameters will be measured:
  - ➥ Average time to bulk encryptions
  - ➥ Average RAM usage
  - ➥ Code size
- You pass the assignment if the code works correctly for multiple encryptions under the same key

Rules

- Comment your code carefully (i.e. optimizations)
- The code will be tested on Ubuntu Linux 14.04 using avr-gcc 4.8.2. Make sure that your compiler options are supported by this version.

- Do **NOT** submit code which does not compile
- Do **NOT** submit code that runs for more than 1M cycles
- Do **NOT** copy and paste code from the Internet
- Team work is allowed, **but** every student has to submit his/her own implementation