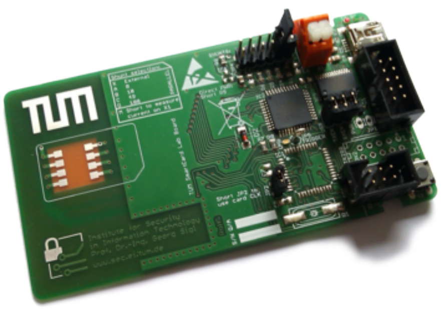
Smart Card Laboratory

Prof. Dr.-Ing. Georg Sigl



Institute for Security in Information Technology
Technische Universität München

Note: This is a draft version of the script. If you find any errors or would like to provide feedback and suggestions for improvement please send your comments to: *oscar.guillen@tum.de*

Contents

1 Lab overview

1.1 Task description

The goal of the lab is to introduce the concepts of Side-channel Analysis (SCA) and to help understand the implementation of SCA attacks and countermeasures in a practical manner. The target of evaluation in the lab is a Smart Card for a Pay-TV system. This Smart Card is used to decrypt a video stream transmitted over the network.
During the first part of the lab, you put on the "black hat" and become the attacker. Your goal is to extract the secret key stored within the device by making use of Differential Power Analysis (DPA). And later to use the extracted key to clone the original card by programming your own Smart Card emulator. In the second part of the lab you put on your "white hat" to implement and test different countermeasures to protect the Smart Card that you programmed. You test the resistance of the countermeasures by improving the attack scripts to try and break them. During this part, the focus is placed on comparing the increase in security with the trade-offs in terms of program size and execution time.

1.2 Organizational matters

The laboratory is performed in groups of four people. It is recommended that the group be divided into two teams, we will call them Team A and Team B. During the first part, Team A is responsible for developing the scripts for the DPA attacks, while Team B works on programming the Smart Card emulator which is used to clone the original card. For the second part, Team A is responsible for implementing countermeasures against DPA in the Smart Card emulator, while Team B improves the attack scripts to bypass these countermeasures.

Tip: Pay attention to good communication within your group and help the other team with their tasks if needed.

There are three major milestones for the laboratory. The first one is the *intermediate presentation*, the second one is the *final presentation* and lastly an *oral exam* at the end of the course. During the intermediate presentation you present the results of the first part of the lab. These include your attack strategy to perform DPA, the design and implementation details of your clone card, and an outlook of the next steps. During the final presentation you show a detailed description of the DPA countermeasures that were implemented and the improvements made to the attack scripts in order to break them. Here you also show the comparison between the different countermeasures, their resistance, and the impact of code size and speed in your Smart Card emulator. Lastly, in the oral exam you are going to be asked questions based on the work you described in your lab protocol.
You are free to work on the lab tasks where and when you want. The lab room, N1003, is open from Monday to Friday from 7:00 am to 9:00 pm. You are free to use any of the computers in the room. Please do note that there are other persons using the lab as well and remember to log out when you are not using them.

1.3 Environment

For the completion of the assignments in the lab you are given several hardware tools. These include: two Smart Cards emulators, an AVR programmer (AVRISP MkII), a programmer helper (which is used to supply power and clock pulses to the card while programming) and a logic analyzer. Apart from this, you are given a set of Python and Matlab scripts and you will have access to different tools which are installed in the computers found in the lab room. This section aims to provide a reference on how to make the most out of this.

1.3.1 Smart Card Emulator Hardware

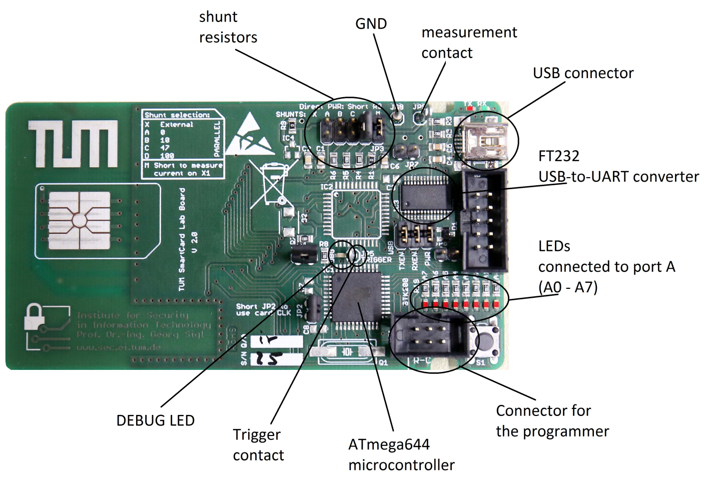

shunt resistors
GND
measurement contact
USB connector
FT232 USB-to-UART converter
LEDs connected to port A (A0 - A7)
Connector for the programmer
ATmega644 microcontroller
Trigger contact
DEBUG LED

Figure 1: SmartCard

Each group receives two SmartCards, the first one is a reference card with a working OS and a pre-programmed key, this will be your target of evaluation for the DPA attacks during the first part of the lab. The second one is a blank card, which is used to implement your clone card. Figure 1 shows the top side of the provided hardware. It contains an ATmega644 8-bit microcontroller, an USB-to-UART converter, which can be used to debug your program, different shunt resistors for the power measurements, connectors and debug LEDs. All the signals needed for the SmartCard (e.g. power supply, clock, I/O, reset) are transmitted over the ISO 7816 interface. The pin assignment is shown in Figure 2. You can find the schematic of the board as well as the data sheet of the microcontroller attached to the lab materials. Before you start the lab take some time to get familiar with the hardware.

1.3.2 Matlab

Matlab may be used to program your DPA scripts and for the measurement of power traces. To ensure to start the right matlab version, load the module with the following command (linux console):

```
$ module load matlab/R2014b
```

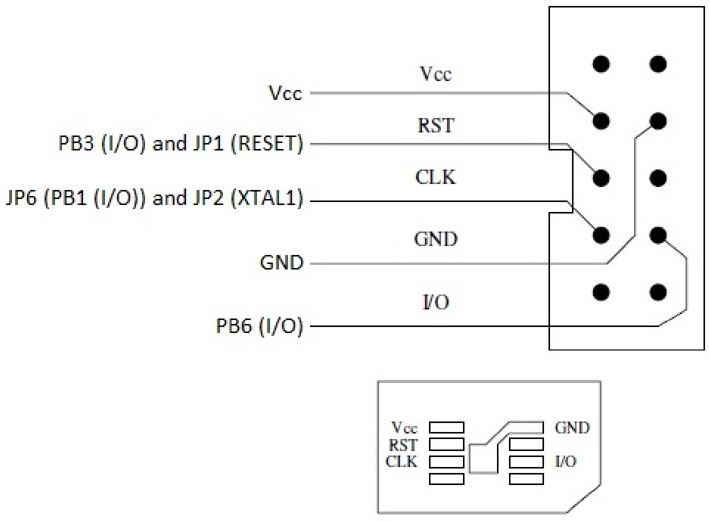


Figure 2: Pin assignment of the SmartCard

Afterwards type "matlab" in the console and press enter to start the loaded module.

1.3.3 Logic analyzer

You are provided a logic analyzer as part of the material given for the lab. This tool will help you understand the communication protocol between the Smart Card and the terminal. To use it, you need to first download the software which you will run from your computer.
Software: logic 1.1.15 (64-bit)
`http://downloads.saleae.com/Logic%20Setup%201.1.15%20(64-bit).exe`

To display the data values at the I/O line, use the following configuration of the logic analyzer:

- Async Serial Analyzer
- Baudrate = 4.8Mhz / 372 clocks = 12900 (bit/sec)
- 8 bits per transfer
- 1 bit stop
- Even parity
- LSB first
- Non inverted

1.3.4 Pay-TV scripts

The lab materials found in the compressed file which you received at the beginning of the course include several scripts. The following is a description of their use.

- client
  With this script, the video stream can be started. The correct SmartCard

has to be inserted to decrypt the stream. You can execute the script, using the following parameters, where x is your group number:

```
$ ./client tueisec-sigltv 2000x
```

- test_system
  You can use this script to test the clone card before the DPA is done and to test the if the key is right before the clone card is done

1.3.5 Setting up Eclipse for AVR

Eclipse and the AVR tools are already installed in the lab computers. In order to configure Eclipse for programming the AVR microcontroller the following steps are required:

1. Install the C support for Eclipse CDT:
   - Start eclipse (version 3.5.2, Galileo)
   - Go to "Help>Install New Software".
   - In the "work with:" text box write:
     `http://download.eclipse.org/tools/cdt/releases/galileo`
   - Press enter
   - Check the boxes for CDT Main Features and CDT Optional Features
   - Hit "Next>" and follow the instructions on the next pages.

2. Install the AVR Plugin (manual installation is required):
   - Download the AVR Eclipse Plugin stable release version 2.3.4:
     `http://sourceforge.net/projects/avr-eclipse/files/avr-eclipse%20stable%20release/2.3.4/avreclipse-p2-repository-2.3.4.20100807PRD.zip/download`
   - Open Eclipse
   - Go to "Help > Install New Software"
   - Click on the "Add..." Button.
   - In the new dialog hit "Archive...", select the downloaded file and click on "OK".
   - Select the AVR Eclipse Plugin from the list, hit "Next>" and follow the instructions on the next pages.

3. Create a project:
   - Open Eclipse
   - Select File > New > Project
   - In the new Project Wizard select C Project.
   - Select AVR Cross Target Application
   - Enter a project name (e.g. "hello_world")
   - Click "Next >"
   - Automatic build configurations can be selected
   - Click "Next >"
   - Select the target processor as ATMEGA644 and clock frequency to 3276800 (the values for MCU type and MCU frequency can later be changed via the project properties)
   - Click Finish

4. Create a source file:
    - Select the project you just created (e.g. "hello_world")
    - Click on "File > New > Source File"
    - Type main.c for the name
    - Click Finish.
    - Type the code from Listing 1 in the editor
    - Click "File > Save"
    - Click "Project > Build Project"

Listing 1: Blinking LED sample program

```
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
    DDRA| = (1<<PINB2); // Debug-LED
    while(1)
    {
        PORTB^=(1<<PINB2);
        _delay_ms(250);
    }
}
```

5. Configure the AVRISP MKII
    - Click "Project > Properties"
    - Expand the AVR menu
    - Select AVRDude
    - Create a new configuration
    - Select Atmel AVR ISP mkII
    - On "Override default port -P" write "usb"
    - The command line preview should look like this:
      avrdude -cavrisp2 -Pusb
    - Click "OK" and then click "Apply"

6. Program the device
    - Click on "AVR > Upload Project to Target Device"
    - Wait till the program is loaded
    - You might need to disconnect the programmer from the card in order to correctly see the LED blinking.

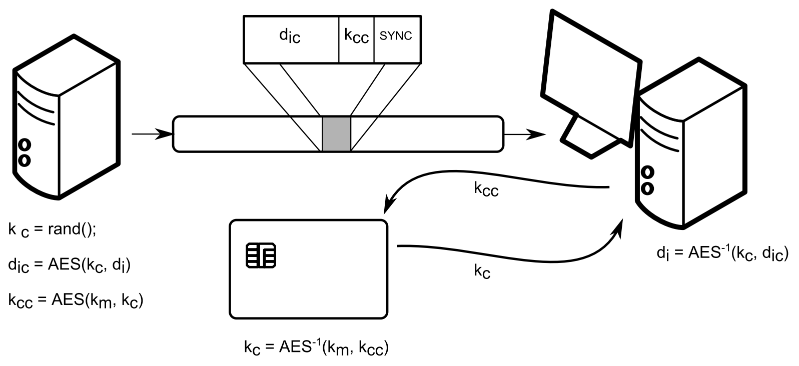
1.3.6 Debugging with the serial port

The AVRISP MkII programmer which is included in the lab material does not provide debugging capabilities. Because of this, the hardware of the Smart Card emulator includes a USB-to-UART converter which may be used to connect the serial port from the microcontroller to a USB port in your computer. You may use this interface to send debug information to a console. To setup a serial console in Linux, please take a look at the following tutorials:
`https://help.ubuntu.com/community/SerialConsoleHowto`
`http://goo.gl/KswwA`

2 The PayTV-system

Hardware attacks such as Side-channel Analysis are threat to the security of embedded devices. This is especially when the hardware is handed out to consumer, as each one of them could be a potential attacker. This is the scenario that is depicted our laboratory. Each group receives a Smart Card, on which a cryptographic algorithm is executed. The context is that of a PayTV system. You can only view the encrypted video stream, if the card with the correct key is inserted into the card reader. To understand this scenario, we will take a look into how the Pay-TV system developed for the lab works.

A server is used to transmit a video stream which should only be seen by authorized customers. First, the video stream is divided into chunks d_i with a size of 16kB. These chunks are encrypted using AES. For each chunk, a random chunk key k_c is generated and used to encrypt d_i. The random chunk key k_c is then encrypted with a master key k_m using AES-128 as well. The encrypted data chunk d_{ic} and the encrypted chunk key k_{cc} are transmitted to the client. To obtain the chunk key, the client has to decrypt k_{cc} with the correct master key k_m. This operation is done by the SmartCard, which contains a secure copy of the master key k_m. Afterwards the client decrypts the video chunk. In the lab the client is a PC with a card reader.



d_{ic}
k_{cc}
SYNC
k_{cc}
k_c
k_c = rand();
d_{ic} = AES(k_c, d_i)
k_{cc} = AES(k_m, k_c)
$k_c = AES^{-1}(k_m, k_{cc})$
$d_i = AES^{-1}(k_c, d_{ic})$

Figure 3: Structure of the PayTV-System

Figure 3 shows the general structure of a PayTV-system. The important parts can be summarized as follow:

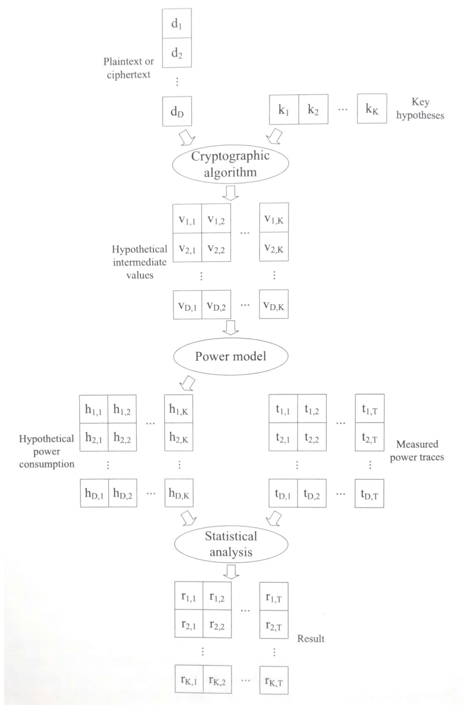
- Server:
  - Video data stream is divided into chunks d_i
  - A random key k_c encrypts each data chunk
  - The random key k_c is then encrypted with a master key k_m to generate k_{cc}
  - k_m is known only by the server and the card
- Server sends packet that include:
  - An encrypted data chunk, d_{ic}
  - The encrypted random key, k_{cc}
  - Synchronization data

- PC:
  - Sends the encrypted random key, k_{cc} , to the SmartCard
  - Uses k_c to decrypt d_{ic}
  - Displays the plaintext data chunk d_i
- SmartCard
  - Decrypts k_{cc} with k_m to obtain k_c

3 Differential Power Analysis (DPA)

3.1 Theory of DPA

To extract the secret key k_m from within the microcontroller, Differential Power Analysis is used in this lab. DPA attacks exploit the relation between the power consumption of a cryptographic device and the data and operations that it performs.[2, p. 119]



Plaintext or ciphertext
d_1 d_2 d_D
k_1 k_2 k_K
Key hypotheses
Cryptographic algorithm
Hypothetical intermediate values
$v_{1,1}$ $v_{1,2}$ $v_{1,K}$ $v_{2,1}$ $v_{2,2}$ $v_{2,K}$ $v_{D,1}$ $v_{D,2}$ $v_{D,K}$
Power model
Hypothetical power consumption
$h_{1,1}$ $h_{1,2}$ $h_{1,K}$ $h_{2,1}$ $h_{2,2}$ $h_{2,K}$ $h_{D,1}$ $h_{D,2}$ $h_{D,K}$
$t_{1,1}$ $t_{1,2}$ $t_{1,T}$ $t_{2,1}$ $t_{2,2}$ $t_{2,T}$ $t_{D,1}$ $t_{D,2}$ $t_{D,T}$
Measured power traces
Statistical analysis
$r_{1,1}$ $r_{1,2}$ $r_{1,T}$ $r_{2,1}$ $r_{2,2}$ $r_{2,T}$ $r_{K,1}$ $r_{K,2}$ $r_{K,T}$
Result

Figure 4: Block diagram of a DPA attack[2, p. 122]

The first step in a DPA attack is to choose an intermediate result to attack. We are looking for a function that makes use of the secret we want to extract and values which we know. This function can be expressed as $f(d, k)$, where d is a known, non-constant, data value and k is a part of the key. The second step is to measure the power consumption while the device encrypts or decrypts a data block D. For each of these runs, you as an attacker need to know the corresponding data values d. To perform the attack, you have to calculate a hypothetical intermediate value using the known function $f(d, k)$ and known data values d for every possible choice of k. This is why k has to be a part of the key and not the complete key itself. Think about it as a Divide-and-Conquer approach. The next step is to map these values to a matrix of hypothetical power consumption values making use of a power model. The most commonly used power models are the Hamming-distance and the Hamming-weight model. For this lab, you will make use of the latter. The last step of the DPA attack is the comparison of each column of the hypothetical power consumption matrix with each column of the measured power traces matrix. You can use the following correlation coefficient to compare the two matrices [2, p. 120ff]. Figure 4 shows a block diagram of the DPA attack.

$$r_{i,j} = \frac{\sum_{d=1}^{D}(h_{d,i} - \overline{h}_i) \cdot (t_{d,j} - \overline{t}_j)}{\sqrt{\sum_{d=1}^{D}(h_{d,i} - \overline{h}_i)^2 \cdot (t_{d,j} - \overline{t}_j)^2}}$$

Tip: For detailed information about the DPA attack use the book "Power Analysis Attacks" [2] from Stefan Mangard, Elisabeth Oswald and Thomas Popp.

3.2 Measurements with the Picoscope

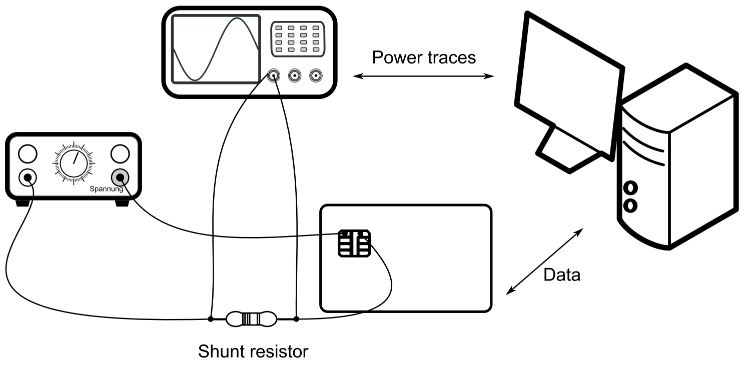


Figure 5: DPA Setup

As described in Section 3.1, you have to measure the power consumption of the cryptographic device to reveal the key k_m. To obtain the power traces you measure the voltage drop over a shunt resistor. This gives you the information about the current consumption of the device. The hardware for the Smart Card emulator includes different shunt resistors, which may be selected by configuring the jumpers. The setup of the DPA is shown in Figure 5. The measuring instrument is the USB oscilloscope "Picoscope 5204". The trigger input of the oscilloscope (AUXIO) is connected to the trigger pin of the SmartCard (JP5). Channel A of the oscilloscope should be connected to the pin of the card JP9 (shunt resistors). JP8 is the ground connection. You can use the provided measuring script "take_measurements.m" to get your power traces. This script communicates with the terminal and sends random data for decryption to the card. The power traces and the decrypted values are saved to the output directory. You can modify the amount of traces, the sample rate, the sample window and the output directory. Please note that the sample window is taken from the falling edge of the trigger signal and extends backward in time. Figure 6 shows the duration of the sample window in comparison to the trigger signal.

3.3 Improvements to the DPA scripts

To perform a DPA attack, you often have to handle a lot of data. Therefore you should take care about the following points: trace compression and memory

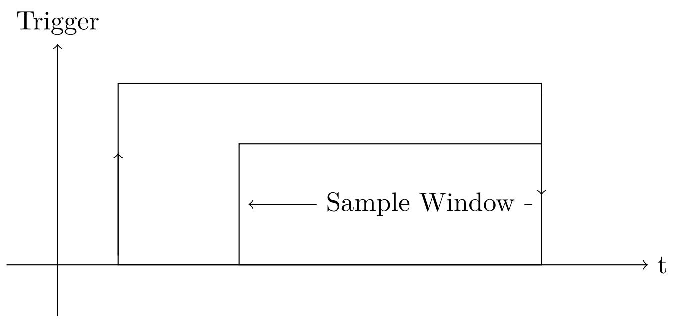


Figure 6: Dependency of the trigger and the sample window

management.

3.3.1 Trace compression

Usually there is a high information redundancy in power traces. This means that traces may be compressed to reduce the number of points to analyze and therefore reduce the complexity of the DPA attack without loosing much information. Redundancy comes from the fact that the oscilloscope will sample the power consumption several times during a clock cycle of the Smart Card. Basic possibilities of trace compression are:[2, p. 82f]

- Keeping only the maximum value per clock cycle
- Integrating over the whole trace
- Integrating over a window
- Calculation of the mean in a window

Try out different compression techniques with different window sizes and log your results to find the one that works the best for you.

3.3.2 Memory management

To handle the big amount of data especially while performing the second-order DPA, you have to care about the memory management.

Think about the amount of data which you are loading into the RAM while processing the power traces. Try and find an strategy to keep the RAM usage low while still being able to process all the data. One common strategy would be processing data in parts at a time and later combining the intermediate results to calculate the final result. A useful resource is:
`http://blogs.mathworks.com/loren/2014/12/03/reading-big-data-into-matlab/`

3.4 DPA countermeasures

DPA attacks work because the power consumption of the SmartCard depends on the computation of intermediate values in a cryptographic algorithm as it is run within the microcontroller. The goal of DPA countermeasures is to avoid or at least to reduce these dependencies. The two major types of countermeasures are: hiding and masking[2, p. 167].

3.4.1 Hiding

The goal of hiding is to minimize the correlation between the values of the secret data being processed and the power they consume. Basically there are two approaches to break this dependency:

- Randomize the current consumption for the operations.
- Maintain the same current consumption for each operation and each value.

Table 1 shows the different possibilities to reach these two goals for both hardware and software implementations.

	Software	Hardware
Random current consumption	Time axis (shuffling, dummy operations)	Time (shuffling, dummy operations) and amplitude axis (increase noise ratio)
Equal current consumption		Reduce signal (cell level)

Table 1: Hiding possibilities in software and hardware

Since for the lab only software countermeasures are viable, hiding may only be performed in the time axis. This means that two options are possible, randomizing the time when operations happen by shuffling them in time or affecting the traces alignment by adding dummy operations at random times during the execution of the decryption routine. Both of these countermeasures need to be implemented and tested for the second part of the lab.

3.4.2 Masking

Masking countermeasures aim to make the power consumption independent of the intermediate values. They do this by modifying the intermediate values in such a way that an attacker will not be able to predict them, and thus will not be able to compute a valid hypothesis. This is done by introducing random values which are combined with the intermediate values of the block cipher and removed at a later time. Since the internal values are protected by these other values, we say that they are "masked". There are different methods to apply a mask. For AES we make use of a Boolean Mask, where the operation used is an XOR function. Section 4.3.2 provides a deeper explanation and describes the implementation of this method.

3.5 Attacking the countermeasures

3.5.1 Attacks on hiding

There are a few approaches to attack hiding. Firstly, it is possible that your attack script will still work by using enough traces. Secondly, you can improve the attack using trace compression. Finally, another possibility is to align the power traces as a pre-processing step and later perform the DPA attack. Read more about attacking hiding countermeasures in [4].

Tip: Talk to your group mates to understand how the countermeasures were implemented and come up with a plan to attack them.

3.5.2 Attacks on masking

The first attack to try out is a conventional DPA attack with a high number of traces. An attack may still be successful when there is an error in the implementation or a large bias in the random numbers used for the masks. After this has been tested, the next step is to improve the attack scripts to perform what is called a second-order attack. Second-order DPA attacks exploit the joint leakage of two intermediate values instead of a single instance. Therefore, even in a protected implementation the secret key may be extracted. Read more about second-order DPA attacks in [4].

Tip: If you are using the masking scheme presented in the introductory lecture, before using randomly generated numbers for the masks use the same fixed values for all of them. This will help you make sure that your second order attack scripts work correctly.

4 Smart Card Emulator

One of the most important tasks during the laboratory is the creation of the code for your Smart Card emulator. The emulator is used in the first part of the lab to create a clone of the reference card and in the second part to assess the strength of your countermeasures against side-channel analysis. The code for the smart card emulator for this laboratory should be composed of three main blocks which are:

- Operating System
- Communication Interface
- Cryptographic Block

4.1 Operating System

Operating System (OS) is the software that takes care of the basic hardware functions of a device such as peripheral control and provides common services for other applications. For this laboratory the Operating System will be very simple. Its functionality is limited to managing the code that takes care of configuring the hardware, interpreting the commands being received through the Communication Interface, and controlling the inputs and outputs to the Cryptographic Block. In this sense it can be thought as a *protocol state machine*, more than an actual OS, where the behavior of the Smart Card at a given time depends on its current state.
For the basic functionality required for the lab there are three active states to be taken into account: *Power-up*, *Wait* and *Execute Command*. During Power-up the hardware is configured and the initialization routines are executed. This state is finished by sending an Answer-to-Reset to the terminal (described in Section 4.2.1). The card then waits for commands to be sent by the terminal, we call this the Wait state. Once commands are sent, the Communication Interface block is in charge of sampling the signals, transforming them into bits and concatenating these bits into each of the bytes which ultimately form the commands. Error detection and correction mechanisms should be in place during this process to ensure correct data reception. If the command is correctly received, then the card goes into the Execute Command state where the command is decoded and executed. After completing execution the card will return to the Wait state. These states and the transitions between them are shown in Figure 7.

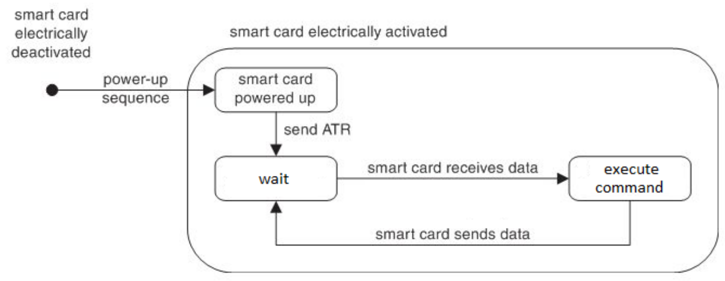


Figure 7: Flow chart of the Smart Card states

4.2 Communication Interface

Data communication occurs over the ISO7816 I/O contact on the card. Communication between the card and the terminal is asynchronous and half duplex. This means that the I/O pin is bidirectional and is used for both transmitting and receiving data. The electrical characteristics specified in ISO7816 for serial communication are similar to the ones used in the more widely known RS-232 standard. During the Wait state the I/O line is set to a high level using a pull-up resistor. Data transfer of a character is initiated by a start bit, which sets the I/O signal to low for one elementary-time-unit (etu). The start bit is followed by eight data bits, one parity bit and two stop bits (see Figure 8). The parity bit is set to high if the count of bits set to high in the data section is odd and is set to low if the count is even. The time between the stop bit and the next start bit receives the name of guard time. After the stop bit, the I/O line returns to a high level. If the receiver does not detect a parity error, it waits for the next start bit after the guard time. If an error is detected, the receiver must indicate this by setting the line low for one etu starting at half an etu of the stop bit. The transmitter must check the I/O line during the stop bit and if it is low re-send the character. The signal for a transmission error is shown in Figure 9.[4, p. 256].

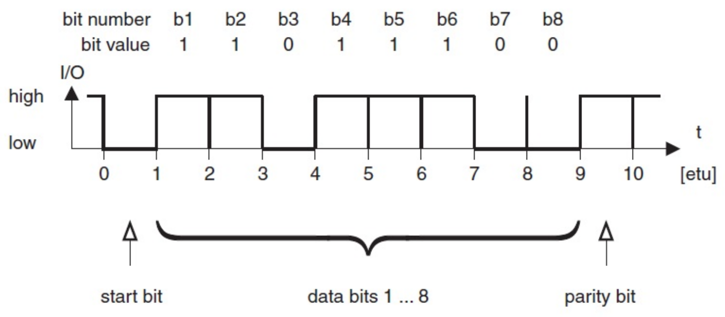

bit number b1 b2 b3 b4 b5 b6 b7 b8
bit value 1 1 0 1 1 1 0 0
I/O
high
low
t
0 1 2 3 4 5 6 7 8 9 10 [etu]
start bit
data bits 1 ... 8
parity bit

Figure 8: ISO 7816 character framing showing the initial character TS with the direct convention, which is indicated by the value $3B_{16} = 00111011_2$

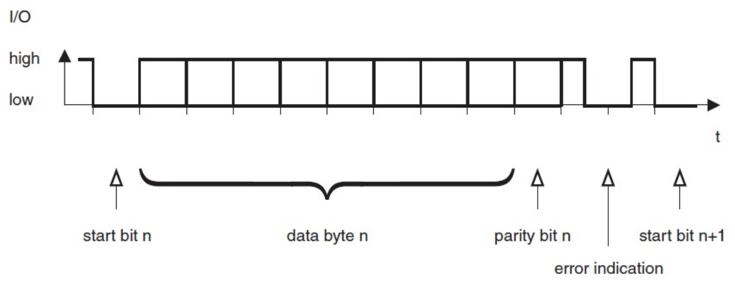


Figure 9: Signaling a transmission error with the T = 0 protocol by setting the I/O line low during the guard time

Tip: To avoid errors caused by spurious signals, sampling may be improved using multiple samples in conjunction to decision rules. Take a look at [4, p. 247] for more information.

The duration of a bit is called an elementary-time-unit (*etu*) (see Figure 8).[4, p. 206]. The etu is equal to the count of F/D clock cycles on the CLK pin. Where F and D are the transmission parameters: F is the clock rate conversion integer and D the baud rate adjustment integer [1, p. 13]. The default values are $F = 372$ and $D = 1$.

$$1\ etu = \frac{F}{D} \cdot \frac{1}{f}$$

Tip: The communication between the card and the terminal occurs asynchronously, this means there is no shared timing signal between them. To avoid errors during sampling, synchronization between the clock within your Smart Card emulator and the communication signal is needed.

4.2.1 Answer-to-Reset

After plugging in the Smart Card into the terminal, the card receives power (VCC) and a clock signal (CLK) and the controller has time for initialization. The terminal opens the communication by setting the reset signal (RST) to high. As a result, the card sends the so called "Answer to Reset" (ATR) over the I/O-pin. The ATR sequence contains information about the card and establishes the communication parameters. In our case, we use the minimal ATR sequence. The characters which compose this ATR sequence are the initial character TS, the format character T0 and the optional interface characters.

- **The initial character TS**
  The TS is the first byte of the ATR and must always be sent. It specifies the convention used for the communication protocol and also contains information to determine the divisor value. There are only two codes allowed for this byte: '3B' with the direct convention (which is used in our case) and '3F' with the inverse convention. Figure 8 shows the timing of the bit sequence of the initial character.[4, p. 206]
- **The format character T0**
  T0 is the second byte of the ATR sequence and specifies which interface characters follow it. Equal to the initial character TS, the T0 is mandatory in the ATR. [4, p. 207] In the case of SiglTV the coding of the format character is 0x90.
- **The interface characters**
  The interface characters consist of TA_i, TB_i, TC_i and TD_i bytes and specify all the transmission parameters of the protocol. There are default values defined for all the transmission protocol parameters, because this bytes are optional. In case of the ATR used in the lab, the interface characters $TA_1 = 0x11$ and $TD_1 = 0x00$ are used to complete the sequence. [4, p. 207]

Figure 10 shows the reset, I/O and clock signals during an ATR sequence. You should be able to recognize timing diagrams like this one using the provided logic analyzer.

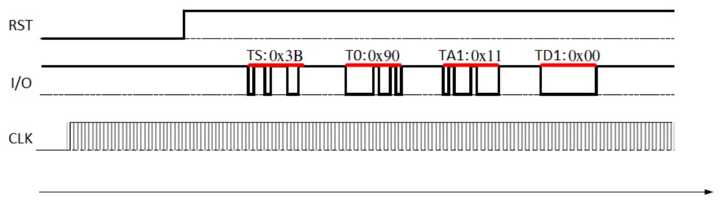


Figure 10: Answer to Reset

Note: If a valid ATR sequence is not received, the terminal will shut down the clock signal after a timeout period is reached. Keep in mind to use the provided "programm helper" to flash the microcontroller.

4.2.2 T=0 Protocol

After the conclusion of the ATR, the card waits for instruction from the terminal. As shown in Figure 11, the command structure consists of a header containing a class byte (CLA), an instruction byte (INS) and three parameter bytes (P1 to P3) [4, p. 255]. It can be optionally followed by a data section.

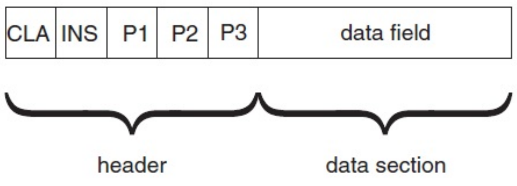


Figure 11: Command structure with the T = 0 protocol [4, p. 255]

After transmitting the header as a string of five bytes, the terminal waits for a procedure byte. Table 2 shows the three types of procedure bytes.[1, p. 23]

- If the value is ’60’, it is a NULL byte. It requests no action on data transfer. The interface device shall wait for a character conveying a procedure byte.[1, p. 23]
- If the value is ’6X’ or ’9X’, except for ’60’, it is a SW1 byte. It requests no action on data transfer. The interface device shall wait for a character conveying a SW2 byte. There is no restriction on SW2 value.[1, p. 23]
- If the value is the value of INS, apart from the values ’6X’ and ’9X’, it is an ACK byte. All remaining data bytes if any bytes remain, denoted D_i to D_n, shall be transferred subsequently. Then the interface device shall wait for a character conveying a procedure byte.[1, p. 23]

- If the value is the exclusive-or of 'FF' with the value of INS, apart from the values '6X' and '9X', it is an ACK byte. Only the next data byte if it exists, denoted D_i, shall be transferred. Then the interface device shall wait for a character conveying a procedure byte.[1, p. 23]
- Any other value is invalid.[1, p. 23]

Byte	**Value**	**Action on data transfer**	**Then reception of**
NULL	'60'	No action	A procedure byte
SW1	'6X($\neq$'60'), '9X'	No action	A SW2 byte
ACK	INS	All remaining data bytes	A procedure byte
	INS $\oplus$ 'FF'	The next data byte	A procedure byte

Table 2: Procedure bytes [1, p. 23]

Tip: Use the Logic Analyzer with the reference card to get more information about the instructions and the protocol.

4.3 Cryptographic Block

The Advanced Encryption Standard (AES) is the cryptographic algorithm used in the laboratory. This block takes as input the ciphertext corresponding to the encrypted chunk-key k_{cc}, decrypts it using the master-key k_m and returns the plaintext chunk-key k_c. Therefore you only need to implement the *decryption* function. Specifically, decryption using AES-128 in ECB mode.

4.3.1 Basic AES Implementation

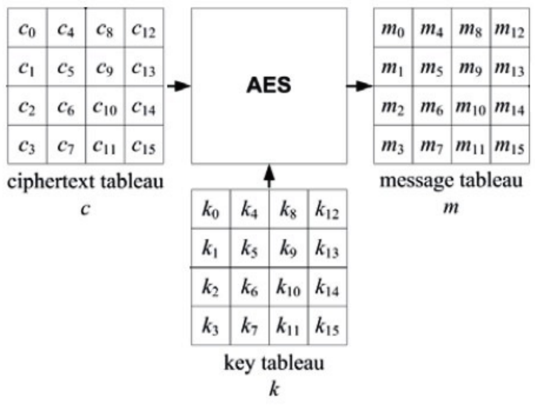

c_0 c_4 c_8 c_{12}
c_1 c_5 c_9 c_{13}
c_2 c_6 c_{10} c_{14}
c_3 c_7 c_{11} c_{15}
ciphertext tableau
c
AES
m_0 m_4 m_8 m_{12}
m_1 m_5 m_9 m_{13}
m_2 m_6 m_{10} m_{14}
m_3 m_7 m_{11} m_{15}
message tableau
m
k_0 k_4 k_8 k_{12}
k_1 k_5 k_9 k_{13}
k_2 k_6 k_{10} k_{14}
k_3 k_7 k_{11} k_{15}
key tableau
k

Figure 12: Inputs and output for an AES decryption

The AES decryption block takes as inputs 16 bytes corresponding to the ciphertext and the 16-byte master key, and returns as output the 16-byte plaintext. Within the decryption block bytes are arranged in a state matrix, as shown in

Figure 12. The state matrix is updated by applying several transformations. Figure 13 shows the block diagrams which describe the order of transformations used during encryption (a) and decryption (b) for AES. The details of each transformation can be found in [3].

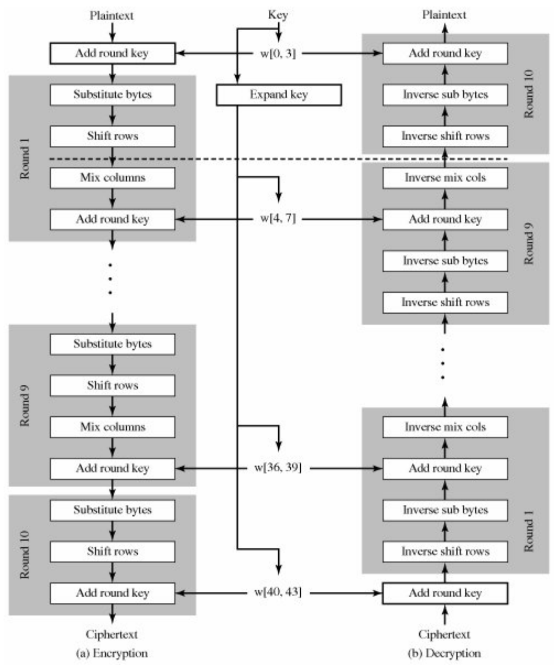


Figure 13: Block diagram of the AES encryption and decryption

Debugging AES may be tricky given that the properties of a block cipher will make a change in a single bit at the input generate a completely different output. Therefore, one of the most common ways to test the correctness of the implementations of cryptographic algorithms is making use of test vectors and compare inputs and outputs with known values. You can obtain the vectors for AES from the NIST website:
Test Vectors for AES (ECB Mode)
`http://csrc.nist.gov/groups/STM/cavp/documents/aes/KAT_AES.zip`

Tip: Set the trigger pin when an AES decryption starts and clear it once it finishes. This pin is used to align the power traces and to measure the run-time duration of your algorithm.

4.3.2 Hardened AES Implementation

In this section two basic techniques to protect the AES are discussed and an introduction to random numbers is given.

Hiding - SW possibilities in MCUs
As mentioned in Section 3.4.1, the main software possibilities of hiding are to randomize the current consumption over the time axis. Shuffling executes the critical operations randomly in a certain time slot and dummy operations moves the execution time of the critical operations over the time axis.[2, p. 167]

Masking
For the implementation of masking, you need six independent random variables for the masks m, m', m_1, m_2, m_3 and m_4. Figure 14 shows which masks are applied on which state matrices in an AES encryption process. The masks m and m' are the input and output masks for the SubBytes operation and the masks m_1, m_2, m_3 and m_4 are the input masks for the MixColumns operation. Therefore the two following precomputations are needed [2, p. 229ff]:

- Masked S-box table S_m such that $S_m(x \oplus m) = S(x) \oplus m'$.
- Output masks for the MixColums operation by applying this operation to (m_1, m_2, m_3, m_4) to generate the masks m'_1, m'_2, m'_3 and m'_4.[2, p. 229]

Random source
The random source is an important element of the implementation of countermeasures. Therefore this section gives an introduction to random numbers. Following random number application exists:

- **Quasirandom Numbers (QRN)**
  Deterministically generated sequences with special statistical properties to minimize, e.g., discrepancies in the calculation of integrals (Monte-Carlo integration). In general repeatable, usually predictable.
- **Pseudorandom Numbers (PRN)**
  Deterministically generated sequence which cannot be discriminated from true random numbers by several statistical tests. In general repeatable, not necessarily unpredictable.
- **True Random Numbers (TRN)**
  Sequences generated from physical random processes. In general unpredictable and unrepeatable.
- **Cryptographically Secure Pseudorandom Numbers (CSPRN)**
  Unpredictability is required, not necessarily unrepeatability.

To handle random numbers we need the following definitions:

- Let X be a discrete random variable that takes values from {0,1}. Then the quantity
  $$\epsilon = Pr[X = 0] - 1/2$$
  is called the bias of X. (German: Schiefe)
- Let X be a discrete random variable that takes values from {0,1}. Then the quantity
  $$\epsilon' = Pr[X = 0] - Pr[X = 1]$$
  $$\epsilon' = Pr[X = 0] - (1 - Pr[X = 0])$$
  $$\epsilon' = 2Pr[x = 0] - 1$$
  is called the imbalance of X. (German also: Schiefe)

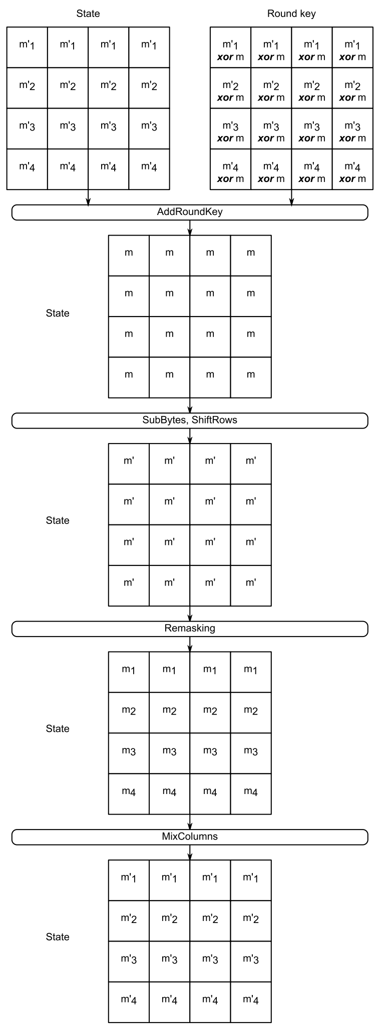


Figure 14: AES states with the different masks [2, p. 230]

- Piling-Up Lemma:
  Let X_j be independent discrete random variables with biases e_j, j = 1, 2, ?, k. Then the bias ϵ of the sum X = $X_1 \oplus X_2 \oplus ? \oplus X_k$ is given by

  $$\epsilon = 2^{k-1}\epsilon_1\epsilon_2 \cdots \epsilon_k$$

  Writing the Piling-Up Lemma with imbalances, we find

  $$\epsilon' = \epsilon'_1\epsilon'_2 \cdots \epsilon'_k$$

The general structure of cryptographic random number generators is shown in Figure 15.

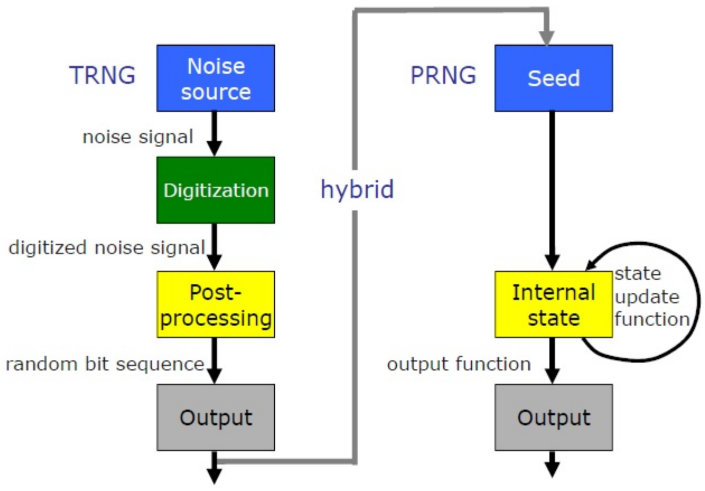


Figure 15: General structure of cryptographic RNGs

References

[1] ISO/IEC 7816-3:2006(E).

[2] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks*. Springer, 2007.

[3] NIST FIPS Pub. 197: Advanced encryption standard (aes). *Federal Information Processing Standards Publication*, 197:441–0311, 2001.

[4] Wolfgang Rankl and Wolfgang Effing. *Smart Card Handbook*. John Wiley and Sons, 2010.

Acknowledgements

Thanks to Thomas Zeschg, tutor during the summer semester 2015, for his help in the creation of this script.