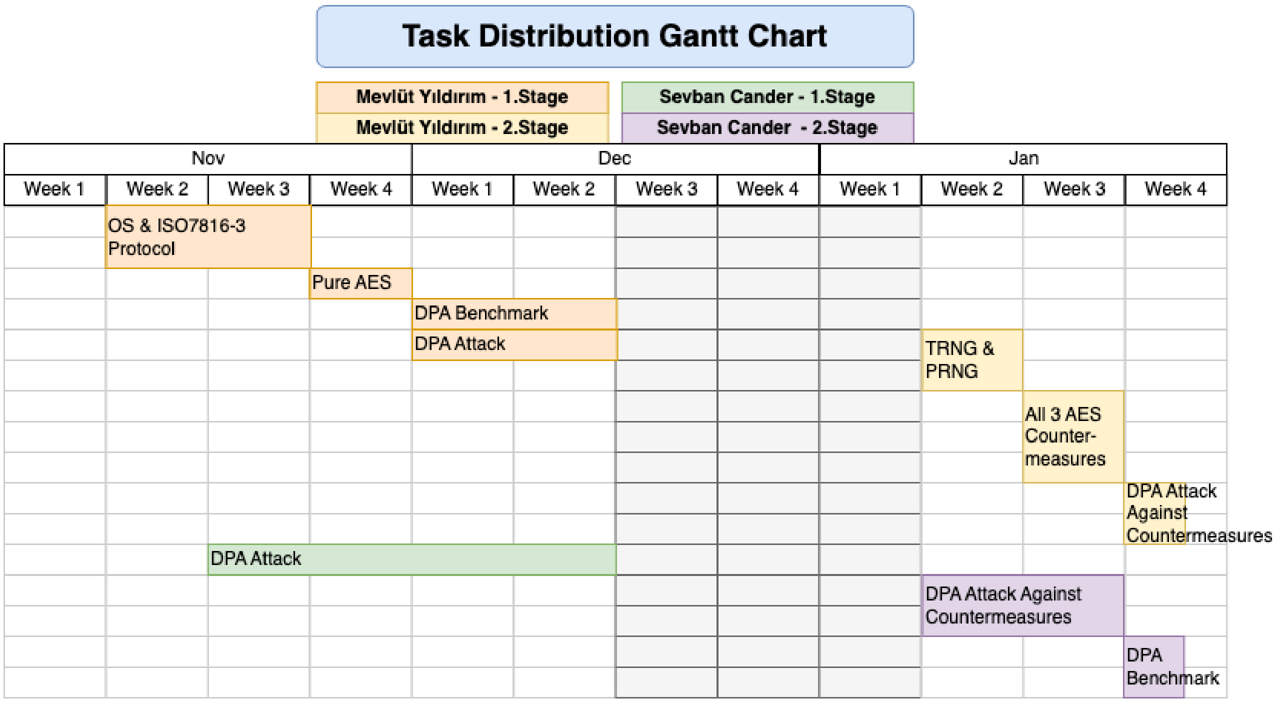
TUM Smartcard Lab
Project Report

January 22, 2025

Mevlüt Yıldırım - Sait Sevban Cander

Group 2



1 First Stage

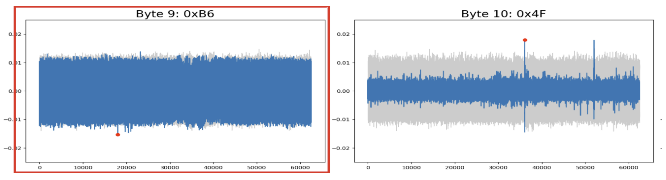
1.1 DPA Attack

1.1.1 Measurement and Setup

In the first phase of the DPA, power measurements of the smartcard are taken during data decryption. A PC Oscilloscope with a measurement script is used for this, triggering power traces while the smartcard decrypts random data. Measurement settings like the number of samples and traces can be adjusted. The sample count controls how much of the process is recorded, and too much data can disrupt the attack, so it must be set carefully. The trace count determines how many decryption cycles are measured—more traces improve accuracy but require extra time and storage.

1.1.2 DPA implementation and Optimization

The second and main phase of the DPA attack involves executing the attack and recovering the key. First, the weak points of AES for DPA are analyzed. The S-box transformation is identified as a suitable target for the attack, as the power measurements reveal information about the 16-byte key based on the number of transistors that change state during this transformation.
Compression
In this project, three windowing techniques are used to eliminate redundant information and improve computational efficiency. The first, called "squared," highlights larger values. The second, "absolute," considers negative values as well. The third, "max," focuses on the maximum value within an interval, also emphasizing larger values.
Correlation
After compression, the next step in the DPA attack is calculating the correlation between hypothetical power consumption and patterns in the power traces. The attack is targeted at the time of the S-box transformation, with hypothetical power consumption computed using the Hamming weight model. To enhance performance, a precomputed look-up table reduces computational cost. Pearson Correlation is chosen for its effectiveness in DPA attacks. Following the correlation, 16 key bytes with the highest correlation are identified. In some cases, the correlation is weak, while in others, there is a distinct peak in the correlation graph. Examples of poor and strong correlation results are shown in the figure below.



The difference in correlation graphs arises from varying parameters, such as the number of traces or the window size, which affect the results.

1.1.3 Results and Analysis

As a result of our DPA attack, we identified the Master Key as: `0xA2, 0xD0, 0xF1, 0x24, 0x4A, 0xAA, 0x94, 0xD0, 0xA7, 0x25, 0x4F, 0x26, 0xEB, 0x6D, 0x88, 0x05`. We then successfully decrypted the video.
Using the wide range of different parameters, it is better to use a benchmarking to see the each effect separately. For this purpose, benchmarking is applied to the DPA attack. Some important results from this benchmarking are these:

- The minimum number of traces that the key can be recovered is 200 with this method.
- With same window size, squared compression performs better than other compression techniques. It can recover the key with smaller number of traces.
- Small window size and large window size is not a good choice for compression. Optimal window size for 62500 sample count is around 7.

1.2 SmartCard OS & ISO7816-3 Protocol Implementation

1.2.1 USART Configuration

Steps for SmartCard-compatible USART configuration:

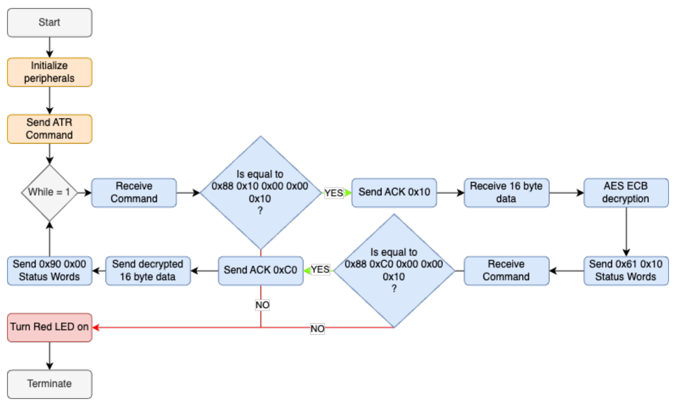
- Configured the packet structure for ISO protocol: 8 data bits, 1 even parity bit, and 1.5 stop bits.
- Set the correct baud rate $(OS\_CLOCK/372)$ and enabled SmartCard mode.
- Used polling for communication with custom functions, where data transfer/reception was managed by checking the transmitter/receiver buffer status.
- As the protocol is half-duplex (single-wire), mode was switched before each packet transfer/reception.

1.2.2 ISO7816-3 Protocol Implementation

Preparations for cloning the protocol:

- Analyzed the reference card communication by monitoring T=0 protocol commands.
- Extracted and understood these commands for implementation on the clone.

To initiate communication, the OS uses the configured USART to send the ATR command before entering the main loop. The protocol is managed within the `start_iso7816` function, which handles all steps for transferring a single packet. These steps are illustrated in the state machine diagram below:



Note: Error handling in this implementation is not designed to reflect real-world scenarios thoroughly. The reason is that no errors were encountered during testing. To keep the OS implementation simple, the system responds to errors by lighting a red LED and terminating the program. However, we acknowledge the importance of error correction/recovery mechanisms in real-world applications.

1.2.3 AES Implementation

Optimizations in AES Implementation:

- The S-Box and Reverse S-Box are precomputed, enabling fast substitution without runtime calculations.
- Galois Field multiplication constants for MixColumns are hardcoded for improved efficiency.
- Fixed memory allocation is used, with no dynamic memory operations.
- All round keys are precomputed and stored in a fixed-size array, reducing runtime overhead.

Note: AES execution time and memory usage details are provided alongside the results of the hardened AES implementations in Stage 2. Please refer to that section for the results.

2 Second Stage

2.1 Hardening of The OS and AES

2.1.1 Random Number Generator

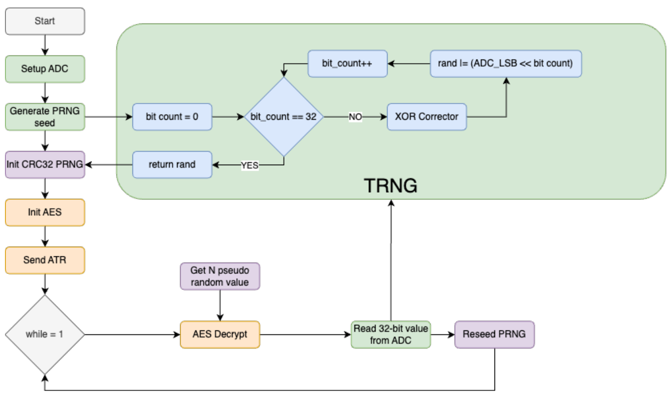
TRNG
The ADC is configured as an entropy source for the True Random Number Generator (TRNG) using the following settings:

- **Noise Sources:** Internal temperature sensor introduces natural variability and noise.
- **Sampling Time:** A sampling time of 13.5 cycles is selected to balance noise capture and conversion speed.
- **Resolution:** The ADC operates at 12-bit resolution.
- **Channel Selection:** Internal temperature sensor channel is sampled to leverage its inherent noise.
- **LSB Extraction and Correction:** Only the least significant bit (LSB) of each conversion is extracted. An XOR-based correction is applied to ensure randomness by introducing variability into the raw LSB sequence.

A 32-bit random number is generated by performing 64 ADC conversions. In this method, the LSBs of two consecutive ADC conversions are XORed to generate a new bit, which adds randomness and reduces correlation between consecutive bits. The resulting bits are then packed into a 32-bit value, and the final random number is returned for use.

PRNG
As a PRNG, SHA-256, which is cryptographically secure, was initially tested. However, due to its operational latency, countermeasure-enabled operating systems failed to decrypt the video stream in time, leading to timeouts. For this reason, CRC32, a lightweight and faster alternative, was used for pseudo-random number generation, despite not being cryptographically secure.
To enhance its security, the CRC32 seed is reinitialized by reading a new 32-bit value from the ADC after decrypting each video stream packet. This approach aims to provide an additional layer of security, compensating for the lack of cryptographic strength in CRC32. *The core CRC32 code was sourced from GitHub (crc32.c) and modified as necessary before being integrated into the operating system.*
Note: While CRC32 is typically used for error correction and checksums, it was chosen here as a practical solution to meet the goals of this laboratory course.

Below is a flowchart illustrating the TRNG and PRNG process, showing the steps involved in random number generation.



2.1.2 AES Countermeasures

Shuffling

- **Permutation Creation:** A permutation array of 16 elements was generated, with each element representing an index in the AES state array. The **Fisher-Yates algorithm**, combined with the CRC-32 PRNG, was used to shuffle the array, ensuring randomization.

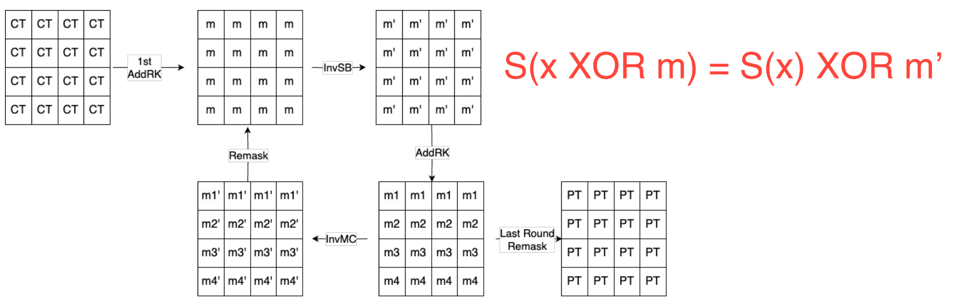
- **Shuffled AES Operations:** The permutation was applied during the `AddRoundKey` and `InvSubBytes` stages of AES decryption. It determined the order of state and key accesses, effectively randomizing the execution sequence of these critical operations.

Dummy Operations

- **Fixed Number of Operations:** A total of **100** dummy inverse sub-byte operations are added per AES decryption. This fixed count prevents attackers from deducing information. The value is adjustable via a macro.
- **Dummy Operation Logic:** A random byte is generated using the PRNG, its inverse S-box value is computed, and the result is XORed with a final dummy result to ensure inclusion in computations. To prevent the compiler from optimizing dummy operations, results are stored in a `volatile` variable.
- **Random Distribution:** Dummy operations are randomly allocated across AES rounds using the PRNG, while the remaining operations are added to the final round to maintain the total count.
- **Random Ordering:** Within each round, dummy operations are inserted either before or after the actual inverse sub-byte operation with a 50% probability determined by the PRNG.

Masking

- **10 Masks:** `m1-m4`, `m`, `m'`, and `m1'-m4'` are utilized. `m1-m4`, `m`, and `m'` are generated via PRNG, while `m1'-m4'` are derived from `m1-m4`.
- **InvSubBytes Masks:** `m` and `m'` are applied as input and output masks during `InvSubBytes`, including masked reverse S-box creation.
- **InvMixColumns Masks:** `m1-m4` are input masks, while `m1'-m4'` serve as output masks.
- **Decryption Flow:** Mask Initialization → Masked AddRK → [InvSR → Masked InvSB → Masked AddRK → (InvMC → ReMask)] → Final ReMask.



OS & AES Results

Implementation	Time [ms]	Difference [%]
Reference card (no countermeasure)	54.1	N/A
Own OS No Countermeasure	51.5	-4.8%
Own OS Dummy Ops.	53.9	+4.66%
Own OS Shuffling	51.8	+0.58%
Own OS Masking	53.7	+4.27%
Own OS All 3	56.6	+9.90%

(a) Execution Time (for 1 block 16-byte) Results

Implementation	Text [kB]	Data [B]	BSS [B]	Total [B]	Difference [%]
Own OS No Countermeasure	3.8	8	176	3952	N/A
Own OS Dummy Ops.	5.856	28	192	6036	+52.77%
Own OS Shuffling	5.716	28	192	5972	+51.15%
Own OS Masking	6372	28	448	6848	+173.27%
Own OS All 3	6660	28	452	7140	+180.66%

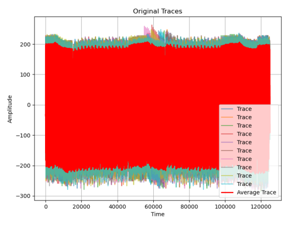
(b) Memory Usage Results

2.2 DPA Attack Against Countermeasures

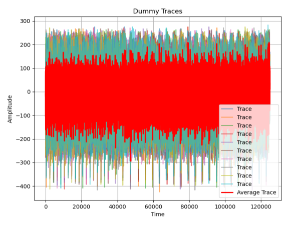
2.2.1 Attacking Own Card

In the first step, a simple DPA attack was conducted on the hardened OS developed in this project. For shuffling protection, using 1000 traces with absolute compression successfully recovered all bytes. However, with other compression techniques like squared or max, two bytes were recovered incorrectly.
For the dummy operations countermeasure, the same attack yielded no recovered bytes. Further preprocessing was attempted to improve results. The approach assumes random numbers are uniformly distributed, meaning that averaging multiple measurements could neutralize the effect of randomness. For example, with 10 power traces and random values distributed in [1,10], averaging 10 traces would theoretically yield a constant value of 5.

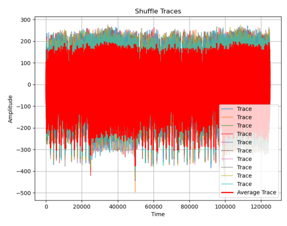
This effect is visualized in the figure below: the red section represents the averaged power trace over 10 traces, while other colors depict individual traces. The figure clearly shows that dummy operations introduce significant noise, as seen in the large differences between traces. Despite preprocessing, no key bytes were recovered. One possible reason is that averaging traces significantly reduces their number; for instance, averaging every 10 traces out of 1000 leaves only 100 traces, which is insufficient for a simple DPA attack, even in the unprotected case. Another reason might be trace misalignment. Since dummy operations occur randomly, AES operations are not consistently timed across measurements. Averaging misaligned traces likely causes the loss of critical data, hindering key recovery.



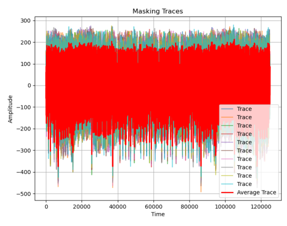
(a) No countermeasure



(b) Dummy Operations



(c) Shuffling

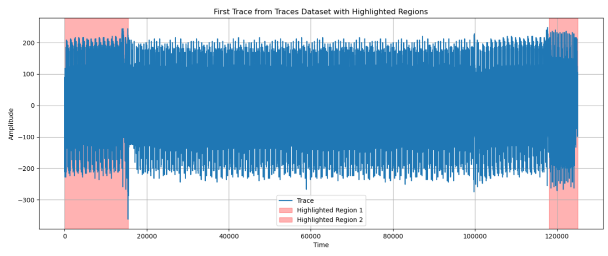


(d) Masking

2.2.2 Attacking Reference Card

In the second phase of this project, the smart card was secured with countermeasures like hiding and masking, which introduce noise into the measurements and make key recovery more challenging. To overcome this, the DPA attack needed refinement. However, with a sufficient number of traces, the same DPA attack could still succeed.

Shuffling: An attempt to recover the key from a hardened OS using shuffling was successful with 7050 traces and absolute compression with a window size of 17. Detailed results are provided in the table below. To achieve this success, some preprocessing of the raw data was necessary. The power trace included a noisy segment at the end, leading the DPA to identify maximum correlation in irrelevant regions. This is shown in the figure below, where the noisy segment is highlighted in red on the right. Since the S-box conversion does not occur in this region, removing the noisy part clarified the data. Additionally, the random timing of operations in AES caused variability in its duration. As a result, measurements were taken using a wider window from the last trigger indicating the end of AES. This, however, included irrelevant parts of the trace, highlighted in red on the left in the figure.



Dummy Operations: The same preprocessing steps were applied to traces with dummy operations. However, no stable results were observed, indicating that the protection was effective in this scenario.

Masking: For masking protection, a simple DPA attack was ineffective. Implementing a second-order DPA was considered, but due to the time constraints of this project, it was not completed, as it required significant computational effort.

2.2.3 DPA Attack Benchmark Results

Implementation	Broken [yes/no]	Min. # of Traces	Duration [s]	Compression Method	Window Size
Own Card (no countermeasure)	Yes	250	0.91	Squared	7
Own Card + Dummy Operations	X	X	X	X	X
Own Card + Shuffling	Yes	400	0.65	Absolute	13
Own Card + Masking	X	X	X	X	X
Reference Card (no countermeasure)	Yes	200	0.86	Max	9
Reference Card + Dummy Operations	X	X	X	X	X
Reference Card + Shuffling	Yes	7050	0.7	Absolute	17
Reference Card + Masking	X	X	X	X	X