NTNU | Norwegian University of Science and Technology

TTM4205 – ChipWhisperer Lab, Fall 2026

Secure Cryptographic Implementations

Assignment

This is one out of three assignments in the course TTM4205 Secure Cryptographic Implementations during the fall semester of 2026. The ChipWhisperer lab contains problems related to side-channel and fault attacks on real hardware. The problems are adjustments of the material available at github.com/newaetech/chipwhisperer. All tasks require detailed answers where you describe and document what you have done to complete them, e.g., written explanations, calculations, code, plots, and screenshots.

This assignment counts for at most 30 **points**, but the lab contains 37 **points** of work. Every part and task states how many points it is worth: the SCA Lab (8 points), the Fault Lab (11 points), the RSA Lab (12 points), and the ECC Lab (6 points). You therefore have 7 points of slack: you can skip tasks that you do not get working, or lose points along the way, and still be awarded the full score. Any points above 30 are simply capped, so there is nothing to lose by attempting everything. We may also give full or partial credit if you show that you understand a task and made a real attempt at it, even if you do not solve it entirely.

Collaboration. This assignment is solved in *groups of 2 to 3 students*. Register your group in Canvas before you start working with the assignment. You may discuss the problems freely within your group and with the course staff. Each group submits a single joint report written in your own words, all group members are expected to contribute, and copying from other groups is not permitted.

Getting help. The teaching assistants are there to help you with this lab, and the easiest way to get help is to simply ask them in person during the exercise classes and the lab sessions listed in the lecture plan on the course website. Bring your setup, show them what you have tried, and they will help you get further. You are of course also welcome to ask on the Canvas forum, which is the right place for questions that are useful for everyone.

External resources. You are *allowed* to rely on external resources to solve the lab. Every resource you rely on must be *clearly* cited with a reference. Using external material without attribution is considered cheating; see i.ntnu.no/wiki/-/wiki/English/Cheating+on+exams.

Use of AI. What you hand in must be your own work: the attacks, the analysis, the explanations, and the code must come from you. You are *not* allowed to use AI tools to solve the assignment or to produce your write-up. You *may* use AI tools to polish a finished submission, for example to tidy up code or improve wording and formatting. In this assignment, you may also use AI tools to troubleshoot the technical setup, for example to get the toolchain, drivers, or notebooks running on your machine: they may help you get the experiments running, but not run them.

Equipment and computers. All the equipment you need is available in the **CRYPTO-LAB** in Electro A175, packed in ready-made boxes; the contents are listed in Appendix A together with reference pictures. Each group needs only one box, and the boxes are numbered: use the box whose number matches your group number in Canvas, so that we always know which group has which equipment. You may take the box home with you if you prefer to work there, but you must hand the same box back in when you have completed the lab, and no later than the end of the semester. If anything is broken or missing, please report it to the course staff so that we can replace it and make sure the next group gets working equipment.

Use the computers in the CRYPTO-LAB or your own laptop; the software is not preinstalled in the other computer labs on campus, and you may not have permission to install it there.

Lab access. All students enrolled in the course get card access to the CRYPTO-LAB, and the room is never booked for teaching, so you can work there whenever it suits you, also outside the scheduled sessions.

Setup. Install guide: chipwhisperer.readthedocs.io/en/latest/getting-started.html. The assignment files are in the *jupyter/lab* sub-folder of git.ntnu.no/ie-iik/chipwhisperer-lab-ttm4205. This repository replaces NewAE's chipwhisperer repository in the installation guide above. If you use your own laptop or Windows, we also provide a recommended virtual-machine configuration with Vagrant at git.ntnu.no/ie-iik/chipwhisperer-v6-vagrant. This requires Vagrant and VirtualBox with Extension Pack; once both are installed, the virtual machine is configured with the bash/PowerShell command `vagrant up` run inside your `chipwhisperer-v6-vagrant` folder. Appendix B lists problems that students often run into, and how to solve them.

Start with the notebook `1 - Connecting to Hardware.ipynb`: it checks that your installation works and that the ChipWhisperer `Husky` is detected, and it is the fastest way to find out whether your environment is set up correctly before you begin. The ECC Lab has its own preparation notebook, `ECC_lab_setup.ipynb`, which you run once before `5-ECC_lab`. Notebooks are not graded.

Report. You do not need to document your setup in the report. What we do expect is that your results are presented so that a reader can follow them: label the axes of every plot, and explain in the text what the plot shows and what you conclude from it. Include a few key lines of code where they help explain what you did, for example the part of a glitch loop that decides what counts as a success, but do not include all of your code, neither in the text nor as an appendix.

All submissions must be written in LaTeX, and we provide a mandatory report template to be used at overleaf.com/read/mrpwqqxdkbfv#667c9c.

Submission. Submit your work in Canvas; the submission link is available from the course website at ttm4205.iik.ntnu.no.

Deadline. December 4th at 23:59.

Contents

1 SCA Lab (8 points)

This section contains tasks from the folder `SCA_lab`. The graded tasks are in the notebook `1-SCA_lab`. There are additional tutorial notebooks in the folder `Tutorial` that are a good introduction to ChipWhisperer. These tutorial notebooks are **not** graded.

1.1 Trace Capture (Notebook 2.1, 3 points)

Task. The AES implementation in this lab can insert a small, data-dependent delay (the `ADD_JITTER` option adds a loop that runs 0–15 times depending on the first plaintext byte). Compile and program the target both with and without this jitter, capture and plot the traces in each case, compare the two plots, and explain how you could overcome the jitter. Include the plots in your answer.

1.2 Manual Sum of Absolute Differences Resync (Notebook 2.2, 3 points)

Task. Realign the jittery traces using Sum-of-Absolute-Differences (SAD) resynchronization. Plot your traces before and after the SAD resync, describe the differences you see, and explain why realigning the traces makes the CPA attack succeed.

1.3 CPA against Jittery Traces (Notebook 2.3, 2 points)

Task. Run the CPA attack on the jittery traces. Is the attack successful before the SAD resync, and after it? Explain why in each case. Then explain what makes Dynamic Time Warping (DTW) a more effective realignment technique than SAD in this setting. Include plots of your traces together with your explanation.

2 Fault Lab (11 points)

This section contains tasks from the notebook `3-Fault_lab` in the folder `Fault_lab`.

2.1 Constructing the Glitch Loop (Notebook 3.1, 2 points)

Task. Complete the glitch loop. Include your code and an explanation of how it works. Can you detect any successful glitches? Explain and include a plot of your glitches and crashes. Repeat with a small glitch repeat value and a different external offset, then compare the results.

2.2 Glitching past a Password Check (Notebook 3.2, 2 points)

Task. Complete the partial glitch loop implementation. Were you able to glitch past the password check? What do you observe when you change the external offset and glitch repeat values? Include the different glitch plots and explain what they show.

2.3 Glitching a Memory Dump (Notebook 3.3, 3 points)

Task. Finish the implementation of the bootloader glitch loop. Are you able to recover any secret information? If so, what information is this? In the section *Diagnosing the Fault*, multiple reasons for why the glitch was possible are mentioned. Choose one countermeasure and implement it in the `bootloader.c` file built at the beginning of the Jupyter notebook, and rerun your glitching attack. Does your countermeasure work? Explain why or why not.

2.4 Power Traces for Messages (Notebook 3.4, 2 points)

Task. Explain the differences between the power traces for correctly and incorrectly authenticated messages. How can we use this to determine if our glitch is successful? Include the power-trace plots in your answer and explain how we can use the information from these to determine whether the glitch is successful or not.

2.5 Glitching to Recover the Key (Notebook 3.5, 2 points)

Task. Finish the glitch loop implementation and include a listing showing the changes you made to the code. How many ciphertexts did you need to recover the key?

3 RSA Lab (12 points)

This section contains tasks from the notebook `4-RSA_lab` in the folder `RSA_lab`.

Note. You must use the `XMEGA` target board together with the ChipWhisperer `Husky` in this lab.

3.1 Power Trace on Different Key Sizes (Notebook 4.1, 2 points)

Task. Try different keys sent to the RSA trace function. Explain what you observe and describe, in text and pseudocode, how such traces could be used to recover an RSA key.

3.2 Key Recovery (Notebook 4.2–4.3, 5 points)

Task. Finish implementing the key recovery for your chosen key: select a reference section of the trace, use a SAD match to find where that pattern repeats, and turn the matches into recovered key bits. Are you able to recover the original key? Include a listing showing your implementation and explain why it recovers the key. The recovery misses the *last* bit of the key: explain why the last bit is not recovered, and how you could recover it.

3.3 Plotting Bandwidth-Limited Data (Notebook: Bandwidth-Limited Data, 2 points)

Task. In the *Experimenting with Bandwidth-Limited Data* part of the notebook, are you able to recover the same key from the bandwidth-limited trace? Using a mixture of pseudocode and text, explain why this method of recovering the key works.

3.4 Fixing the Problem (Written Analysis, 3 points)

Task. The function `get_pt` in the firmware file `simpleserial-rsa-xmega.c` is called when you capture the RSA trace. For the sake of simplicity, this function does not perform a full RSA decryption, but part of one using 16 bytes of key material. Using this code as a starting point, explain why you are able to recover the key and give one possible countermeasure. Include pseudocode of your countermeasure and explain how it relates to the `get_pt` function.

4 ECC Lab (6 points)

This section contains tasks from the notebook `5-ECC_lab` in the folder `ECC_lab`. This lab attacks a hardware ECC implementation on the CW305 FPGA. In this exercise we do not sample our own traces, but use pre-recorded ones.

4.1 Find Ones and Zeroes (Notebook 5.1, 3 points)

Task. One of the major points of this lab is to investigate how each bit of the nonce k is processed in order to recover it. Use your own words and plots from the lab to explain how we can identify when each bit is processed and whether it is a 0-bit or a 1-bit.

4.2 The Attack (Notebook 5.2, 3 points)

Task. Perform the attack by guessing the nonce k one bit at a time, as shown in the notebook. Include your code for *guess k one bit at a time* and report the recovered k.

A Equipment

There are 42 equipment boxes in the CRYPTO-LAB, and each group needs one of them. The boxes are numbered, and your group uses the box whose number matches your group number in Canvas. This appendix lists what a box contains and shows what each item looks like.

A.1 Box Contents

- 1x ChipWhisperer `Husky` (Figure 1)
- 1x `CW313` Adapter Board with supporting circuitry for the `CW312` target boards (Figure 2)
- 1x `SAM4S` Target Board (Figure 2)
- 1x `XMEGA` Target Board (Figure 3)
- 1x `CW308T` to `CW312` Adapter Board (Figure 4)
- 1x 20-pin IDC ribbon cable (Figure 5)
- 1x USB-C cable with adapter (Figure 6)

A.2 Reference Pictures

Figure 1: ChipWhisperer `Husky`.

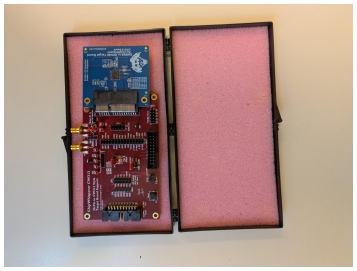

Figure 2: `CW313` board with the `SAM4S` target.

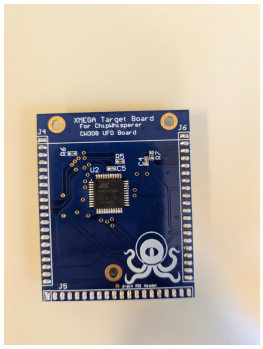

Figure 3: `XMEGA` target board and box.

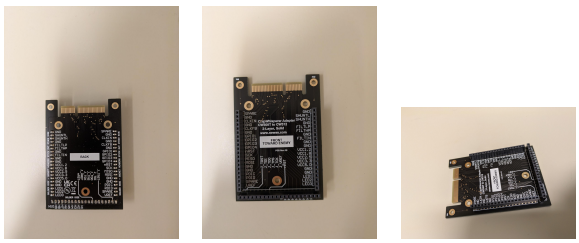

Figure 4: `CW308T` to `CW312` adapter board, seen from the back, the front, and the side.

Figure 5: 20-pin IDC ribbon cables; one is included in each box.

Figure 6: USB-C cable with adapter.

B Common Problems

B.1 Lascar

Lascar is a framework for side-channel analysis and is used in the SCA lab. If you have installed ChipWhisperer manually (macOS) or using the ChipWhisperer installer (Windows) from NewAE's website, Lascar might not be in your installation. You can install it using pip with the command

```
pip install lascar
```

inside your Python environment.

B.2 Dynamic Time Warp

For a few students, the Dynamic Time Warp (DTW) implementation in Lascar is slower than expected. Keep in mind that DTW will be slow regardless, at least compared to the SAD technique used earlier in the SCA lab. Aligning x traces to a reference trace using DTW, where the length of each trace is L, has a time complexity of $O(L^x)$. However, the DTW implementation used in the lab is an approximation algorithm, so it should have a time complexity of (roughly) $O(L \cdot r)$, with r being the radius parameter set in the lab, which defines how big shifts between the traces are allowed. Note that this is not a one-to-one mapping between the size of the radius and the size of the shift allowed. If you experience that DTW is very slow, try the following:

- reducing the number of traces you are aligning;
- reducing the radius parameter;
- lowering `scope.adc.samples` to capture fewer points per trace.

If the above does not fix your issue, please ask the teaching assistants during an exercise class or lab session, or on the Canvas forum, and we will evaluate whether you can skip this particular task and still get full score on the lab.

B.3 File or Folder Not Found Errors

If you experience file- or folder-not-found errors, make sure that the path to the folder is correct. There have been some issues with the paths missing an additional `../`. These issues should be fixed, but your notebooks only contain the fixes if you are on the latest commit on the main branch; see the next subsection for how to update.

B.4 Updating Lab Notebooks from Git

Fixes to issues with the lab will continually be uploaded to git.ntnu.no/ie-iik/chipwhisperer-lab-ttm4205. To get these updates, the easiest way is to navigate to the lab folder in your terminal and then use the command

```
git pull --autostash
```

to stash your changes, pull from the repository, and pop your changes back into the newly updated folder. You can also make a commit first and then pull by using the command

```
git add . && git commit -m "<commit message>" && git pull
```

B.5 C Code

Parts of the labs will ask you to make a few changes to and/or evaluate some C code. The relevant code for these tasks will always be in the same folder as the latest compiled code you flashed to your ChipWhisperer. The Python command that flashes to your ChipWhisperer is

```
cw.program_target(scope, prog, <PATH/compiled.hex>)
```

and the code is always found in the `firmware/mcu/` folder, relative to the root of the lab repository.

B.6 FPGA Programming Failed

If you get an error when connecting to scope like this

```
(ChipWhisperer Target ERROR|File fpga.py:127) FPGA programming failed. Typically
    ↪ either bad bitstream or prog speed too high (current 10000000)
(ChipWhisperer Scope ERROR|File __init__.py:402) ChipWhisperer error state
    ↪ detected. Resetting and retrying connection...
```

The issue is likely that the firmware is wrong and/or outdated. The issue is most likely caused because a regular ChipWhisperer Husky was flashed with the *Husky Plus* firmware (`cwhuskyplus`) during a firmware update. The board then identifies itself over USB as a Husky Plus, so the ChipWhisperer software loads the Husky Plus FPGA bitstream onto the regular Husky's FPGA. That bitstream does not match the hardware, so FPGA programming fails and `cw.scope()` never connects.

How to recognize it

- `cw.scope()` fails with:

```
FPGA programming failed. Typically either bad bitstream or prog speed too high
OSError: FPGA Done pin failed to go high, bad bitstream?
```

  often followed by an unrelated `AttributeError: 'NoneType' object has no attribute 'getProductID'` from the library's retry code (this can be ignored).

- Lowering `prog_speed` or changing cables does not help.
- `lsusb` reports a Husky Plus although the board is a regular Husky:

```
ID 2b3e:ace6 NewAE Technology Inc. ChipWhisperer Husky Plus <- wrong
ID 2b3e:ace5 NewAE Technology Inc. ChipWhisperer Husky <- expected
```

How to fix it

Flash the regular Husky firmware, which is bundled with the ChipWhisperer software (version 1.5.0 in ChipWhisperer 6.0.0).

1. Check the firmware file and permissions.

```
cat software/chipwhisperer/hardware/firmware/cwhusky/version.txt # 1.5.0
groups | grep dialout # needed to access the bootloader serial port
```

2. Erase the firmware (fresh Jupyter kernel). The board still identifies as `0xACE6`, so connect with that product ID:

```
from chipwhisperer.hardware.naeusb.naeusb import NAEUSB
usb = NAEUSB()
usb.con(idProduct=[0xACE6])
usb.enterBootloader(True) # erases firmware; fails silently
```

After a few seconds, `lsusb` should show `03eb:6124` (bootloader). If it still shows `ace6`, unplug and replug the board, or use the hardware erase button/pads on the Husky (see NewAE's Husky documentation).

3. Flash the Husky firmware. Use the explicit file path so the Husky Plus firmware is not picked up again:

```
import os, chipwhisperer as cw
fw = os.path.expanduser("~/chipwhisperer-lab-ttm4205/software/chipwhisperer/"
                        "hardware/firmware/cwhusky/mcufw.bin")
cw.program_sam_firmware(fw_path=fw) # add serial_port='/dev/ttyACMx' if needed
```

4. Verify. Unplug and replug the board. `lsusb` should show `2b3e:ace5`, and:

```
import chipwhisperer as cw
scope = cw.scope()
print(scope.fw_version_str) # 1.5.0
```

Notes

- Never use `hardware_type='cwhuskyplus'` for a regular Husky.
- An erased board is not bricked: the bootloader is in ROM, so the board can always be replugged and flashed again as `03eb:6124`.
- In bootloader mode, `/dev/ttyACM0` may be the bootloader port; in normal mode it is the Husky's target serial port, not the bootloader.

B.7 `scope.default_setup()` fails

The issue. The python module setuptools 82.0.0 removed the `pkg_resources` module one of the chipwhisperer packages dependencies. The code path for `pkg_resources` is run when executing `scope.default_setup()` in the notebooks.

How to fix it. Downgrade setuptools in the same environment Jupyter uses, then restart the kernel:

```
pip install "setuptools<82"
```

From inside a notebook, `%pip install -force-reinstall "setuptools<82"` ensures the correct environment is used. Note that a later `pip install -upgrade` may pull in setuptools 82 again.

Notes

This issue has been fixed in the latest version of the vagrant config and a patch has been uploaded to the chipwhisperer repository we use in the lab for a vanilla install, but this is not tested yet, so be aware of this potential issue when running your first notebook.