ECE 3170 Cryptographic Hardware for Embedded Systems

Fall 2025

Assoc. Prof. Vincent John Mooney III

Georgia Institute of Technology

Lab 4, 100 pts.

Due Tuesday, Oct. 28 prior to 11:55pm

(please turn in homework electronically on Canvas)

Working with the ChipWhisperer-NANO and JupyterLab

In this lab, you will work through the ChipWhisperer Setup Test and connect the ChipWhisperer-NANO board to your computer. Then, you will recover an Advanced Encryption Standard (AES) key from the internal state of an AES implementation.

If you already have Python and pip installed, you may skip the installation steps and proceed to step 8, where you will perform and test your Jupyter installation and install the ChipWhisperer library.

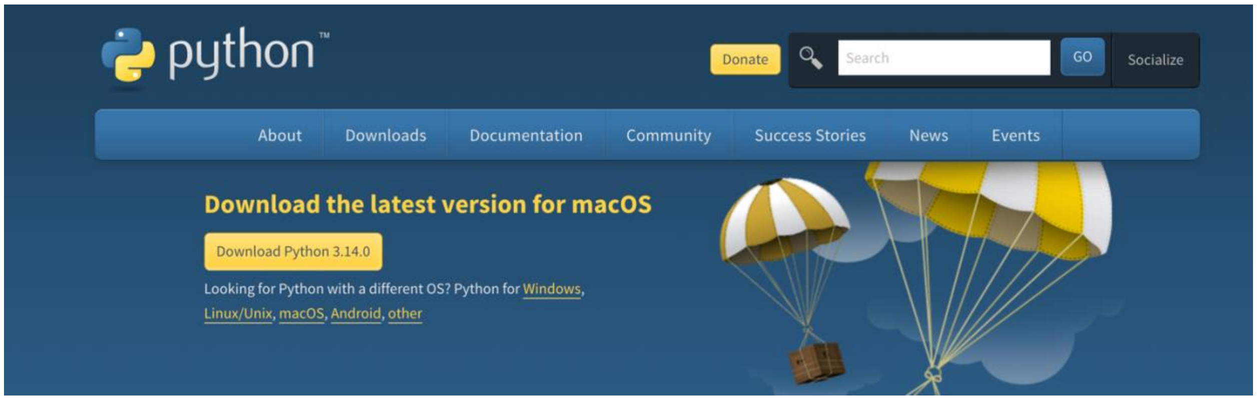
Installation

1. Installing Python and Pip

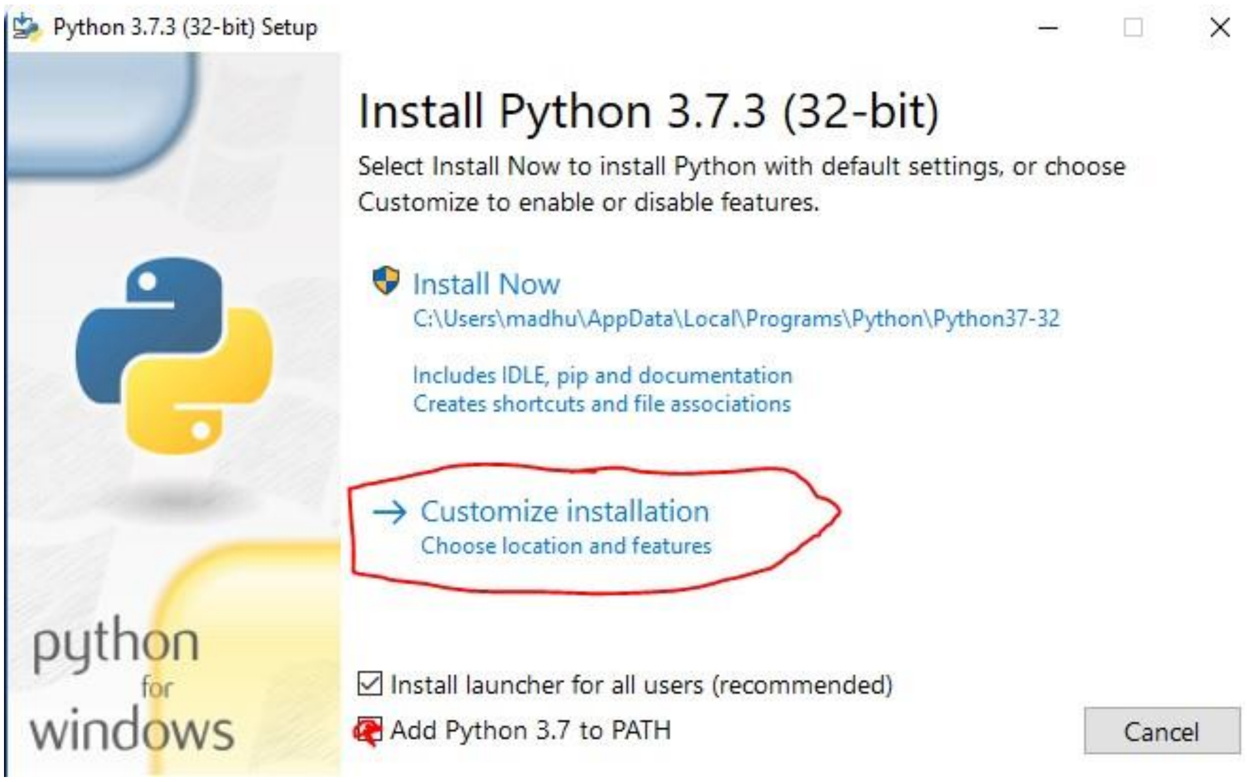
Note: If you already have Python and Pip installed, you may skip to the next section.

A. Windows Machines

1. Download the latest version of Python from https://www.python.org/downloads/.


python™
Donate
Search
GO
Socialize
About
Downloads
Documentation
Community
Success Stories
News
Events
Download the latest version for macOS
Download Python 3.14.0
Looking for Python with a different OS? Python for Windows,
Linux/Unix, macOS, Android, other

2. When running the installer, make sure you check the box to Add Python to PATH. This will allow you to use the python command in your command window.


Python 3.7.3 (32-bit) Setup

Install Python 3.7.3 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

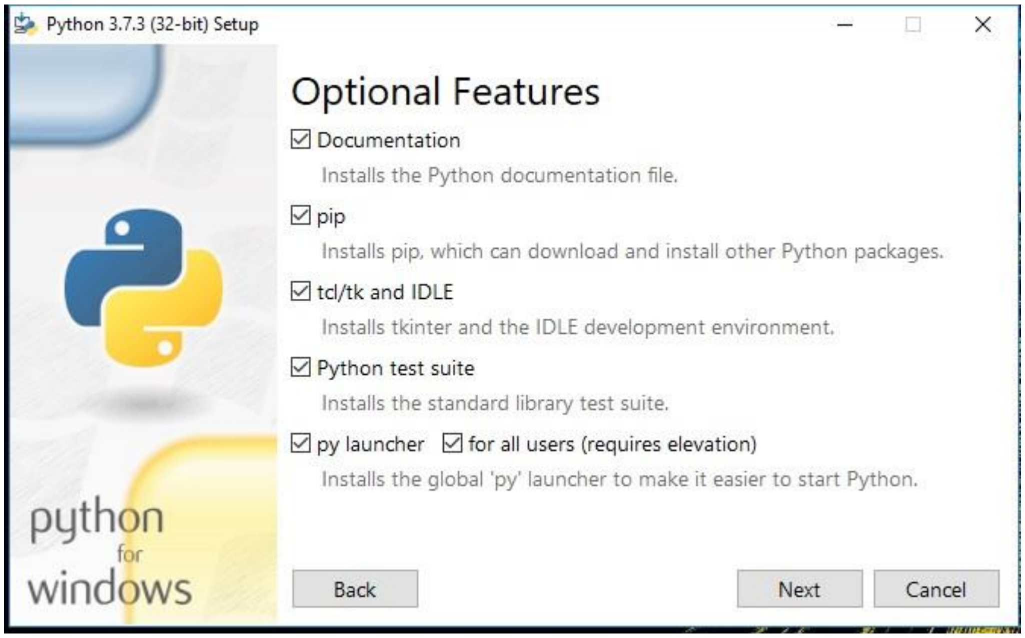
Install Now
C:\Users\madhu\AppData\Local\Programs\Python\Python37-32

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.7 to PATH

3. Select “Customize installation” and ensure **Documentation, pip, py launcher, and Python test suite are checked**. The tcl/tk and IDLE box along with the for all users box need not be checked:


Python 3.7.3 (32-bit) Setup

Optional Features

☑ Documentation
Installs the Python documentation file.

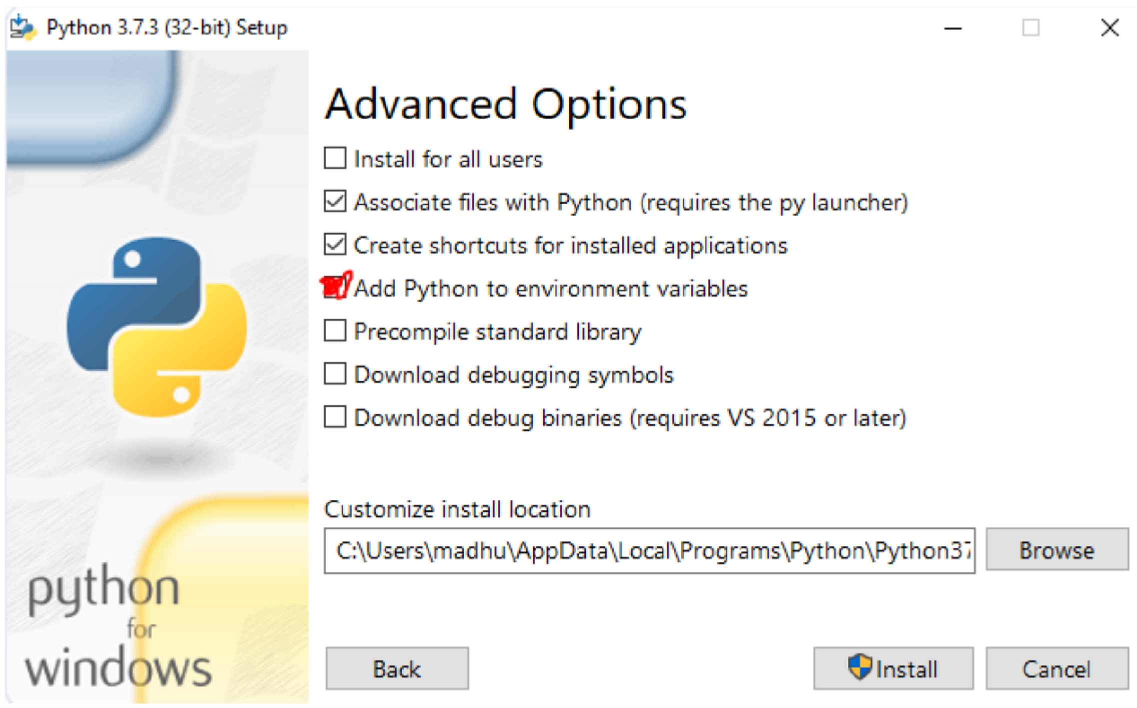
☑ pip
Installs pip, which can download and install other Python packages.

☑ tcl/tk and IDLE
Installs tkinter and the IDLE development environment.

☑ Python test suite
Installs the standard library test suite.

☑ py launcher ☑ for all users (requires elevation)
Installs the global 'py' launcher to make it easier to start Python.

4. Click Next to proceed to the Advanced Options page. Make sure you check **Add Python to environment variables, Associate Files with Python, and Create shortcuts for installed applications**. It's very important you add Python to your environment variables; otherwise, you will not be able to use the python command in the command line.


Python 3.7.3 (32-bit) Setup

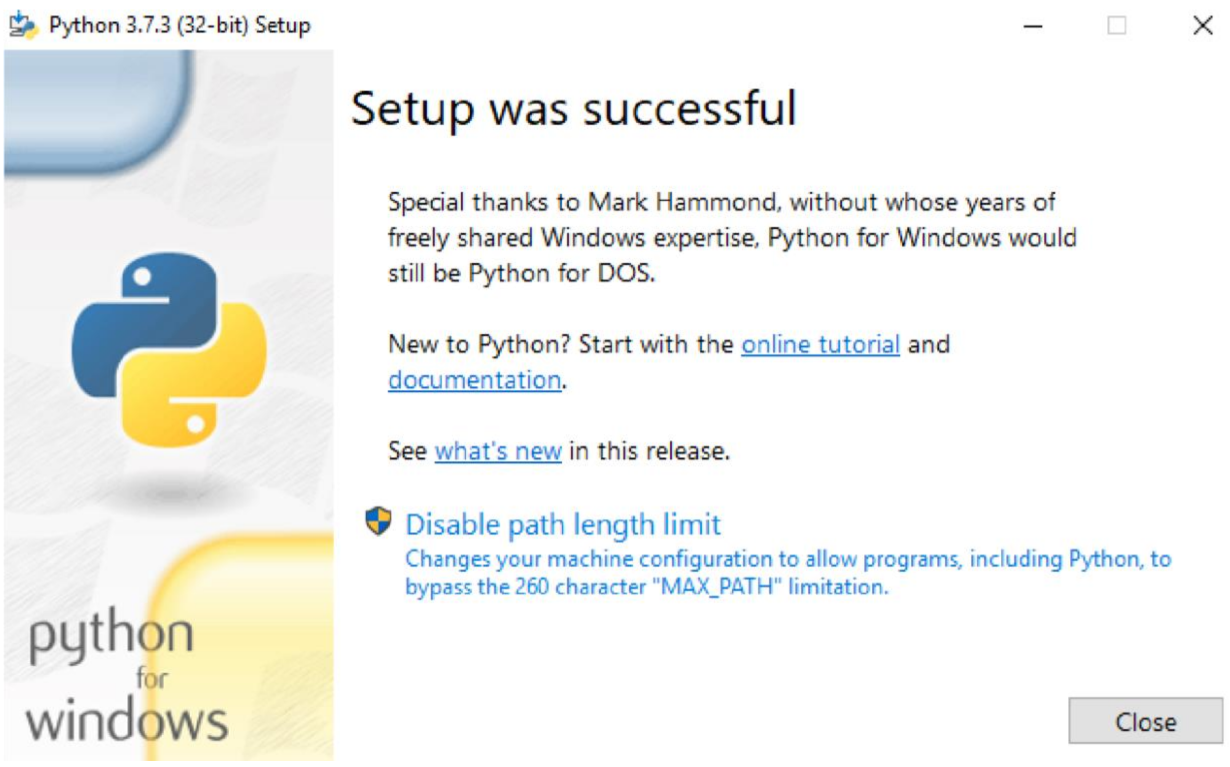
Advanced Options

- [ ] Install for all users
- [x] Associate files with Python (requires the py launcher)
- [x] Create shortcuts for installed applications
- [x] Add Python to environment variables
- [ ] Precompile standard library
- [ ] Download debugging symbols
- [ ] Download debug binaries (requires VS 2015 or later)

Customize install location

C:\Users\madhu\AppData\Local\Programs\Python\Python37

5. Click Install. You should see the following page:


Python 3.7.3 (32-bit) Setup

Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

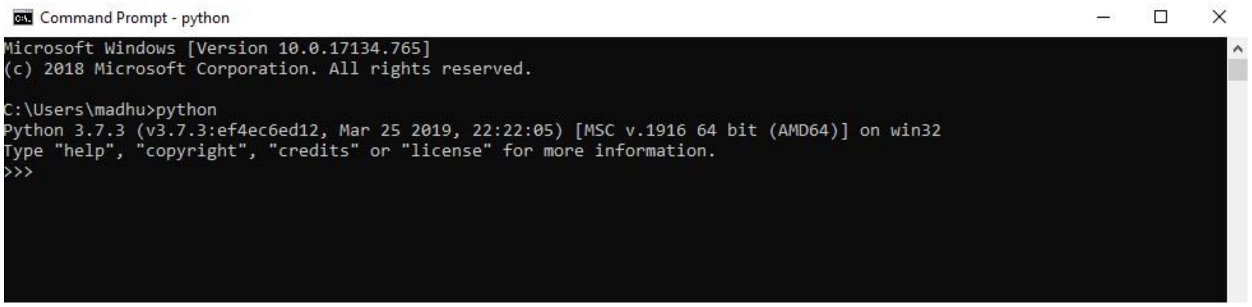
New to Python? Start with the online tutorial and documentation.

See what's new in this release.

Disable path length limit

Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

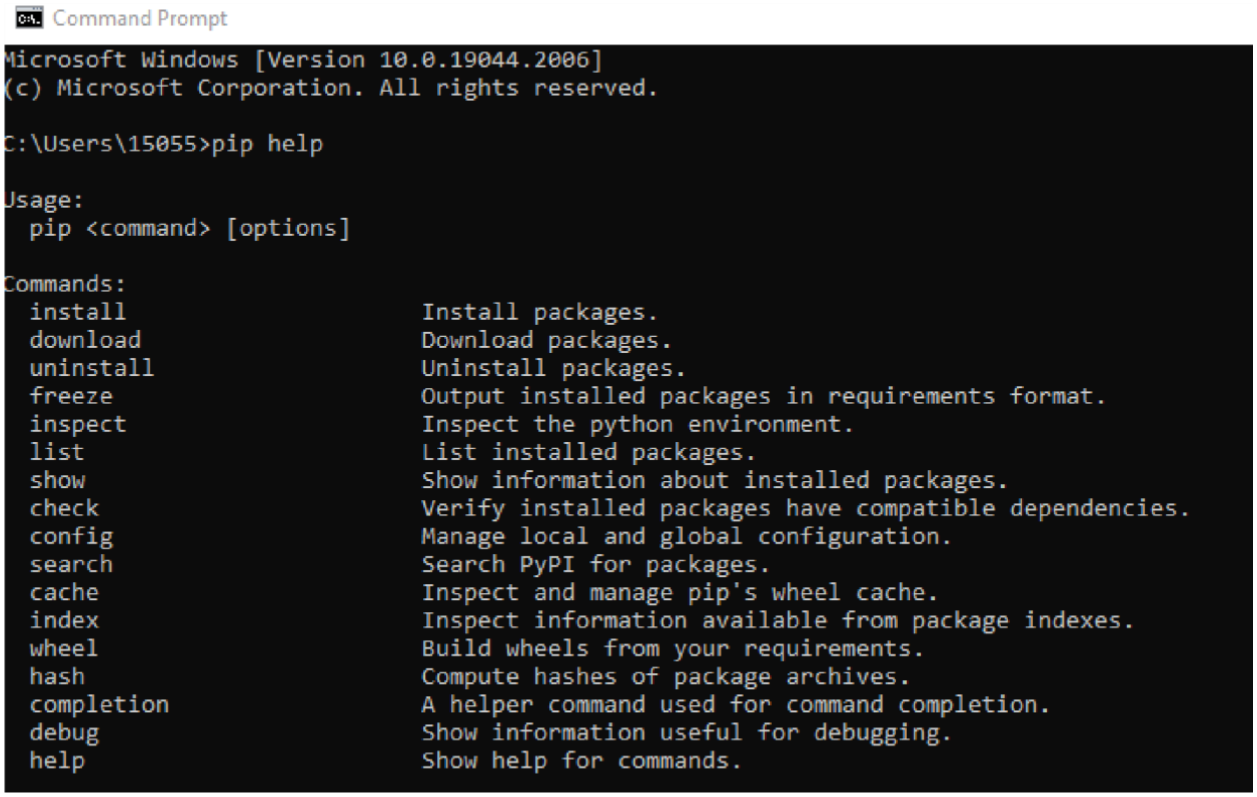
6. Test your Python installation by opening the command prompt (Windows+R -> type cmd -> click “OK”). Once your command prompt is open, type **“python.”** The result should be like the one below:



```
Microsoft Windows [Version 10.0.17134.765]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\madhu>python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type “quit()” and press enter to return to the command line from here.

7. Test your pip installation by typing **“pip help”** in the command line. The result should be like the one below:



```
Microsoft Windows [Version 10.0.19044.2006]
(c) Microsoft Corporation. All rights reserved.

C:\Users\15055>pip help

Usage:
  pip <command> [options]

Commands:
  install                     Install packages.
  download                    Download packages.
  uninstall                   Uninstall packages.
  freeze                      Output installed packages in requirements format.
  inspect                     Inspect the python environment.
  list                        List installed packages.
  show                        Show information about installed packages.
  check                       Verify installed packages have compatible dependencies.
  config                      Manage local and global configuration.
  search                      Search PyPI for packages.
  cache                       Inspect and manage pip's wheel cache.
  index                       Inspect information available from package indexes.
  wheel                       Build wheels from your requirements.
  hash                        Compute hashes of package archives.
  completion                  A helper command used for command completion.
  debug                       Show information useful for debugging.
  help                        Show help for commands.
```

8. You now need to install Jupyter and chipwhisperer. To do so, type **“pip install jupyterlab”** in the command line. Once the Jupyter installation is complete, type **“pip install chipwhisperer”** in the command line.

   NOTE: Windows users with Python 3.14+ should run **“pip install ipykernel==7.0.1”** after these steps

B. Mac Machines

Note: The installation steps for chipwhisperer dependencies use brew. Since it is already required, these steps shall make use of brew. Start installation from the “Packages” header down.

1. Refer to the brew webpage https://brew.sh/ for installation steps, or use the following command.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. Follow the installation instructions for the dependencies linked here, beginning from “Packages”
   NOTE: Python may be installed like this or using the Windows steps above
3. If not done in the previous step, set up a python virtual environment with **“python -m venv my-env”**, and then source it with **“source my-env/bin/activate”**

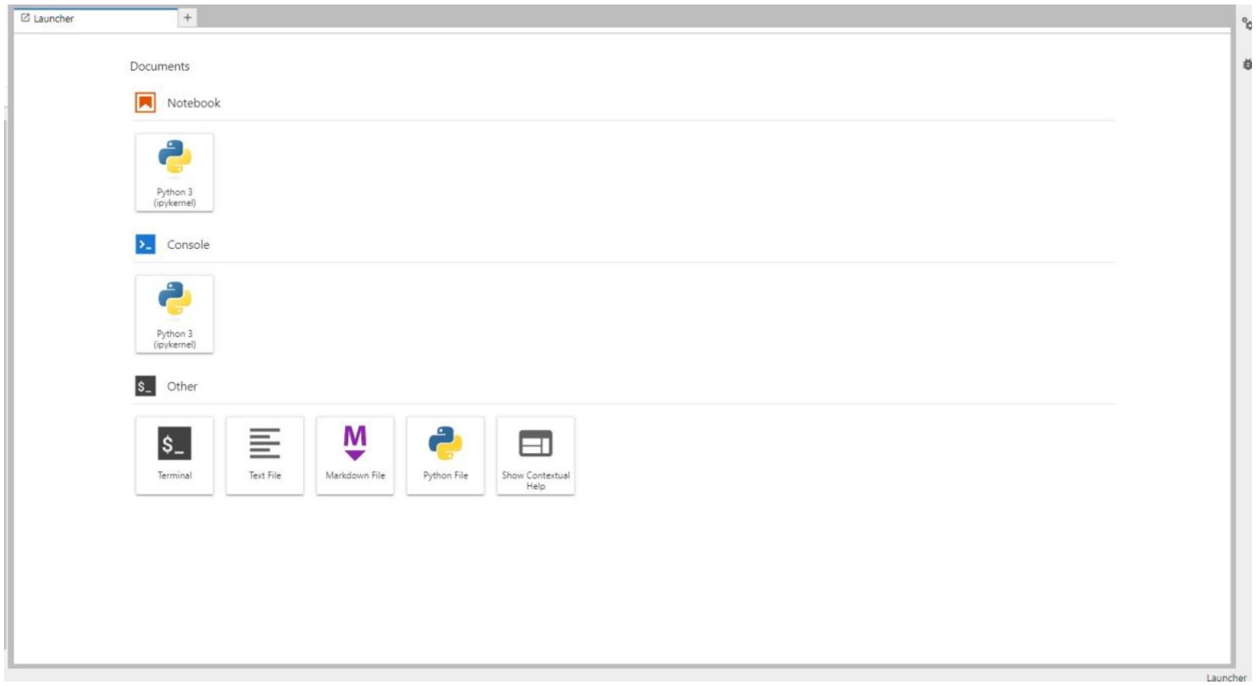
NOTE: you will need to source the venv every time you open a new terminal to run Jupyter

4. Install Jupyter and chipwhisperer with **“brew install jupyterlab”** and **“brew install chipwhisperer”** with the venv sourced (You will see (my-env) on the left side of the terminal)

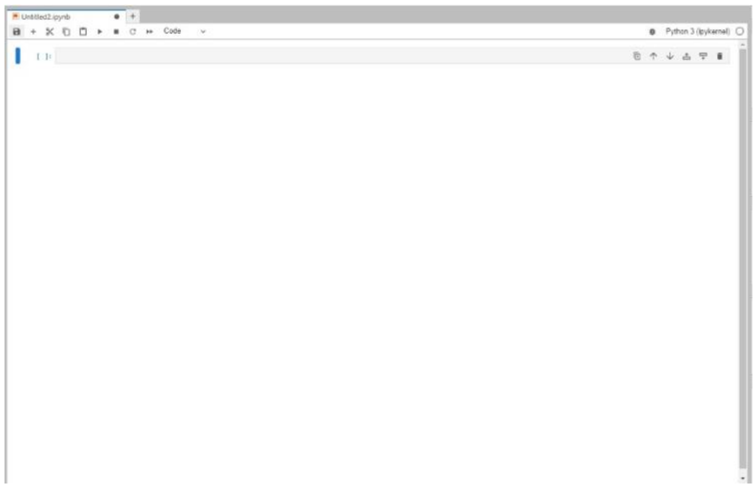
C. Linux Machines

Detailed instructions can be found here

9. Once Jupyter and chipwhisperer have been installed, launch JupyterLab with the command **“jupyter-lab”.** This will open a tab like the one below in your default web browser:



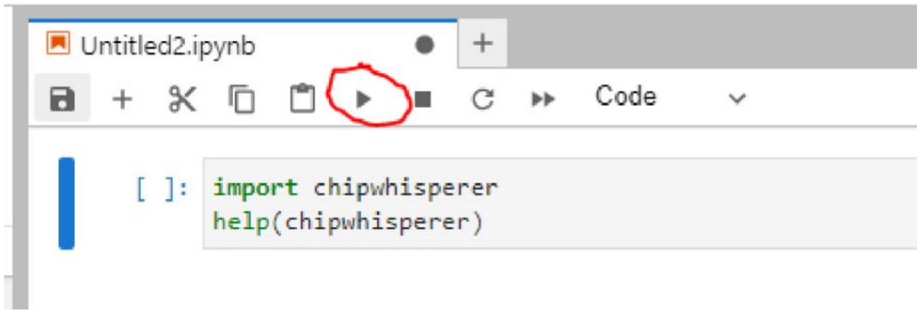
10. Create a new notebook by clicking the Python 3 button under the Notebook category in the screenshot above. Your notebook should look like below:



11. Paste the following Python code into the box with the vertical blue line to the left of it:

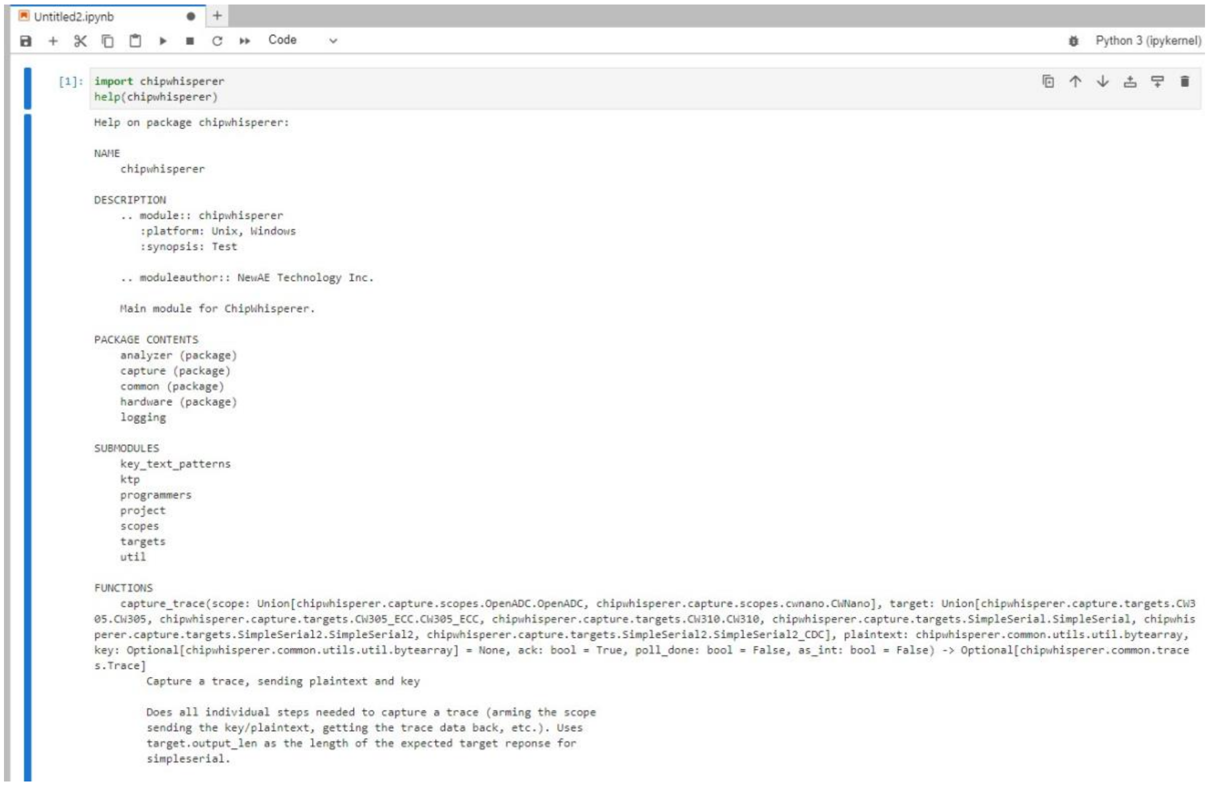
```
import chipwhisperer
help(chipwhisperer)
```

And press the Run button:



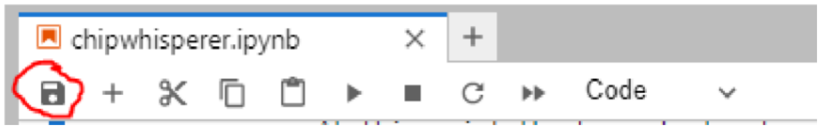
```
[ ]: import chipwhisperer
     help(chipwhisperer)
```

12. The code should compile and run successfully, with output like below:



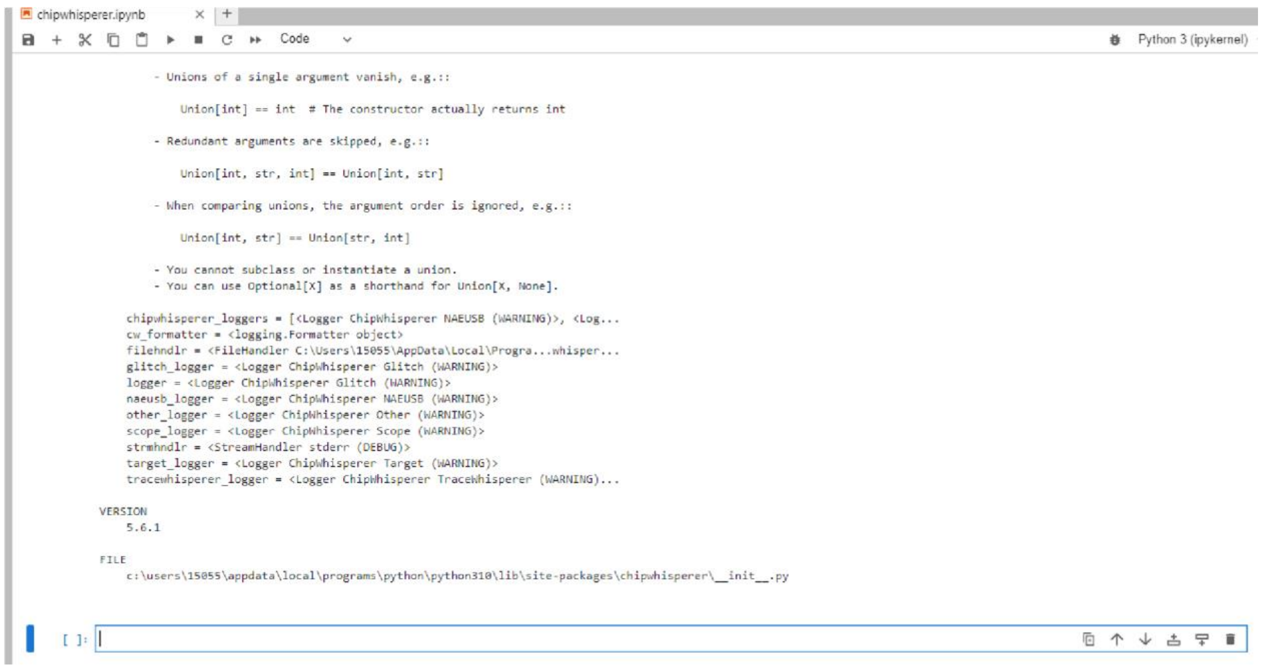
```
[1]: import chipwhisperer
     help(chipwhisperer)

Help on package chipwhisperer:

NAME
    chipwhisperer

DESCRIPTION
    .. module:: chipwhisperer
       :platform: Unix, Windows
       :synopsis: Test

    .. moduleauthor:: NewAE Technology Inc.

    Main module for ChipWhisperer.

PACKAGE CONTENTS
    analyzer (package)
    capture (package)
    common (package)
    hardware (package)
    logging

SUBMODULES
    key_text_patterns
    ktp
    programmers
    project
    scopes
    targets
    util

FUNCTIONS
    capture_trace(scope: Union[chipwhisperer.capture.scopes.OpenADC.OpenADC, chipwhisperer.capture.scopes.cwnano.CWNano], target: Union[chipwhisperer.capture.targets.CW3
05.CW305, chipwhisperer.capture.targets.CW305_ECC.CW305_ECC, chipwhisperer.capture.targets.CW310.CW310, chipwhisperer.capture.targets.SimpleSerial.SimpleSerial, chipwhis
perer.capture.targets.SimpleSerial2.SimpleSerial2, chipwhisperer.capture.targets.SimpleSerial2.SimpleSerial2_CDC], plaintext: chipwhisperer.common.utils.util.bytearray,
key: Optional[chipwhisperer.common.utils.util.bytearray] = None, ack: bool = True, poll_done: bool = False, as_int: bool = False) -> Optional[chipwhisperer.common.trace
s.Trace]
        Capture a trace, sending plaintext and key

        Does all individual steps needed to capture a trace (arming the scope
        sending the key/plaintext, getting the trace data back, etc.). Uses
        target.output_len as the length of the expected target reponse for
        simpleserial.
```

13. You can save your notebook file using the Save and create checkpoint button. You can rename your file by right-clicking on it in the files menu to the left and selecting rename.



How Jupyter Works

Jupyter works with **cells**. The code you just ran was a cell. Once you successfully execute the contents of a cell, you move on to a new cell:



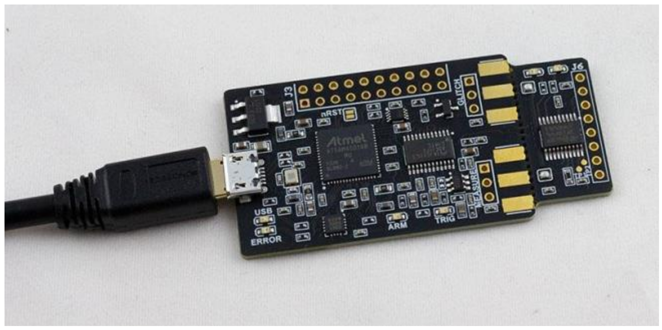
```
            - Unions of a single argument vanish, e.g.::

                Union[int] == int  # The constructor actually returns int

            - Redundant arguments are skipped, e.g.::

                Union[int, str, int] == Union[int, str]

            - When comparing unions, the argument order is ignored, e.g.::

                Union[int, str] == Union[str, int]

            - You cannot subclass or instantiate a union.
            - You can use Optional[X] as a shorthand for Union[X, None].

        chipwhisperer_loggers = [<Logger ChipWhisperer NAEUSB (WARNING)>, <Log...
        cw_formatter = <logging.Formatter object>
        filehndlr = <FileHandler C:\Users\15055\AppData\Local\Progra...whisper...
        glitch_logger = <Logger ChipWhisperer Glitch (WARNING)>
        logger = <Logger ChipWhisperer Glitch (WARNING)>
        naeusb_logger = <Logger ChipWhisperer NAEUSB (WARNING)>
        other_logger = <Logger ChipWhisperer Other (WARNING)>
        scope_logger = <Logger ChipWhisperer Scope (WARNING)>
        strmhndlr = <StreamHandler stderr (DEBUG)>
        target_logger = <Logger ChipWhisperer Target (WARNING)>
        tracewhisperer_logger = <Logger ChipWhisperer TraceWhisperer (WARNING)...

    VERSION
        5.6.1

    FILE
        c:\users\15055\appdata\local\programs\python\python310\lib\site-packages\chipwhisperer\__init__.py

[ ]:
```

A cell is prefixed with [ ]:, to the left of which is a box where you can enter the code for the cell. **Once you move on to a new cell, you can use the imports introduced and variables initialized in previous cells.** Bear this in mind as you complete the rest of this lab.

Connecting to the ChipWhisperer

Next, you will connect to the ChipWhisperer with Jupyter.

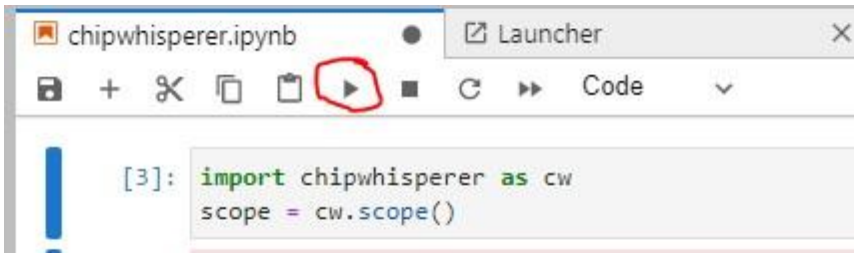
1. Connect the board to your device using the provided USB cable:



2. Start Jupyter. Create a new Python 3 notebook.

3. Insert the following Python code into a new cell:
```
import chipwhisperer as cw
scope = cw.scope()
```

4. Run the code in the cell:



```
[3]: import chipwhisperer as cw
     scope = cw.scope()
```

5. Connect to the target device:
```
target = cw.target(scope, cw.targets.SimpleSerial)
#cw.targets.SimpleSerial can be omitted
```

6. And set the board's default settings: `scope.default_setup()`

7. The cells above should run with no errors. Now, you will upload and build some firmware to the ChipWhisperer. First, you will need to download the ChipWhisperer compiler setup.
(Note: that is a direct link to the windows exe installer). For other platforms/versions you can look at their release page: https://github.com/newaetech/chipwhisperer/releases/).
This installation will take a while to complete **(unless you're in the Klaus lab, in which case, you can skip this step).**

8. You are now going to test building firmware from Jupyter. To do so, you need a make command. If you are on a Windows computer, you will likely not have a make command; however, you can install a package manager like Chocolatey. Follow this 2-step installation process, and you can now use the make command in Jupyter. (Windows)

9. Now run the following code in your notebook. Be sure to replace YOURUSER with the correct username for your system.

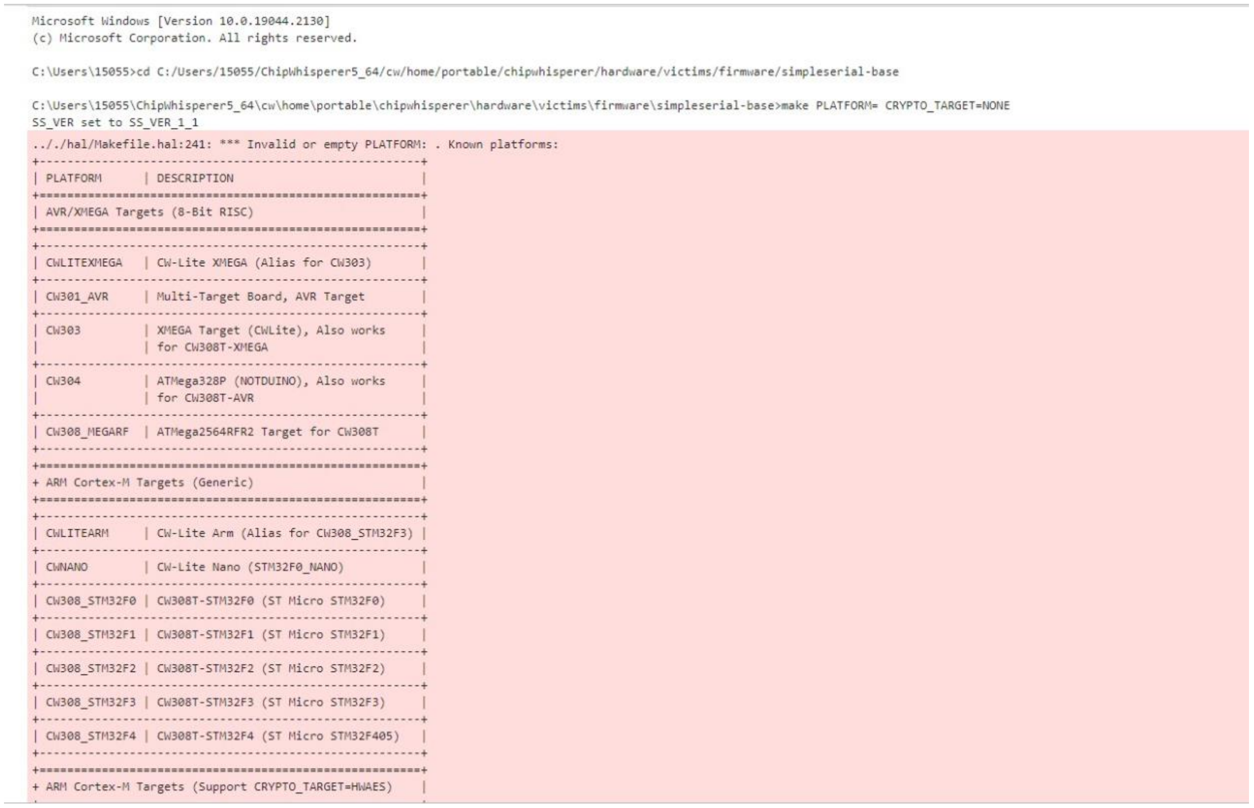
```
%%cmd              -- this line is only needed on windows
cd
C:/users/YOURUSER/ChipWhisperer5_64/cw/home/portable/chipwhisperer/hard
ware/victims/firmware/simpleserial-base/ make PLATFORM=
CRYPTO_TARGET=NONE
```

   If code errors and indicates that the directory is not found, you may need to manually locate the `simpleserial-base/` on your machine.

   Mac users may need to do the following: run **"brew upgrade make"**, or **"brew install make"** if the make command errors. Then, paste the following into your cell:

```
echo 'export PATH="/opt/homebrew/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc
cd
/Users/YOURUSER/chipwhisperer/firmware/mcu/simpleserial-
base/
make PLATFORM= CRYPTO_TARGET=NONE
```

   The output should be a list of supported platforms:



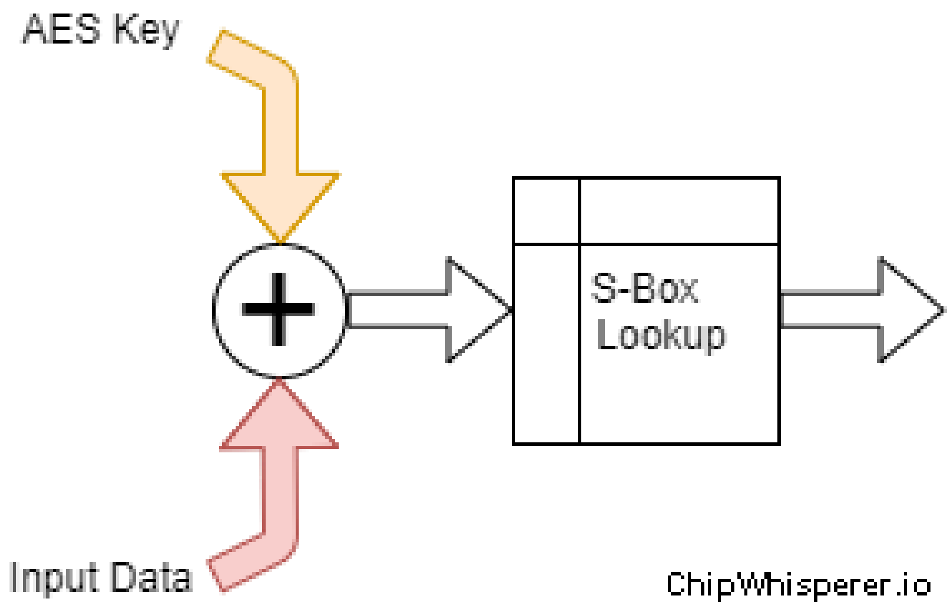
```
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

C:\Users\15055>cd C:/Users/15055/ChipWhisperer5_64/cw/home/portable/chipwhisperer/hardware/victims/firmware/simpleserial-base

C:\Users\15055\ChipWhisperer5_64\cw\home\portable\chipwhisperer\hardware\victims\firmware\simpleserial-base>make PLATFORM= CRYPTO_TARGET=NONE
SS_VER set to SS_VER_1_1
../hal/Makefile.hal:241: *** Invalid or empty PLATFORM: . Known platforms:
+-------------------------------------------------------+
| PLATFORM      | DESCRIPTION                           |
+=======================================================+
| AVR/XMEGA Targets (8-Bit RISC)                        |
+=======================================================+
+-------------------------------------------------------+
| CWLITEXMEGA   | CW-Lite XMEGA (Alias for CW303)       |
+-------------------------------------------------------+
| CW301_AVR     | Multi-Target Board, AVR Target        |
+-------------------------------------------------------+
| CW303         | XMEGA Target (CWLite), Also works     |
|               | for CW308T-XMEGA                      |
+-------------------------------------------------------+
| CW304         | ATMega328P (NOTDUINO), Also works     |
|               | for CW308T-AVR                        |
+-------------------------------------------------------+
| CW308_MEGARF  | ATMega2564RFR2 Target for CW308T      |
+-------------------------------------------------------+
+=======================================================+
+ ARM Cortex-M Targets (Generic)                        |
+=======================================================+
+-------------------------------------------------------+
| CWLITEARM     | CW-Lite Arm (Alias for CW308_STM32F3) |
+-------------------------------------------------------+
| CWNANO        | CW-Lite Nano (STM32F0_NANO)           |
+-------------------------------------------------------+
| CW308_STM32F0 | CW308T-STM32F0 (ST Micro STM32F0)     |
+-------------------------------------------------------+
| CW308_STM32F1 | CW308T-STM32F1 (ST Micro STM32F1)     |
+-------------------------------------------------------+
| CW308_STM32F2 | CW308T-STM32F2 (ST Micro STM32F2)     |
+-------------------------------------------------------+
| CW308_STM32F3 | CW308T-STM32F3 (ST Micro STM32F3)     |
+-------------------------------------------------------+
| CW308_STM32F4 | CW308T-STM32F4 (ST Micro STM32F405)   |
+-------------------------------------------------------+
+=======================================================+
+ ARM Cortex-M Targets (Support CRYPTO_TARGET=HWAES)    |
```

This concludes the section on the ChipWhisperer hardware setup.

Recovering an AES Key from Internal State

As the final task for this lab, you will recover an AES key from a single leaked bit of the internal state. This part of the lab does not work with the board, but serves to make you comfortable working with Jupyter notebooks as the next lab will use both Jupyter and the ChipWhisperer board.

Recall that AES data flow resembles the following:



The input data is XOR'd with a key byte and then passed through the S-box. Proceed to the next page to begin this part of the lab.

1. Start Jupyter ("jupyter-lab" in command line) and create a new Python 3 notebook. Define the following S-box in your notebook:

```
sbox = [
    # 0   1   2   3   4   5   6   7   8   9   a   b   c   d   e   f
0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,  # 0
0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,  # 1
0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,  # 2
0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,  # 3
0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,  # 4
0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,  # 5
0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,  # 6
0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,  # 7
0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,  # 8
0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,  # 9
0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,  # a
0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,  # b
0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,  # c
0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,  # d
0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,  # e
0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16   # f
]
```

2. Now, moving on to a new cell, modify the following Python code block to implement the correct function for AES. Remember, you want the function to XOR input data and key and then look up the result in the S-box from the previous step.

```
def aes_internal(inputdata, key):
    return ???
```

3. Run the following test cases in a new cell. If the values are not as expected, check your implementation. Otherwise, continue to the next steps.

```
#Simple test vectors - if you get the check-mark printed all OK.
assert(aes_internal(0xAB, 0xEF) == 0x1B)
assert(aes_internal(0x22, 0x01) == 0x26)
print("✔ OK to continue!")
```

4. In your next cell, you will define a function which uses a secret key to not expose the AES key.

```
def aes_secret(inputdata):
    secret_key = 0xEF
    return aes_internal(secret_key, inputdata)
```

5. At this point, you can get a value of some internal part of AES. For the attack demonstrated in this lab, you will observe a single bit of the value.

   Your objective is to build a list of data. This should be **a 1000-item list of random numbers in the interval [0, 255].** The following Python import may help you:

```
import random
```

   Also, list comprehension can be used to populate a list:

```
input_data = [somefunction(args) for _ in range(lower, upper)]
```

   Or you can populate your list using a loop. Either works.

   Now run the following test cases in a new cell and make sure the check-mark prints:

```
#Simple test vectors - if you get the check-mark printed all OK.
assert(len(input_data) == 1000)
#Next two can fail for random variables (re-run if you get an error)
assert(max(input_data) == 0xFF)
assert(min(input_data) == 0x00)
print("✔ OK to continue!")
```

6. The definition of leaked data is as follows:

```
leaked_data = [(aes_secret(a) & 0x01) for a in input_data]
```

   You will now attack AES using this definition of leaked data!

7. Build a function to count the number of elements in a list that match between two lists (same value at the same index). One option is to iterate through the number of elements in the list and count the number that are the same. Fill in the following:

```
def num_same(a, b):

    if len(a) != len(b):
        raise ValueError("Arrays must be same length!")

    if max(a) != max(b):
        raise ValueError("Arrays max() should be the same!")

    # Count how many list items match up
    ???

    return same
```

Run the following test case and proceed if you get the check-mark:

```
#Simple test vectors - if you get the check-mark printed all OK.
assert(num_same([0,1,0,1,1,1,1,0], [0,1,0,1,1,1,1,0]) == 8)
assert(num_same([1,1,1,0,0,0,0,0], [0,1,0,1,1,1,1,0]) == 2)
assert(num_same([1, 0], [0, 1]) == 0)
print("✔ OK to continue!")
```

8. The next block is the most important. You'll apply the leakage function. For each known byte, pass it through aes_internal(). The value of key_guess is integers in `[0x00, 0x01,...,0xFF]`

   You'll compare the number of matching bits in the leaked data bit and the hypothetical data bit.

```
for guess in range(0, 256):
    #Get a hypothetical leakage list - use aes_internal(guess, input_byte) and
mask off to only get value of lowest bit.
    #You'll need to make this into a list as well.

    hypothetical_leakage = [aes_internal(guess, input_byte) & 0x01 for
input_byte in input_data]

    #Use our function
    same_count = num_same(hypothetical_leakage, leaked_data)

    #Print for debug
    print("Guess {:02X}: {:4d} bits same".format(guess, same_count))
```

9. A good thing to do will be to sort by number of correct bits. This can be done efficiently with numpy.argsort, which returns the indices that would sort the list:

```
import numpy as np

guess_list = [0] * 256

for guess in range(0, 256):

    #Get a hypothetical leakage list - use aes_internal(guess, input_byte) and
mask off to only get value of lowest bit
    hypothetical_leakage = [aes_internal(guess, input_byte) & 0x01 for
input_byte in input_data]

    #Use our function
    same_count = num_same(hypothetical_leakage, leaked_data)

    #Track the number of correct bits
    guess_list[guess] = same_count

#Use np.argsort to generate a list of indices from low to high, then [::-1] to
reverse the list to get high to low.
sorted_list = np.argsort(guess_list)[::-1]

#Print top 5 only
for guess in sorted_list[0:5]:
    print("Key Guess {:02X} = {:04d} matches".format(guess,guess_list[guess]))
```

10. In this case, you know bit 0 was the leaked bit, but what if you did not know that? First, define a function that returns the value of a bit being 1 or 0:

```
def get_bit(data, bit):
    if data & (1 << bit):
        return 1
    else:
        return 0
```

Now make a slightly more advanced leakage function using get_bit():

```
def aes_leakage_guess(keyguess, inputdata, bit):
        return get_bit(aes_internal(keyguess, inputdata), bit)
```

11. Lastly, you will write a loop using the leakage function. The result of this loop will be 5 AES key guesses deduced from bit guesses in the range 0-8.

```
for bit_guess in range(0, 8):
    guess_list = [0] * 256
    print("Checking bit {:d}".format(bit_guess))

    for guess in range(0, 256):

        #Get a hypothetical leakage for guessed bit (ensure returns 1/0 only)
        #Use bit_guess as the bit number, guess as the key guess, and data 
from input_data
        hypothetical_leakage = [aes_leakage_guess(guess, input_byte, 
bit_guess) for input_byte in input_data]

        #Use our function
        same_count = num_same(hypothetical_leakage, leaked_data)

        #Track the number of correct bits
        guess_list[guess] = same_count

    sorted_list = np.argsort(guess_list)[::-1]

    #Print top 5 only
    for guess in sorted_list[0:5]:
        print("Key Guess {:02X} = {:04d} matches".format(guess, 
guess_list[guess]))
```

Required Documents to turn in

- Entire Jupyter notebook for the AES attack.
- List of supported platforms for ChipWhisperer, obtained in step 9 of the chipwhisperer hardware connection section of this lab.
- Top 5 AES key guesses, obtained in step 11 of the AES attack section of this lab.