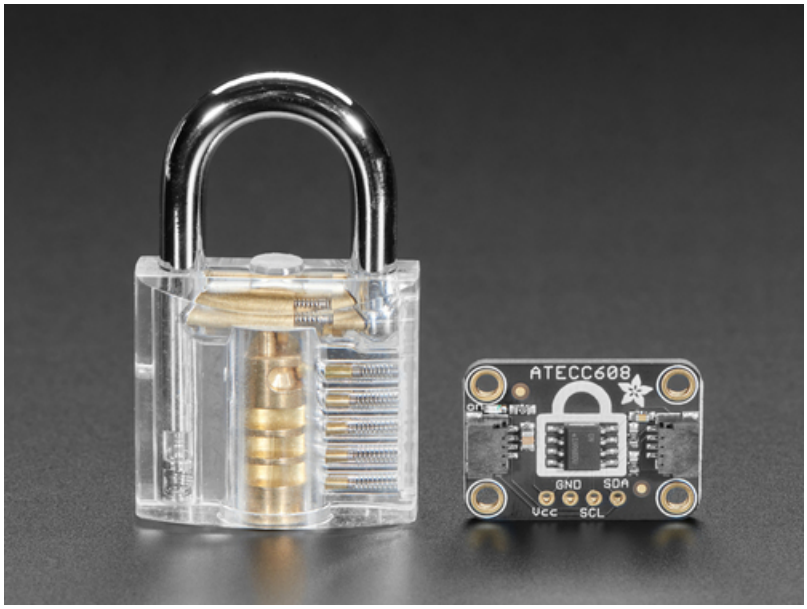
adafruit learning system

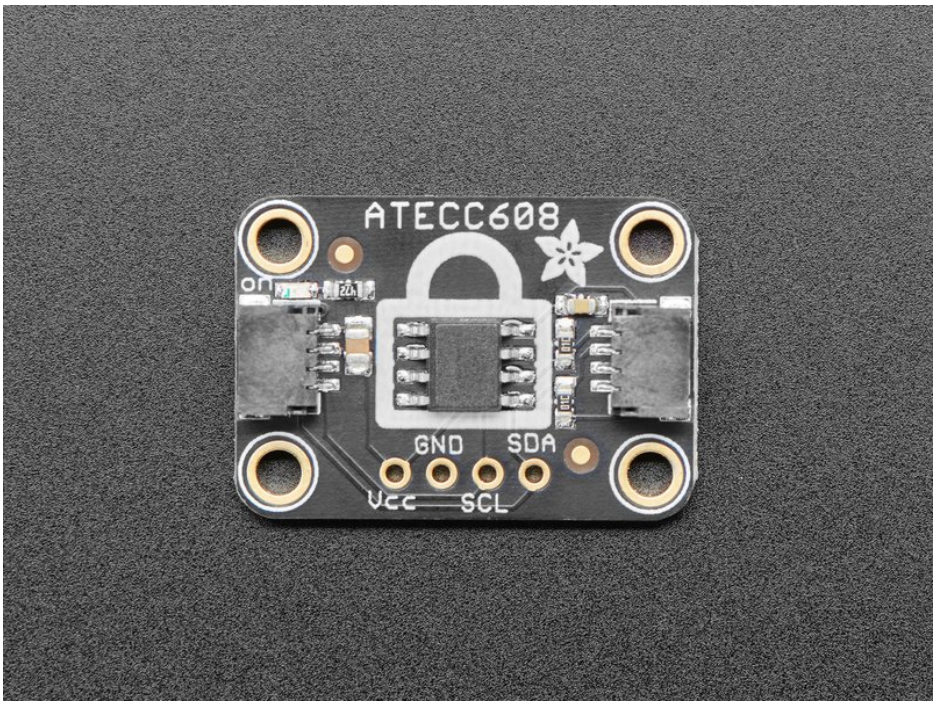
Adafruit ATECC608 Breakout

Created by Kattni Rembor

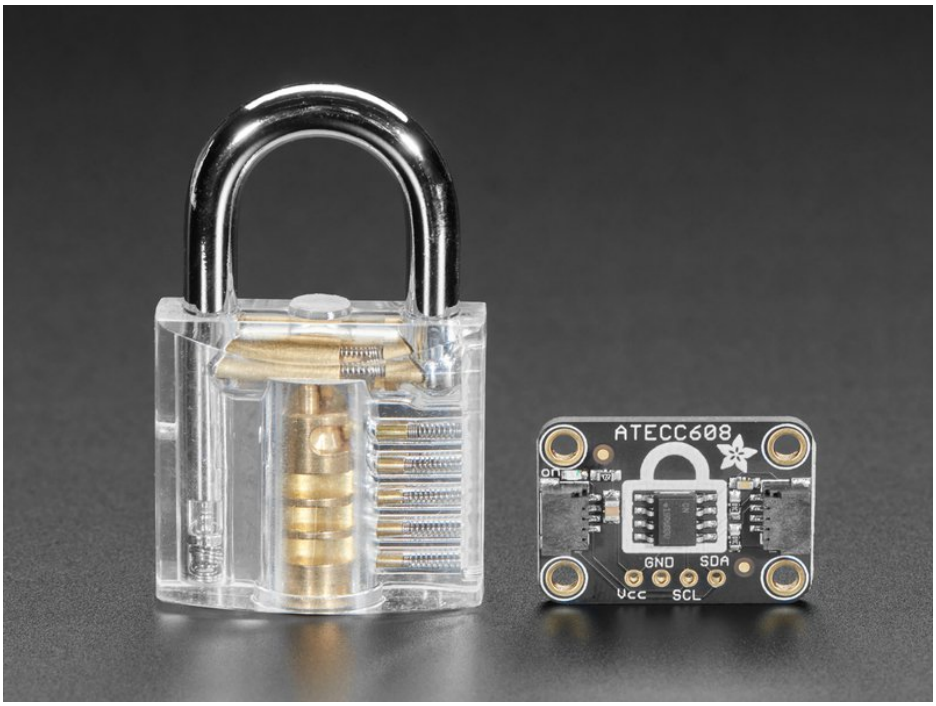


Last updated on 2019-11-04 07:24:19 PM UTC

Overview

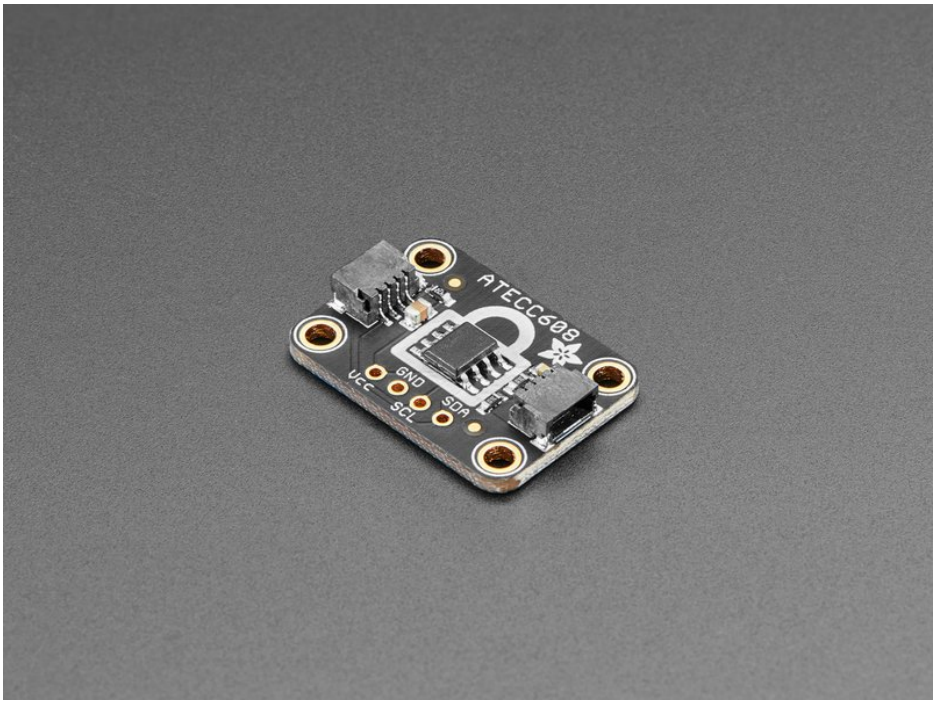


You've got secrets, and you want to keep them safe? Most microcontrollers are not designed to protect against snoopers, but a crypto-authentication chip can be used to lock away private keys securely. Once the private key is saved inside, it can't be read out, all you can do is send it challenge-response queries. That means that even if someone gets hold of your hardware and can read back the firmware, they won't be able to extract the secret!

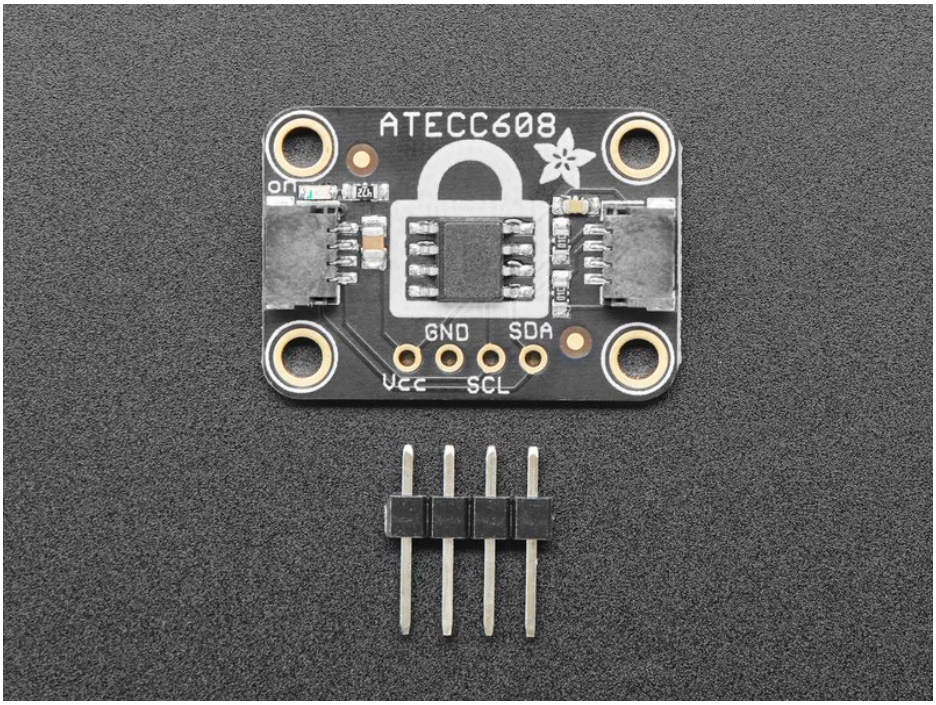


The ATECC608 is the latest crypto-auth chip from Microchip, and it uses I2C to send/receive commands. Once you 'lock' the chip with your details, you can use it for ECDH and AES-128 encrypt/decrypt/signing. There's also hardware

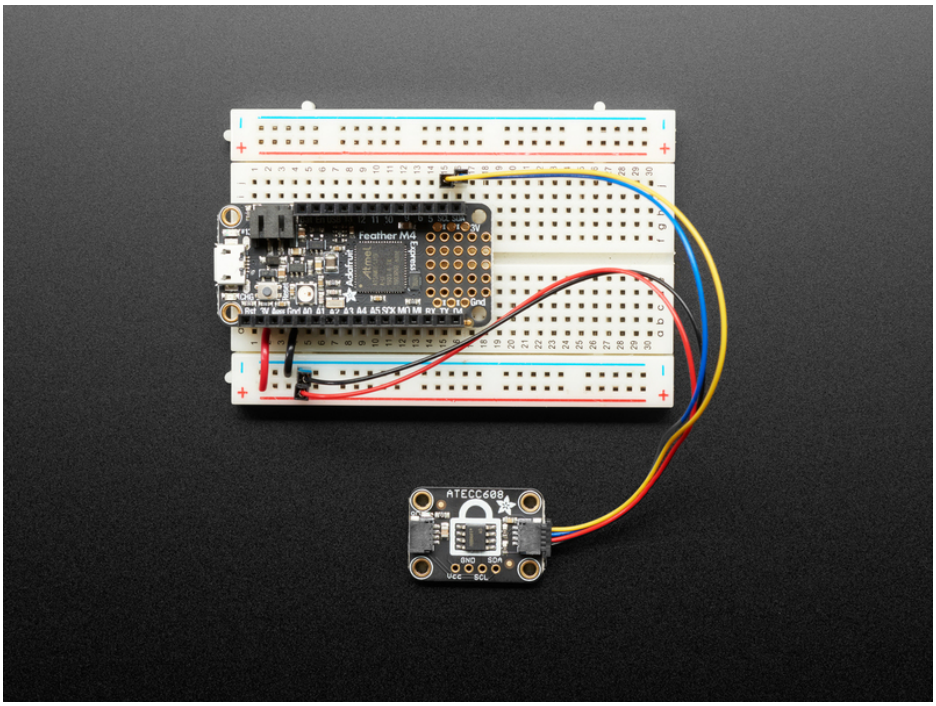
support for random number generation, and SHA-256/HMAC hash functions to greatly speed up a slower micro's cryptography commands.



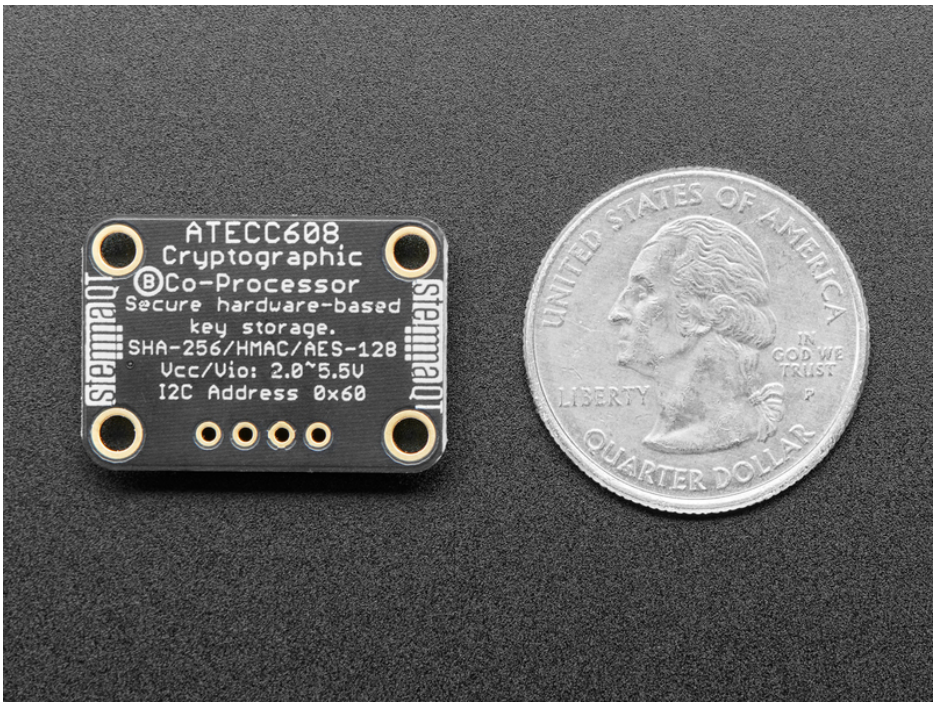
We're starting to see these low-cost secure element chips in various products, so that a less expensive chip can be used to drive peripherals, without worrying about security. This chip does not have a public datasheet, but it is compatible with the ATECC508 earlier version which does, so please refer to that complete datasheet (https://adafru.it/Flg) as well as the ATECC608 summary sheet (https://adafru.it/Flh). The good news is that, despite not having complete documentation, there *is* some software support. For Arduino use, check out the Arduino ATECCx08 library (https://adafru.it/Fli). For Python and C/C++ check out Microchips Cryptoauthlib (https://adafru.it/Flj)(yes we also think it's odd that there's no datasheet but there is published code)



To make working with the ATECC608 as easy as possible, we've put it on a breakout PCB with the required support circuitry and SparkFun qwiic (https://adafru.it/Fpw) compatible **STEMMA QT** (https://adafru.it/Ft4) connectors. This allows you to use it with other similarly equipped boards **without needing to solder.** This chip will work with 3.3V or 5V power/logic micros, so it's ready to get to work with a range of development boards.

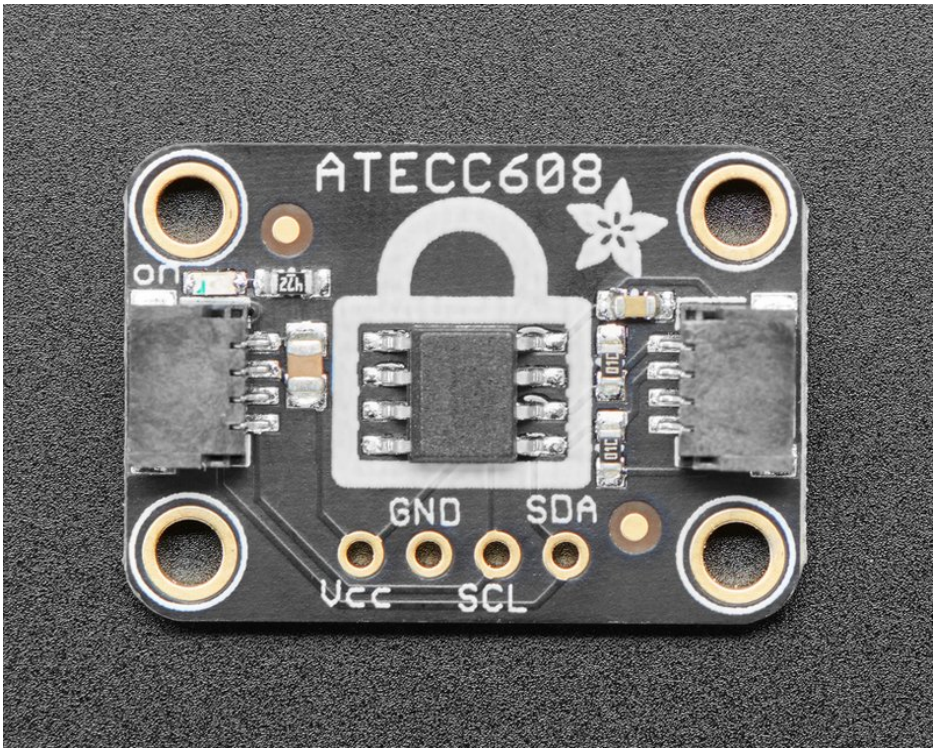


Please note the I2C address is fixed at 0x60 and according to Microchip, you should use this at higher I2C speeds like 400KHz if other devices are on the I2C bus, to avoid some I2C bus contention (much like the datasheet, this is not documented anywhere yet).



Clear Padlock (https://adafru.it/FIk) and STEMMA QT cable (https://adafru.it/FA-) not included (but we have them in the shop!)

Pinouts



Power Pins

- **Vcc** - this is the power pin. You can use 3.3V or 5V. Be sure to use the same power level as the logic level on your board - e.g. for a 5V microcontroller like Arduino, use 5V.
- **GND** - common ground for power and logic.

I2C Logic Pins

- **SCL** - this is the I2C clock pin, connect to your microcontroller I2C clock line.
- **SDA** - this is the I2C data pin, connect to your microcontroller I2C data line.
- **STEMMA QT (https://adafru.it/Ft4) -** These connectors allow you to connectors to dev boards with **STEMMA QT** connectors or to other things with various associated accessories (https://adafru.it/Ft6).

Please note the I2C address is fixed at 0x60 and according to Microchip, you should use this at higher I2C speeds like 400KHz if other devices are on the I2C bus, to avoid some I2C bus contention (much like the datasheet, this is not documented anywhere yet).

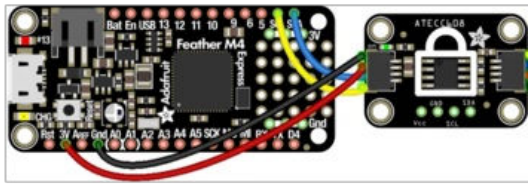
Python & CircuitPython

It's easy to use the ATECC608 with CircuitPython and the Adafruit CircuitPython ATECC (https://adafru.it/FPR) module. This module allows you to easily write Python code that can communicate with the ATECC608 module.

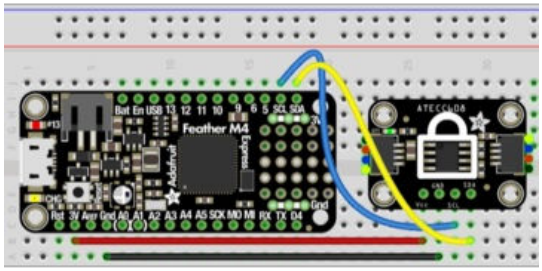
You can use this sensor with any CircuitPython microcontroller board or with a Linux single board computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library (https://adafru.it/BSN).

CircuitPython Microcontroller Wiring

Wiring the ATECC608 is easy, since it only requires power and two wires for an I2C connection. Additionally, the STEMMA QT connectors give you additional solderless options for wiring:



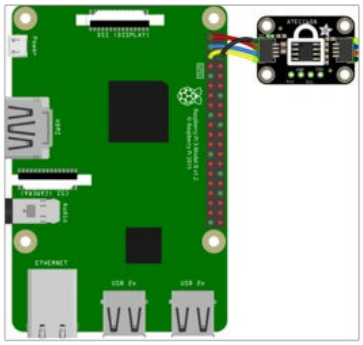
- **Board 3V** to **sensor VIN (red wire)**
- **Board GND** to **sensor GND (black wire)**
- **Board SCL** to **sensor SCL (yellow wire)**
- **Board SDA** to **sensor SDA (blue wire)**

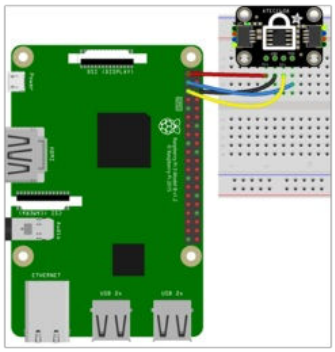


Python Computer Wiring

Since there's *dozens* of Linux computers/boards you can use we will show wiring for Raspberry Pi (https://adafru.it/scY). For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported (https://adafru.it/BSN).

Here's the Raspberry Pi wired with I2C:





- **Pi 3V** to **sensor VIN (red wire)**
- **Pi GND** to **sensor GND (black wire)**
- **Pi SCL** to **sensor SCL (yellow wire)**
- **Pi SDA** to **sensor SDA (blue wire)**

CircuitPython Installation of the ATECC Library

You'll need to install the Adafruit CircuitPython ATECC (https://adafru.it/FPR) library on your CircuitPython board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/uap). Our CircuitPython starter guide has a great page on how to install the library bundle (https://adafru.it/ABU).

For non-express boards like the Trinket M0 or Gemma M0, you'll need to manually install the necessary libraries from the bundle:

- **adafruit_atecc.mpy**
- **adafruit_atecc_asn1.mpy**
- **adafruit_atecc_cert_util.mpy**
- **adafruit_binascii**
- **adafruit_bus_device**

Before continuing make sure your board's **lib** folder or root filesystem has the **adafruit_atecc.mpy, adafruit_atecc_asn1.mpy, adafruit_atecc_cert_util.mpy, adafruit_binascii** and **adafruit_bus_device** files and folders copied over.

Next connect to the board's serial REPL (https://adafru.it/Awz)so you are at the CircuitPython `>>>` prompt.

Python Installation of ATECC Library

You'll need to install the **Adafruit_Blinka** library that provides the CircuitPython support in Python. This may also require enabling I2C on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready (https://adafru.it/BSN)!

Once that's done, from your command line run the following command:

- `sudo pip3 install adafruit-circuitpython-atecc`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

CircuitPython and Python Usage

To demonstrate the power of the ATECC co-processor, you'll initialize it, generate a random number, print (and increase) one of the co-processor's internal counters, and perform a SHA-256 hash operation.

Copy the following code to the `code.py` file on your CircuitPython device:

```
import board
import busio
from adafruit_atecc.adafruit_atecc import ATECC, _WAKE_CLK_FREQ

# Initialize the i2c bus
i2c = busio.I2C(board.SCL, board.SDA,
                frequency=_WAKE_CLK_FREQ)

# Initialize a new atecc object
atecc = ATECC(i2c)

print("ATECC Serial: ", atecc.serial_number)

# Generate a random number with a maximum value of 1024
print("Random Value: ", atecc.random(rnd_max=1024))

# Print out the value from one of the ATECC's counters
# You should see this counter increase on every time the code.py runs.
print("ATECC Counter #1 Value: ", atecc.counter(1, increment_counter=True))

# Initialize the SHA256 calculation engine
atecc.sha_start()

# Append bytes to the SHA digest
print("Appending to the digest...")
atecc.sha_update(b"Nobody inspects")
print("Appending to the digest...")
atecc.sha_update(b" the spammish repetition")

# Return the digest of the data passed to sha_update
message = atecc.sha_digest()
print("SHA Digest: ", message)
```

Save the file and open the board's REPL. You should see the following output:

```
code.py output:
ATECC Serial:  012347A22713EAFCEE
Random Value:  619
ATECC Counter #1 Value:  bytearray(b'!\x00\x00\x00')
Appending to the digest...
Appending to the digest...
SHA Digest:  bytearray(b'\x03\x1e\xdd}Ae\x15\x93\xc5\xfe\\\x00o\xa5u+7\xfd\xdf\xf7\xbcN\x84:
\xa6\xaf\x0c\x95\x0fK\x94\x06')
```

The ATECC will output its serial number and a random value to the REPL. You can modify the call to `atecc.random()` in your code to specify a minimum or maximum integer value to generate.

```
print("ATECC Serial: ", atecc.serial_number)

# Generate a random number with a maximum value of 1024
print("Random Value: ", atecc.random(rnd_max=1024))
```

The code also writes a value of one of the ATECC's hardware counters to the REPL. This code uses counter #1, there's two hardware counters built into the ATECC chip.

If you run the file again (by saving it), you'll notice the counter value increase. The counter's state is saved, even if the ATECC is powered off. If you're building an IoT project, you can save important state information in here, such as the last frame which was transmitted before the connection was lost.

```
print("ATECC Counter #1 Value: ", atecc.counter(1, increment_counter=True))
```

The ATECC is capable of performing hashing using the SHA256 algorithm. Instead of using a slow python-based SHA256 hash algorithm (such as the one found within CircuitPython's hashlib module (https://adafru.it/FPS)), you can use the ATECC's super-quick hardware to hash data for you!

CircuitPython_ATECC implements a CPython3 hashlib-like (https://adafru.it/FPT) interface. First, initialize the SHA256 calculate engine and memory context of the ATECC chip.

```
# Initialize the SHA256 calculation engine
atecc.sha_start()
```

Then, can append bytes to hash object by passing bytes to `sha_update`. You can append as many bytes as you'd like before you create a message digest.

```
# Append bytes to the SHA digest
print("Appending to the digest...")
atecc.sha_update(b"Nobody inspects")

print("Appending to the digest...")
atecc.sha_update(b" the spammish repetition")
```

The digest of the data passed to the `sha_update()` method is then returned and printed to the REPL.

```
# Return the digest of the data passed to sha_update
message = atecc.sha_digest()
print("SHA Digest: ", message)
```

Self-Signed Certificate Demo

This demo will generate a self signed certificate for a private key generated by your crypto chip.

Copy the code below to your `code.py` file.

```
import board
import busio
from adafruit_atecc.adafruit_atecc import ATECC, _WAKE_CLK_FREQ, CFG_TLS

import adafruit_atecc.adafruit_atecc_cert_util as cert_utils

# -- Enter your configuration below -- #

# Lock the ATECC module when the code is run?
LOCK_ATECC = False
# 2-letter country code
MY_COUNTRY = "US"
# State or Province Name
MY_STATE = "New York"
# City Name
MY_CITY = "New York"
# Organization Name
MY_ORG = "Adafruit"
# Organizational Unit Name
MY_SECTION = "Crypto"
# Which ATECC slot (0-4) to use
ATECC_SLOT = 0
# Generate new private key, or use existing key
GENERATE_PRIVATE_KEY = True

# -- END Configuration, code below -- #

# Initialize the i2c bus
i2c = busio.I2C(board.SCL, board.SDA,
                frequency=_WAKE_CLK_FREQ)

# Initialize a new atecc object
atecc = ATECC(i2c)

print("ATECC Serial Number: ", atecc.serial_number)

if not atecc.locked:
    if not LOCK_ATECC:
        raise RuntimeError("The ATECC is not locked, set LOCK_ATECC to True in code.py.")
    print("Writing default configuration to the device...")
    atecc.write_config(CFG_TLS)
    print("Wrote configuration, locking ATECC module...")
    # Lock ATECC config, data, and otp zones
    atecc.lock_all_zones()
    print("ATECC locked!")

print("Generating Certificate Signing Request...")
# Initialize a certificate signing request with provided info
csr = cert_utils.CSR(atecc, ATECC_SLOT, GENERATE_PRIVATE_KEY, MY_COUNTRY, MY_STATE,
                     MY_CITY, MY_ORG, MY_SECTION)
# Generate CSR
my_csr = csr.generate_csr()
print("-----BEGIN CERTIFICATE REQUEST-----\n")
print(my_csr.decode('utf-8'))
print("-----END CERTIFICATE REQUEST-----")
```

Save the file and **open the board's REPL**. You should see the following output:

```
ATECC Serial Number:  012347A22713EAFCEE
The ATECC is not locked, set LOCK_ATECC to True in code.py.
```

You'll want to **edit the following lines in** `code.py` before generating a certificate signing request.

```
# -- Enter your configuration below -- #

# Lock the ATECC module when the code is run?
LOCK_ATECC = False
# 2-letter country code
MY_COUNTRY = "US"
# State or Province Name
MY_STATE = "New York"
# City Name
MY_CITY = "New York"
# Organization Name
MY_ORG = "Adafruit"
# Organizational Unit Name
MY_SECTION = "Crypto"
# Which ATECC slot (0-4) to use
ATECC_SLOT = 0
# Generate new private key, or use existing key
GENERATE_PRIVATE_KEY = True

# -- END Configuration, code below -- #
```

Once the configuration in code.py looks correct, **change the following line from:**

`LOCK_ATECC = False`

to

`LOCK_ATECC = True`

This will **permanently** lock your chip. You will only need to set this once.

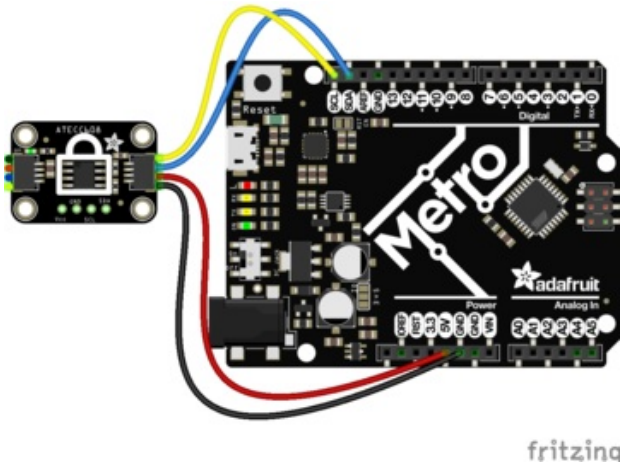
Save and run your code. You should see the following output:

```
ATECC Serial Number:  012347A22713EAFCEE
Writing default configuration to the device...
Wrote configuration, locking ATECC module...
ATECC locked!

Generating Certificate Signing Request...
-----BEGIN CERTIFICATE REQUEST-----

MIIBMDCB1gIBADB0MQswCQYDVQQGEwJVUzERMA8GA1UECBMITmV3IFlvcmsxETAPBgNVBAcTCE5ldyBZb3JrMREwDwYDVQQKEwhBZGFmc


-----END CERTIFICATE REQUEST-----
```

The ATECC608A will quickly generate a certificate signing request and output it to the REPL.

Arduino

Wiring



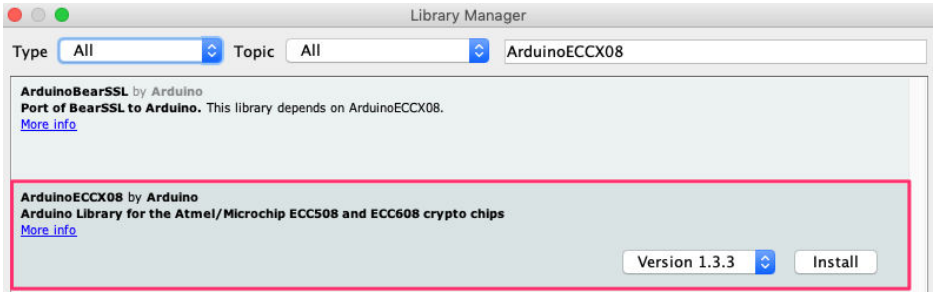
- Connect **Metro or Arduino 5V** to **board Vcc**
- Connect **Metro or Arduino GND** to **board GND**
- Connect **Metro or Arduino SDA** to **board SDA**
- Connect **Metro or Arduino SCL** to **board SCL**

Please note the I2C address is fixed at 0x60 and according to Microchip, you should use this at higher I2C speeds like 400KHz if other devices are on the I2C bus, to avoid some I2C bus contention (much like the datasheet, this is not documented anywhere yet).

Installation

Arduino created an ECCX08 library (https://adafru.it/FIi) which can generate a random number, generate a self-signed certificate, and generate a JSON Web Token (JWT).

Click the **Manage Libraries ...** menu item, search for **Arduino ECCX08**, and select the **ArduinoECCX08** library:



Random Number Demo

This demo demonstrates the ATECC608's ability to quickly generate a random number.

Open up **File -> Examples -> ECCX08 -> ECCX08RandomNumber** and upload to your Arduino wired up to the breakout.

Upload the sketch to your board and open up the Serial Monitor (**Tools -> Serial Monitor**) at 9600 baud. You should see a new randomly generated number printing to the serial monitor every second.

Now you know that the board is working!

Self-Signed Certificate Demo

This demo will generate a self signed certificate for a private key generated by your crypto chip.

Open up **File -> Examples -> ECCX08 -> Tools -> ECCX08SelfsignedCert** and upload to your Arduino wired up to the breakout.

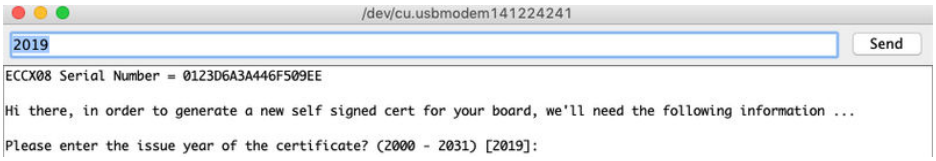
Upload the sketch to your board and open up the Serial Monitor (**Tools -> Serial Monitor**) at 9600 baud.

If the chip is not locked, you will be prompted to lock it.

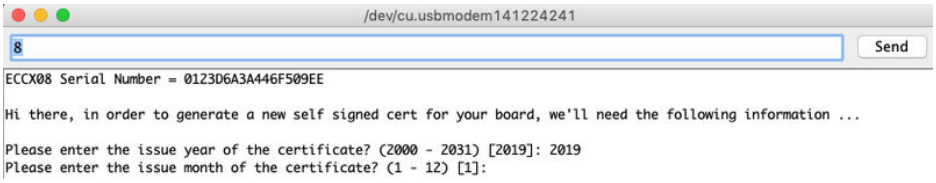
```
The ECCX08 on your board is not locked, would you like to PERMANENTLY configure and lock it now? (y/N)"
```

Type Y into the serial monitor and press the enter key. This will **permanently** lock your chip. You will only do this once.

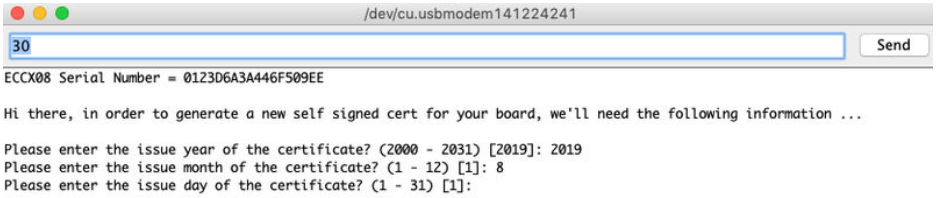
The code will ask you to **enter the issue year of the certificate into the serial monitor. Click send**.



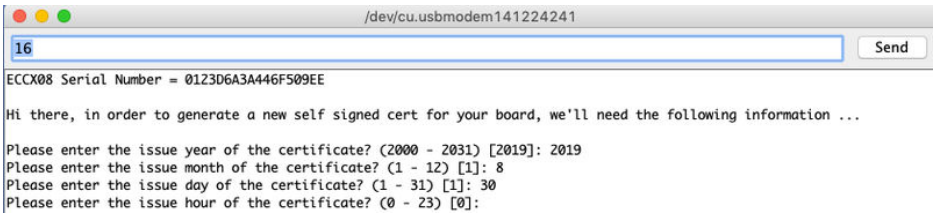
Enter the issue month of the certificate into the serial monitor. **Click send**.



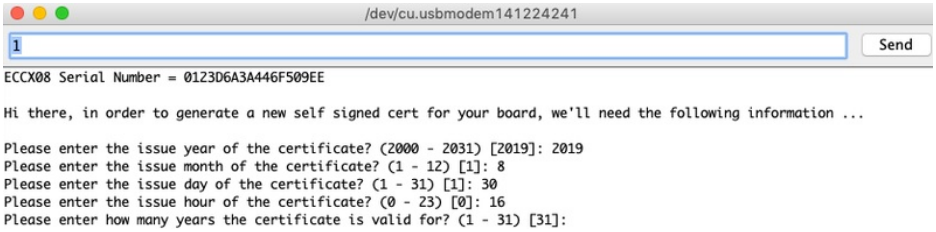
Enter the issue day of the certificate into the serial monitor. **Click send**.



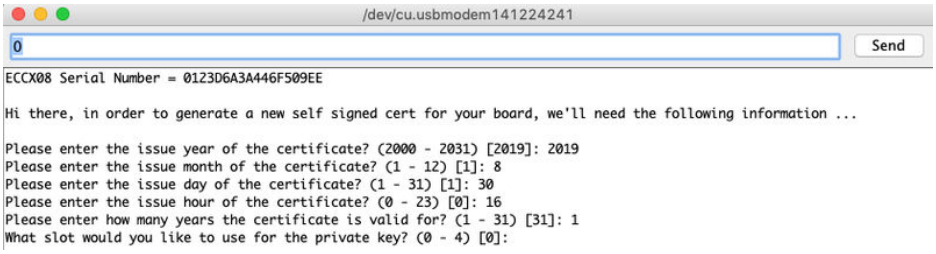
Enter the issue hour (the current hour, in *24* hour time) of the certificate into the serial monitor. **Click send**.



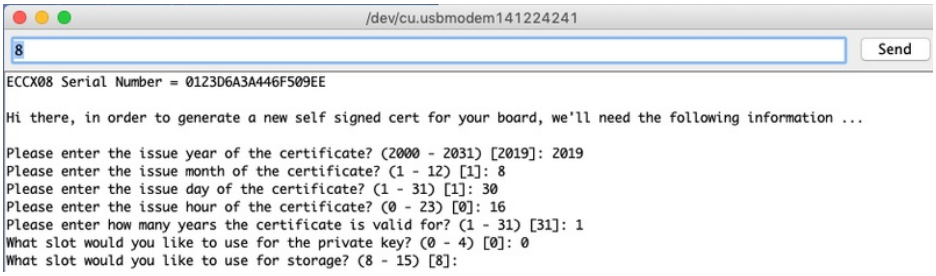
Certificates are only valid for a fixed period of time. **Enter how many years the certificate is valid** for, then **click send**.



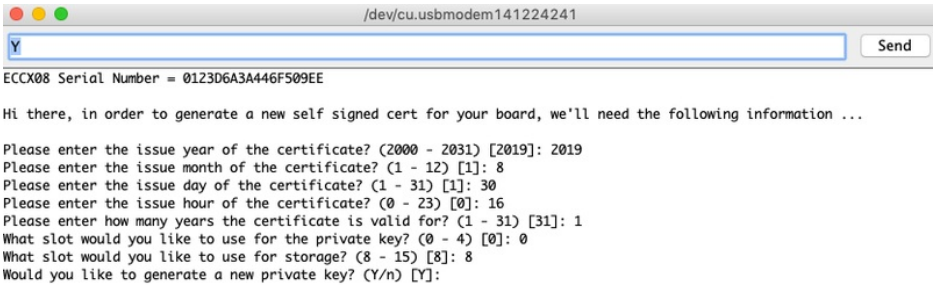
The ATECC608A can store up to 5 private keys. **Select the slot you'd like to use for the private key and click send**.



The issue date, expiration date, and signature are stored in a different slot on the ATECC608A. **Select a slot to use for storing these values**.



Enter *Y* into the serial monitor to generate a new private key. If you have a previous private key stored on the ATECC608A, **enter *n into the serial monitor* to use a key you previously generated** and stored in a slot.



The ATECC608A will quickly generate a self signed certificate and display it on the serial monitor. It'll also include a SHA1 hash for verifying the certificate.

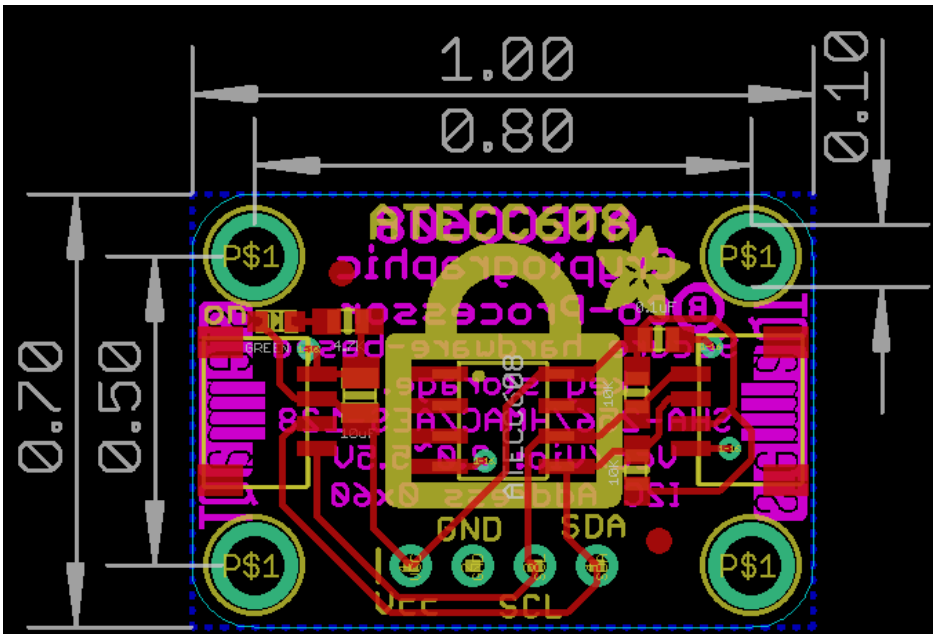
/dev/cu.usbmodem141224241
Send
Please enter the issue hour of the certificate? (0 - 23) [0]: 16
Please enter how many years the certificate is valid for? (1 - 31) [31]: 1
What slot would you like to use for the private key? (0 - 4) [0]: 0
What slot would you like to use for storage? (8 - 15) [8]: 8
Would you like to generate a new private key? (Y/n) [Y]: Y
Here's your self signed cert, enjoy!
-----BEGIN CERTIFICATE-----
MIIBKTCB0aADAgECAgEBMAoGCCqGSM49BAMCMB0xGzAZBgNVBAMTEjAxMjNENkEzQTQ0NkY1MDlF
RTAeFw0xOTA4MzAxNjAwMDBaFw0yMDA4MzAxNjAwMDBaMB0xGzAZBgNVBAMTEjAxMjNENkEzQTQ0
NkY1MDlFRTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABB1+P/eSppnw+YkxDYX+4WlBFtA/sleR
o2m8rcUi3QnDe6EqS8BIpIcOyKp1oPlgooLjGbCHDkFrWE2GONX0IeWjAjAAMAoGCCqGSM49BAMC
A0cAMEQCIFU/xv/uGiuRwOUKteTkmUcibNSiXWhdgqfh4l1vy5J5AiACJJWQP5ba8lp67DCVemIv
TBwJsdBtqNp8ofgJHRfiGg==
-----END CERTIFICATE-----
SHA1: dd744ac88443435edca84e56428dc2b32c550868

Downloads

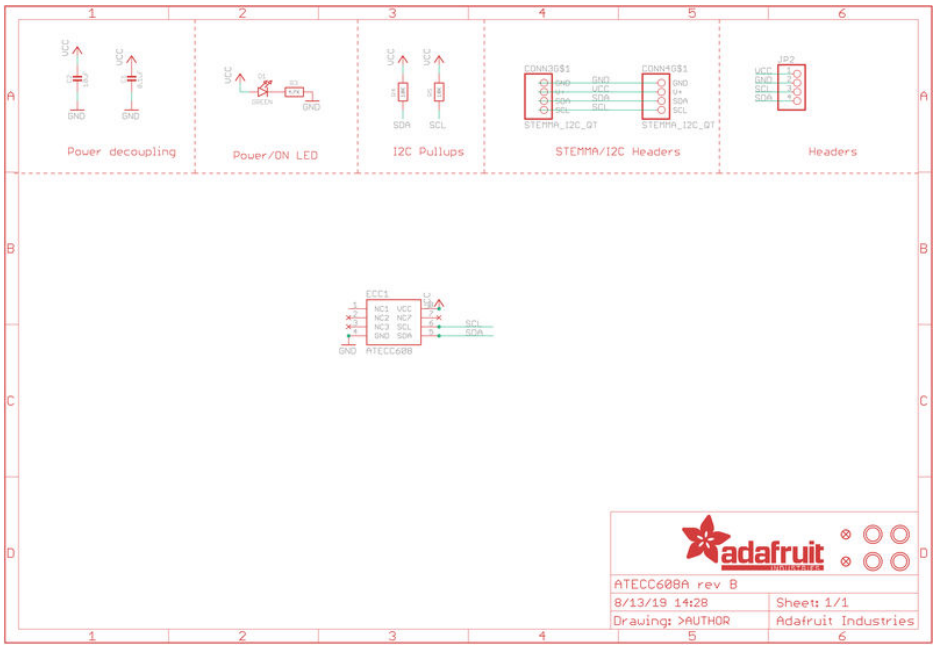
Files

- ATECC608A Data Sheet (https://adafru.it/Flp)
- ATECC508A Data Sheet (full) - the ATECC608A full datasheet is under NDA, but this datasheet is considered a 'subset' and can be used for reference! (https://adafru.it/Flg)
- EagleCAD files on GitHub (https://adafru.it/Flq)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/Flr)

Fab Print



Schematic


Power decoupling
Power/ON LED
I2C Pullups
STEMMA/I2C Headers
Headers
ATECC608A rev B
8/13/19 14:28
Sheet: 1/1
Drawing: >AUTHOR
Adafruit Industries