

Secure Remote Sensor Simulator

M.S. Final Exam

Ram Rohit Gannavarapu



Colorado State University

Agenda

- Introduction & Motivation
- Background and Related work
- System Design
- Hardware Design
- Software Design
- Securing Cloud and external communication
- Conclusion and Future work

Introduction

- An estimated 2,811,185 people were involved in motor vehicle accidents in the United States in 2018 [1].
- The litigation and settlements involving these crashes can reach millions of dollars
- Heavy Vehicle Event Data Recorders (HVEDR) can record event data related to diagnostic faults, hard or quick stops, and the vehicle's last or most recent stop.
- The data from HVEDRs help investigators determine important details about the crash



[1]

Image source: [J. Daily, et al.] "Extracting Event Data from Memory Chips within a Detroit Diesel DDEC V,"

Motivation

- Accessing the smart sensor device over the cloud enabling investigators to operate these devices remotely
 - Secure IoT communication.
- Software Defined Truck
- The Forensic tools have very minimal, or no security measures implemented.
- Despite growing role of embedded systems, their security capabilities remain weak. This thesis implements a way to increase security posture of embedded IoT devices.

Agenda

- Introduction & Motivation
- **Background and Related work**
- System Design
- Hardware Design
- Software Design
- Securing Cloud and external communication
- Conclusion and Future work

Background and Related work

Extracting data from HVEDRs

Extracting directly from vehicle diagnostic port

- OEM specific tools
- RP1210 is used for accessing diagnostic interfaces in the truck environment.
- Only possible when electricals of the vehicles are not compromised

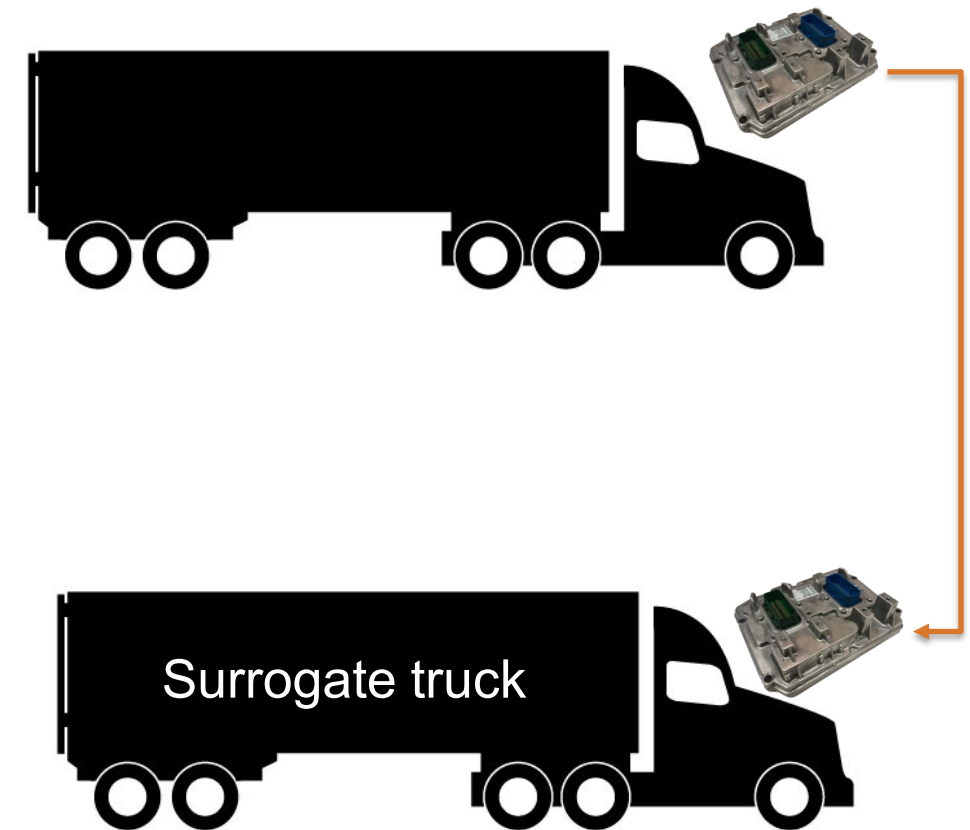


Background and Related work

Extracting data from HVEDRs

Surrogate vehicle method

- Locate a virtually identical sister truck
- Remove the ECM from the severed truck and replace it in the sister truck
- Drawbacks
 - Expensive (Rental/Leasing)
 - Hard to Find



Background and Related work

Extracting data from HVEDRs

Truck in a Box system

- Simulates different passive sensors using Potentiometers
- Compatible only for specific make/model

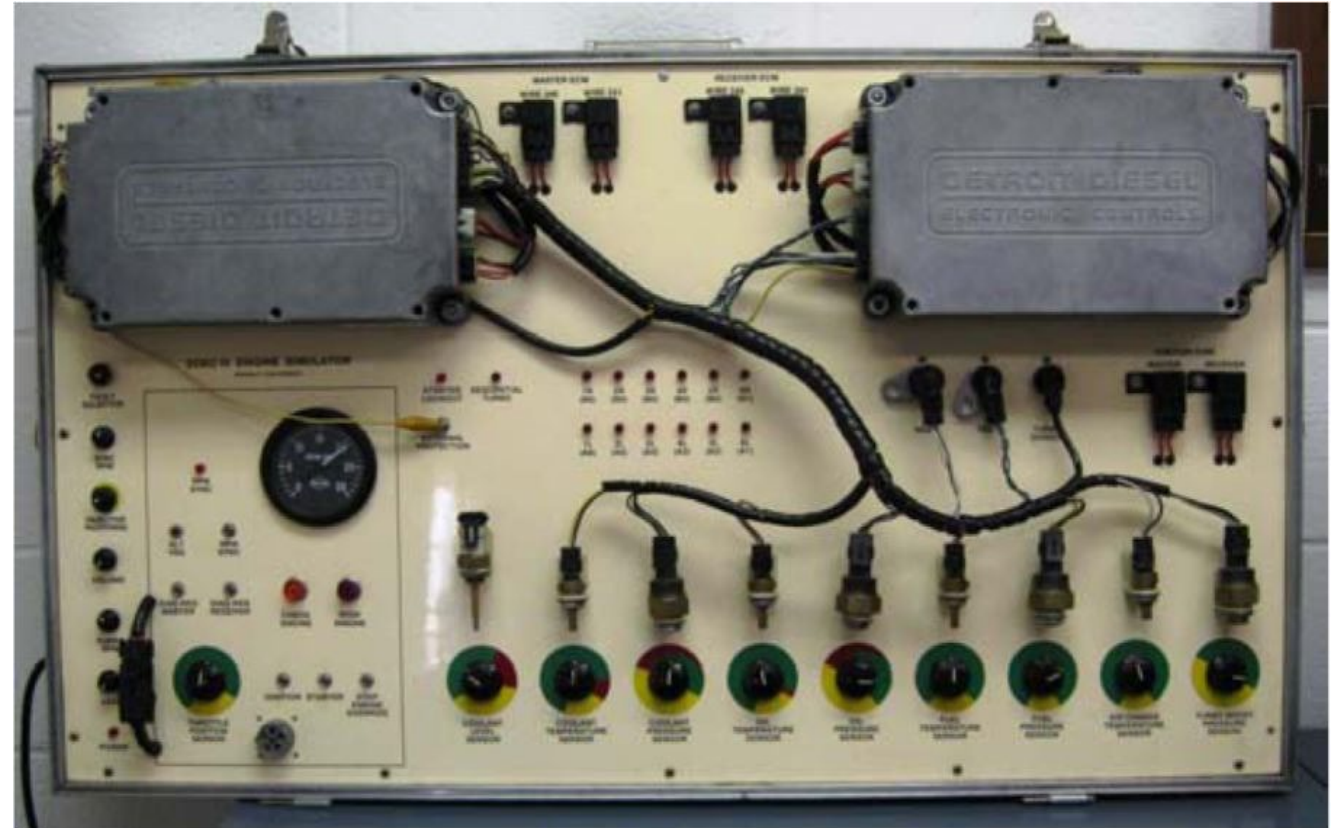


Image source: [4]

Background and Related work

SSS and SSS2



Common Sensors found in a HV

- Engine side harness
 - Crankshaft position sensor
 - Timing reference sensor
 - Ambient Air Temperature Sensor
- Vehicle side harness
 - Accelerator Pedal Position
 - Coolant level sensor
 - Vehicle speed sensor
- They can be categorized into
 - Two wire Sensors
 - Three wire sensors
 - Pulse Width Modulated signals

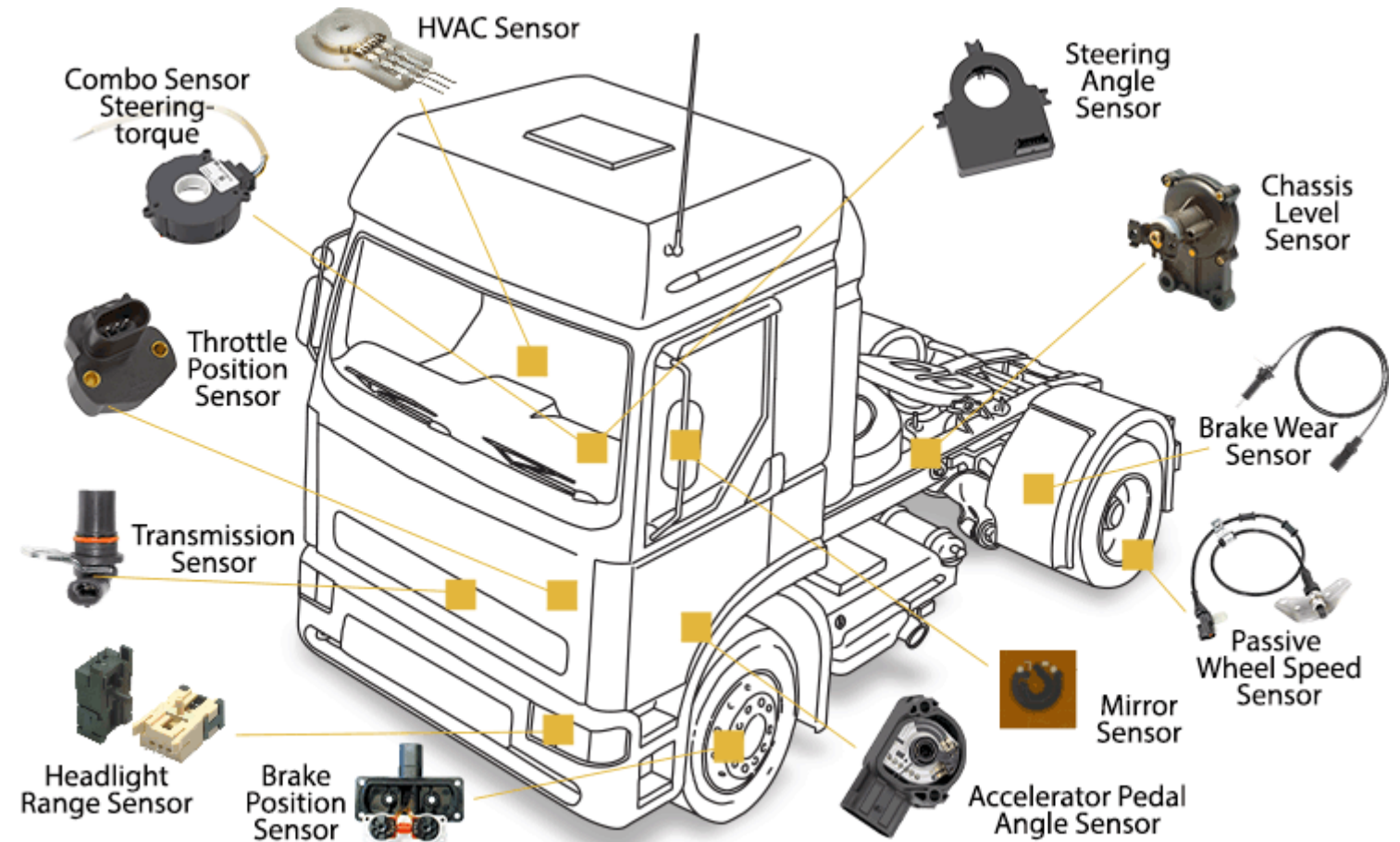


Image source: [Bourns - Commercial Vehicle Sensors](#)

Two Wire Sensors

Resistive Sensors

- Read internally by the ECU through a Wheatstone bridge or a voltage divider.
- Varying the resistance changes the sensor value.
- Some two sensors
 - Intake Manifold Temperature Sensor
 - Engine Coolant Temperature Sensor
 - Fuel Temperature Sensor
 - Inlet Air Temperature Sensor
 - Crankcase Pressure Sensor
 - Ambient Air Temperature Sensor
 - DEF tank level sensor

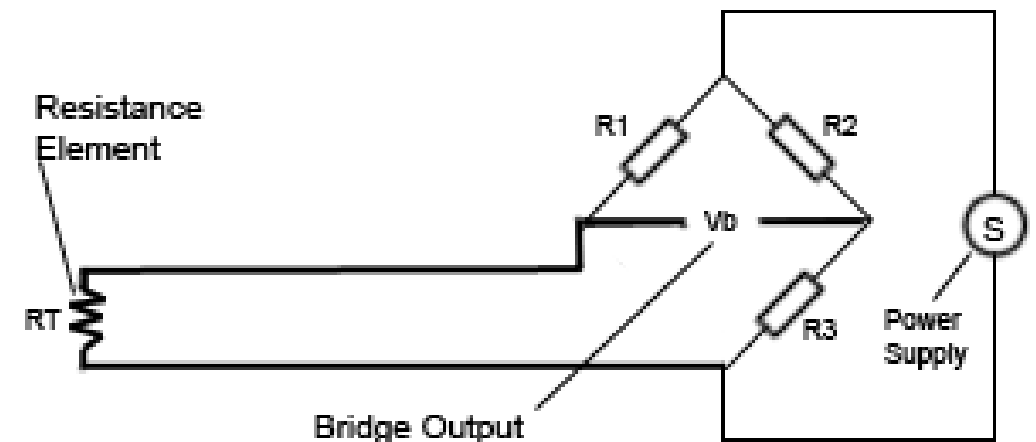
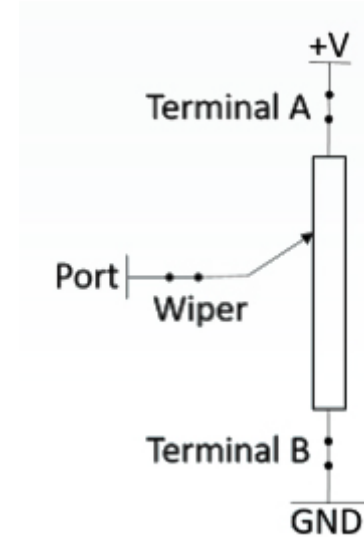


Image Source: [2 Wire RTD Temperature Sensor, RTD, Resistance Temperature Detector \(RTD\) \(thermometricscorp.com\)](http://thermometricscorp.com)

Three Wire Sensors

Analog Outputs

- Produce steady voltage from 0 to 5 /12 VDC
- Voltage dividers produce a voltage signal based the wiper position.
- Pressure sensors.



Pulse Width Modulated Signals

- Accelerator Pedal Position
- Throttle position sensor
- Wheel speed sensor
- Electronic Dosing Valve

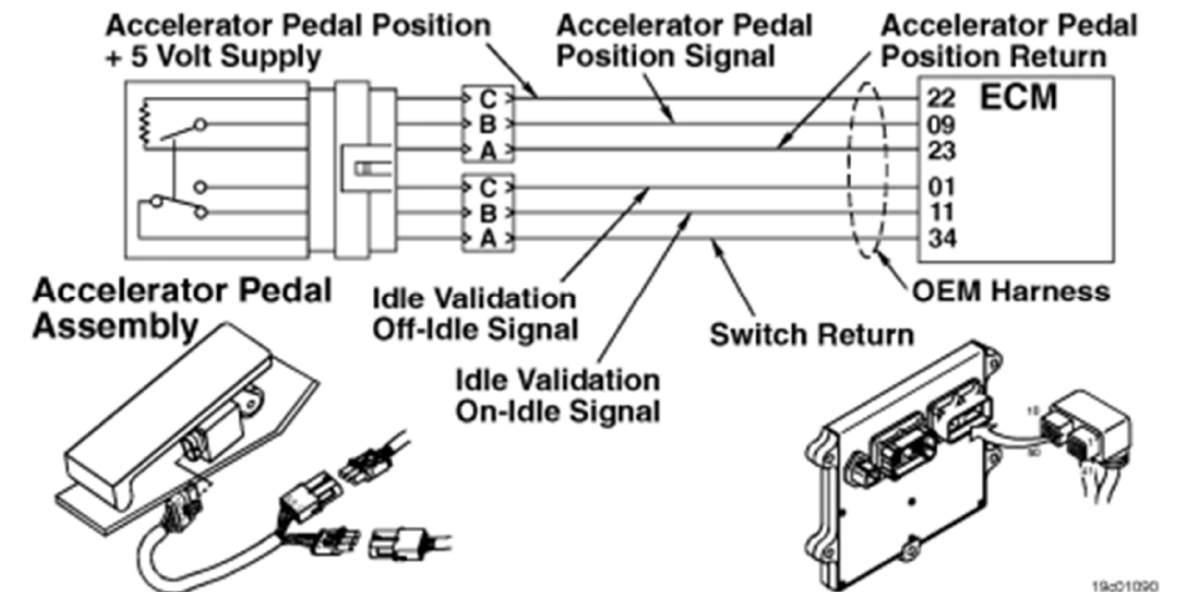
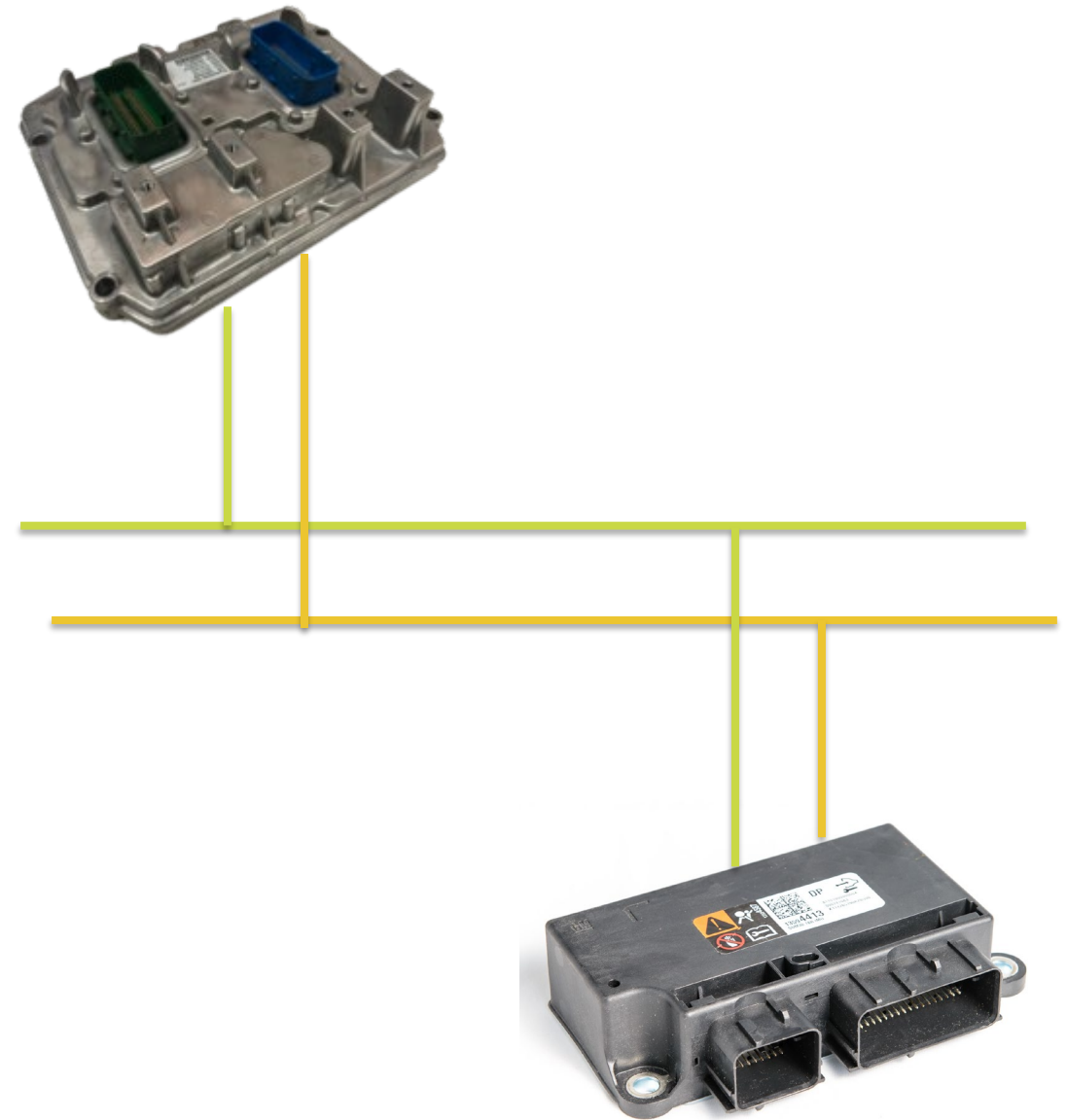


Image Source: J. L. Córcega, "Design of a forensically neutral electronic environment for heavy vehicleevent data recorders," Master's thesis, University of Tulsa, 2015

CAN based Sensors/Modules

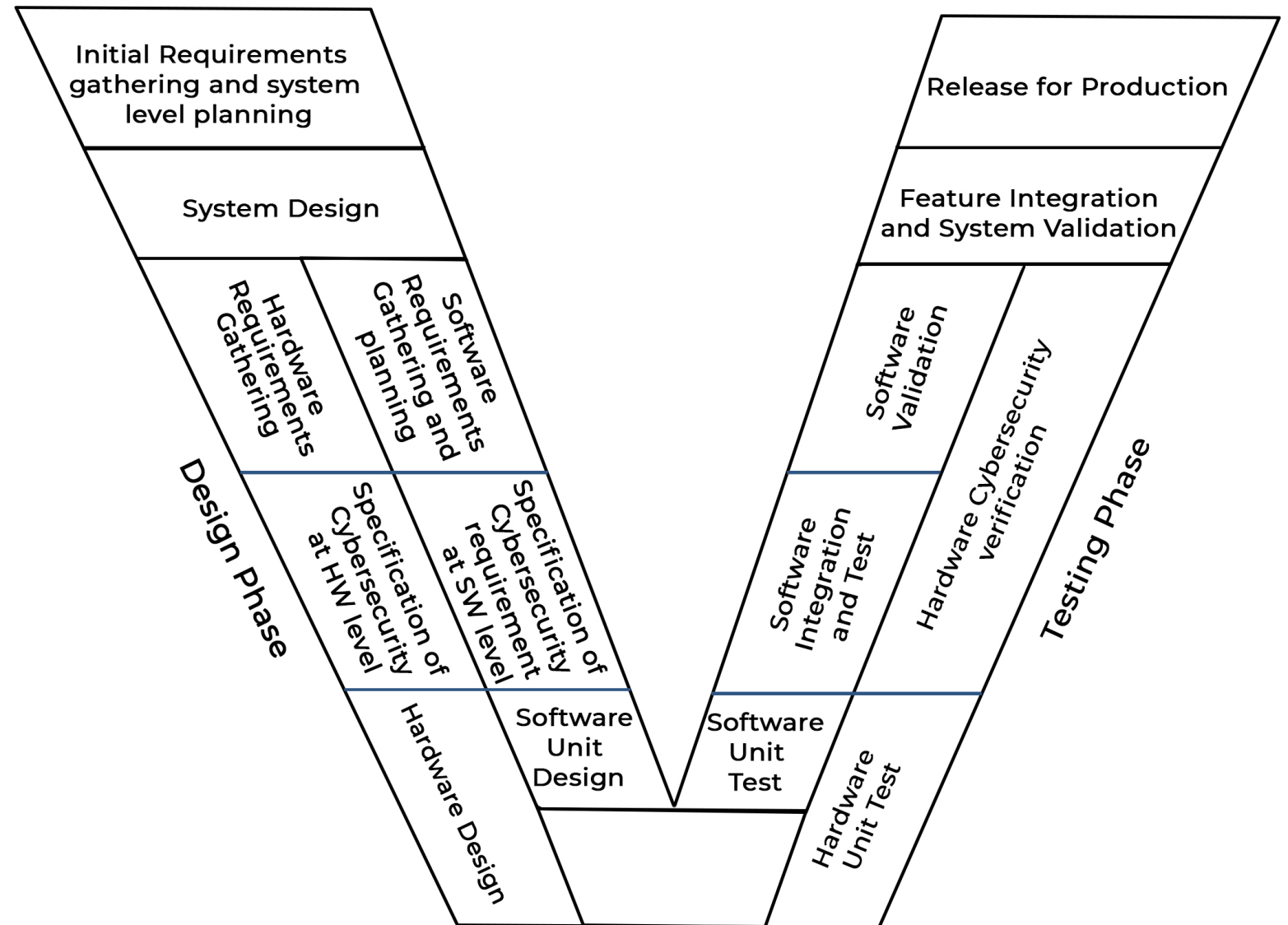
- Depending on the engine configuration, some actuators are controlled over CAN.
- However, neither engine CAN nor any standard CAN network (SAE J1939) are used to actuate these sensors. Instead, modules such as the Aftertreatment control module (ACM) or Motor Control Module (MCM) have a dedicated CAN lines to control these sensors.



Agenda

- Introduction & Motivation
- Background and Related work
- **System Design**
- Hardware Design
- Software Design
- Securing Cloud and external communication
- Conclusion and Future work

System Design



ISO/SAE21434

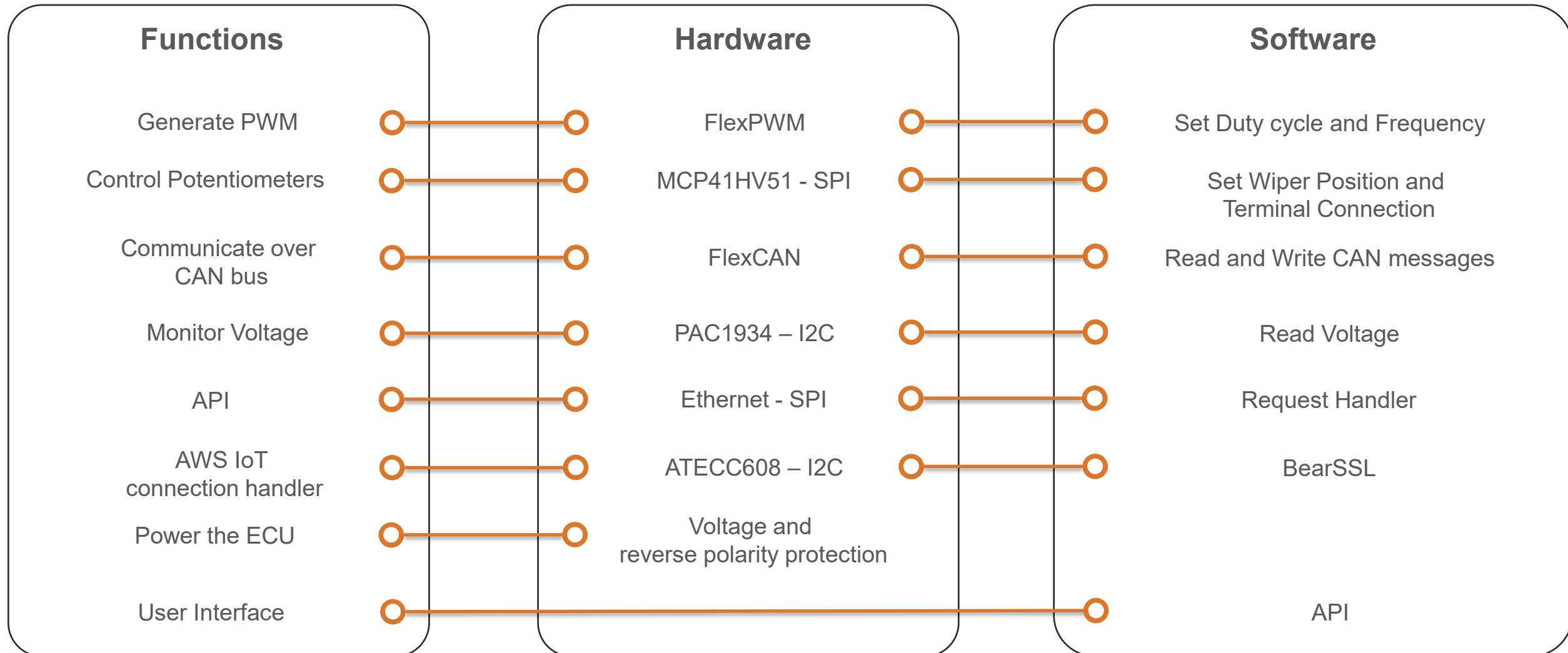
System Design

Requirements

- Simulate Sensors and actuators to create a fault free environment
 - Emulate resistive sensors
 - Generate analog voltages (0.25 - 4.75V)
 - Generate PWM signals
 - Simulate CAN based modules/sensors
- Operate device remotely.
- Provide feedback about the state of the device to users.
- User Interface

System Design

Function Allocation



Agenda

- Introduction & Motivation
- Background and Related work
- System Design
- **Hardware Design**
- Software Design
- Securing Cloud and external communication
- Conclusion and Future work

Hardware Design



Hardware Design

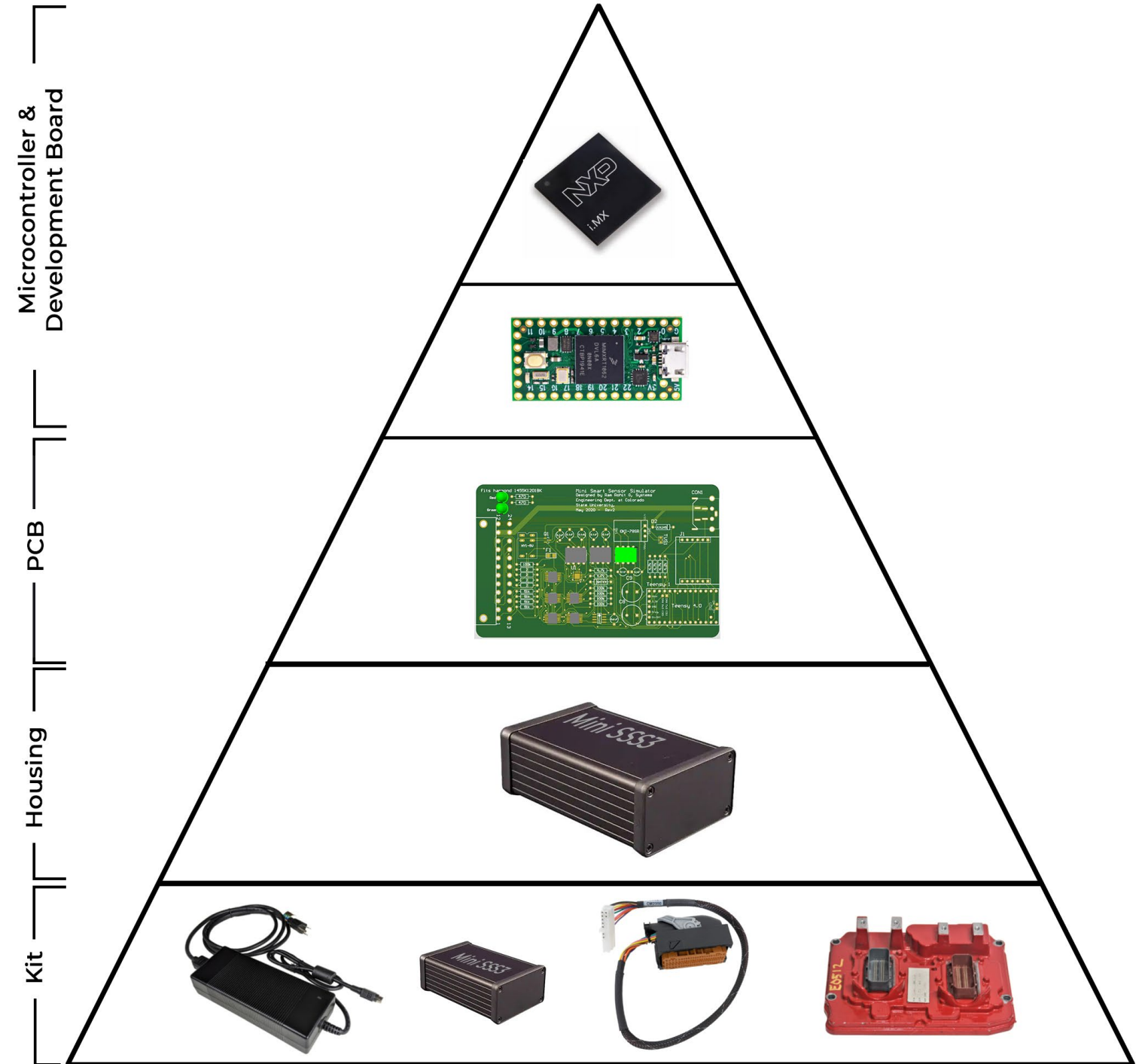
Requirements

- Generate PWM signals
- Simulate resistance to ground (two wire sensors)
- Generate analog voltages (three wire sensors)
- Ethernet Interface
- Measure Voltage for feedback
- Hardware based secure key storage

Hardware Design

Approach

- Teensy 4.0 Development Board
 - NXP iMXRT1062
 - 600 MHz Cortex-M7
 - 3 - SPI
 - 3 – I2C
 - 3 – CAN (1 FD)
 - 2Mb Flash, 1024K RAM

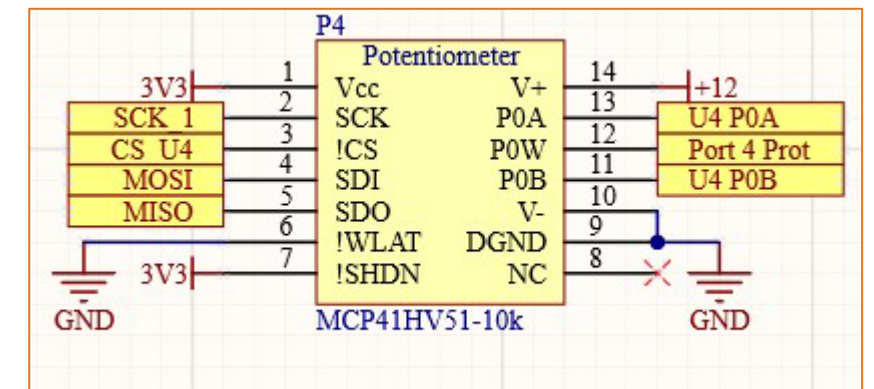
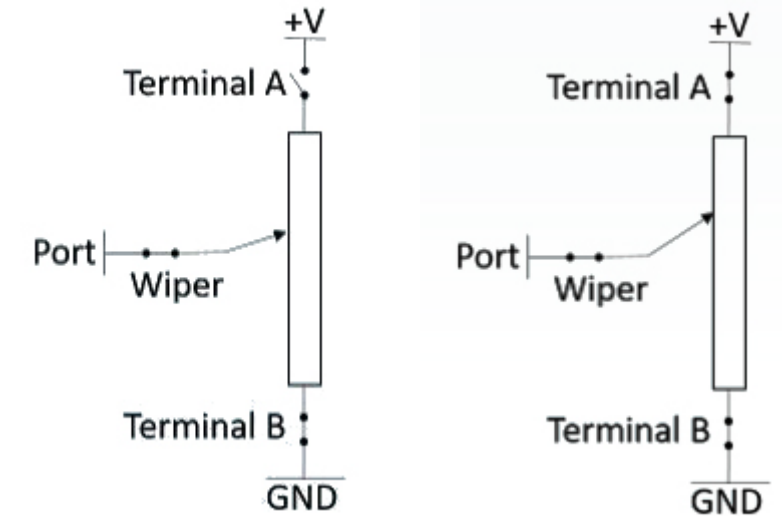


Hardware Design

Emulating Resistive sensors/ Generating Analog Outputs

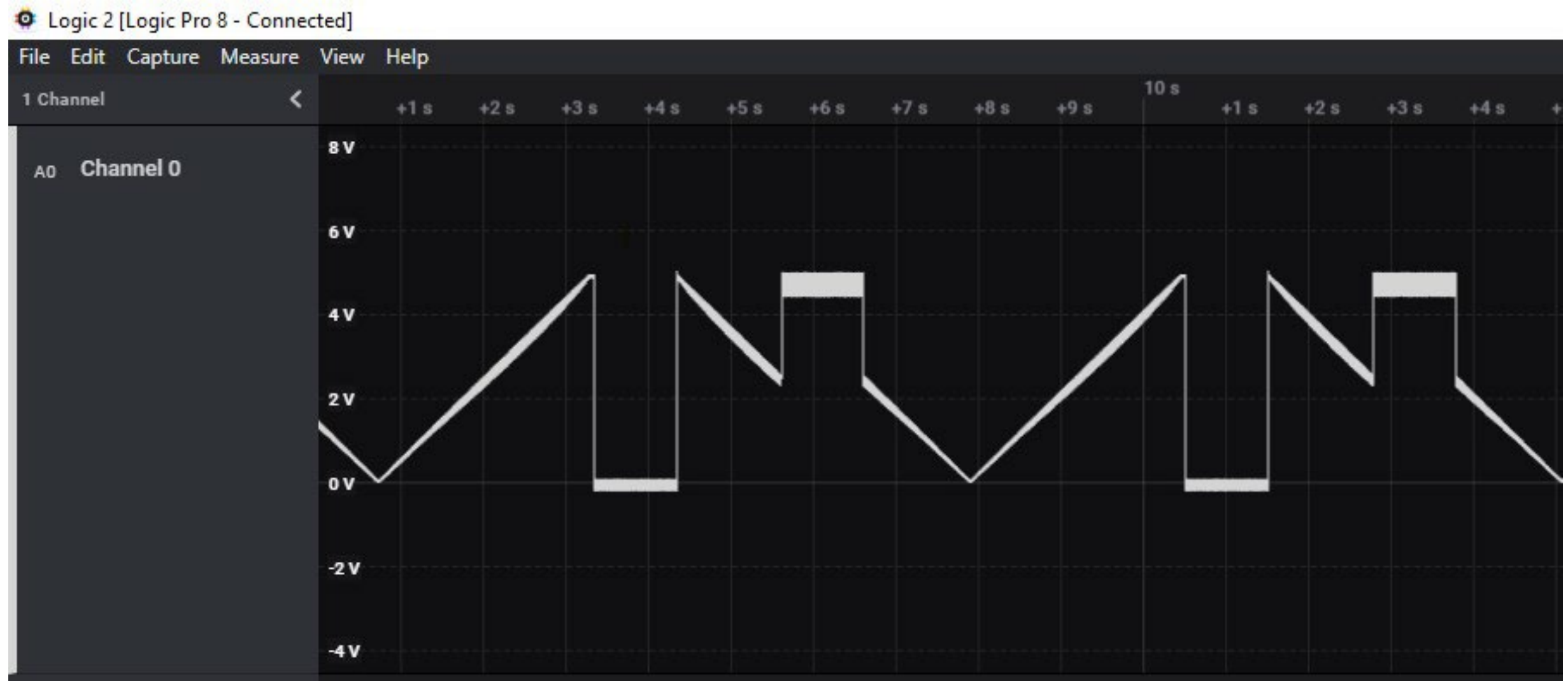
- Microchip MCP41HV51 Digital Potentiometer
 - 8-bit wiper position resolution
- Terminal Connections register (TCON)

R-1	R-1	R-1	R-1	R/W-1	R/W-1	R/W-1	R/W-1
D7	D6	D5	D4	R0HW	R0A	R0W	R0B
bit 7				bit 0			



Hardware Design

Emulating Resistive sensors/ Generating Analog Outputs



Hardware Design

Generating PWM Signals

- Utilizing Hardware based PWM modules
 - Counter
 - Comparator
- PWM signals generated from Teensy are 3.3v and needs to be amplifies to 5V.
- Limitation with setting frequency.
 - Each PWM module drives up to 3 PWM outputs.
 - One Counter/Timer linked to a PWM module.

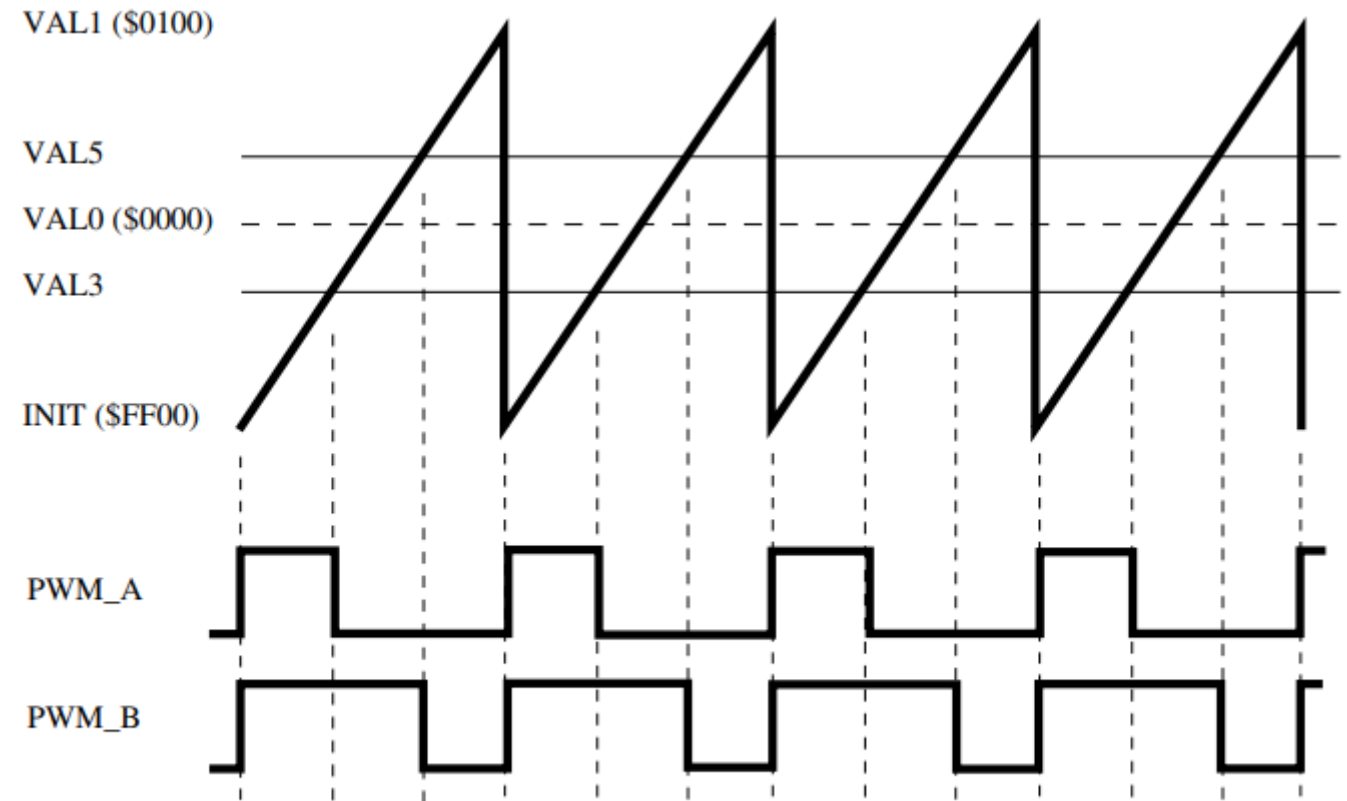
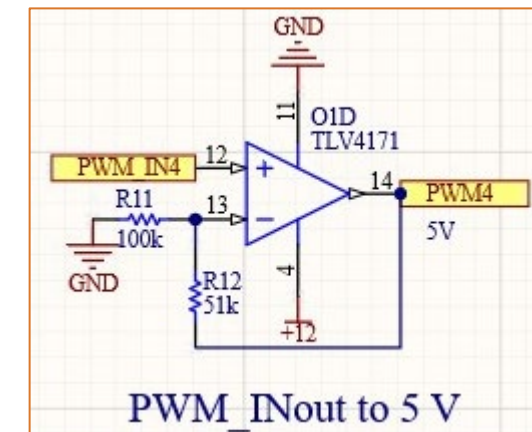
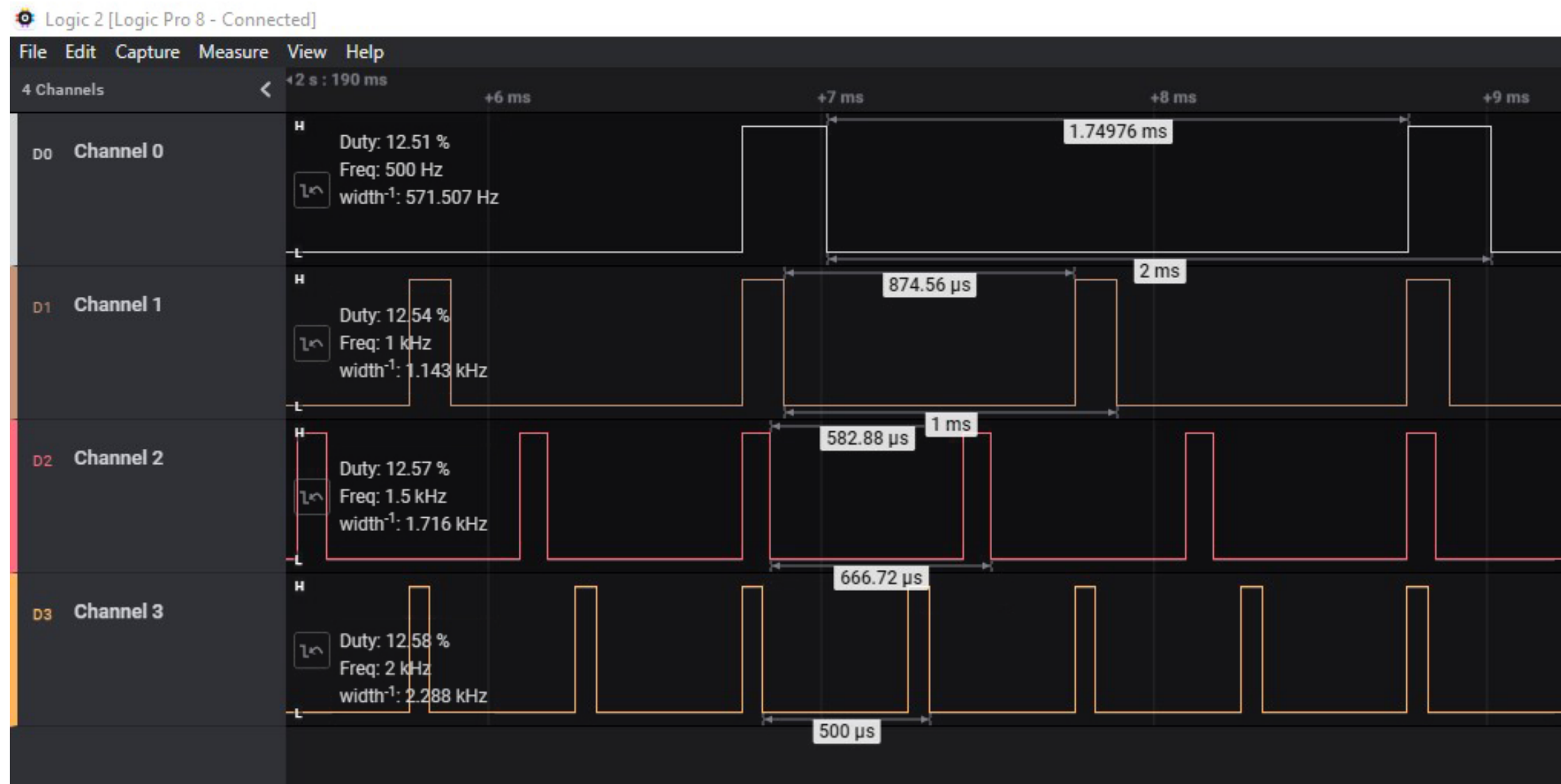


Figure 55-4. Edge Aligned Example (INIT=VAL2=VAL4)



Hardware Design

Generating PWM Signals with varying frequency



Hardware Design

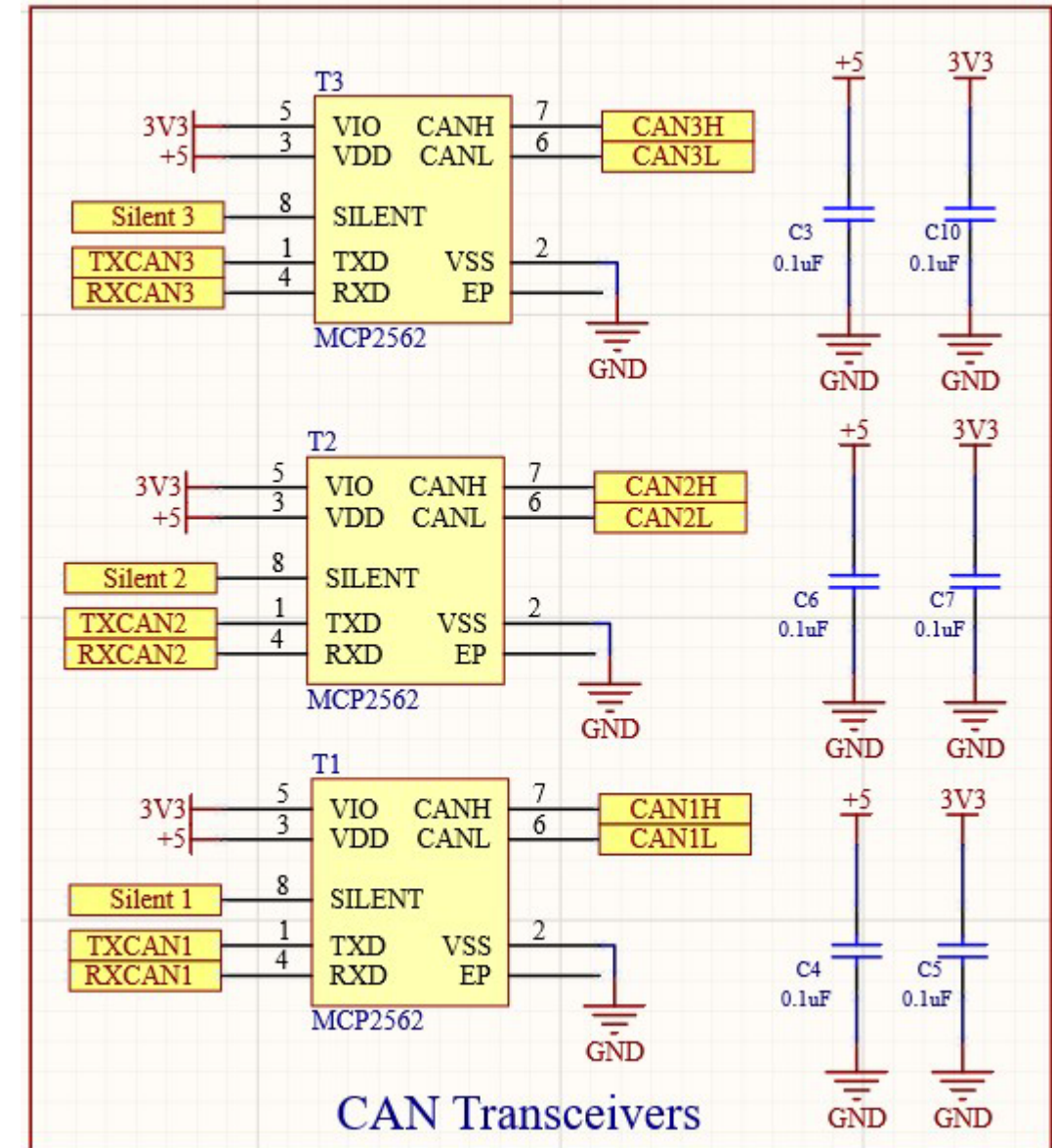
Generating PWM Signals with varying duty cycle



Hardware Design

Read/Write CAN messages

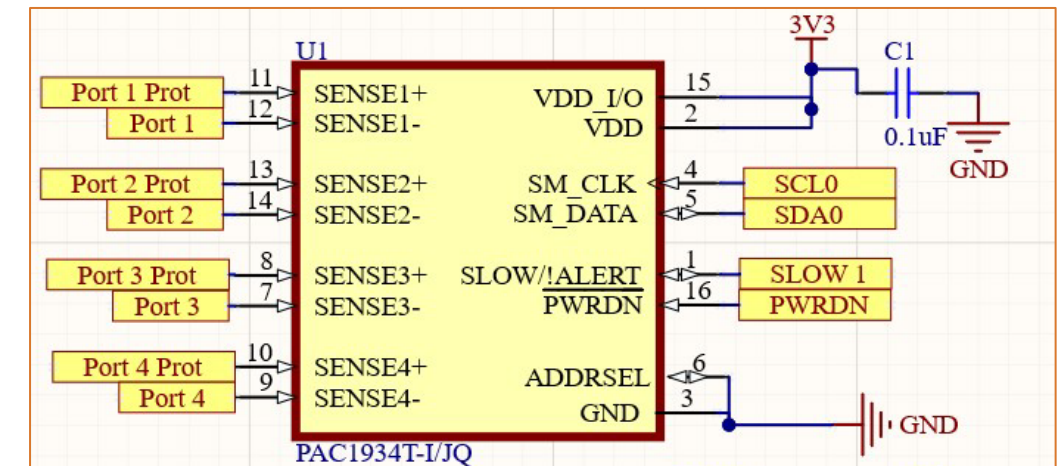
- Teensy 4.0 has 3 in built CAN controllers
- Additional CAN transceivers is added to interface with the CAN bus.



Hardware Design

Microchip PAC1934

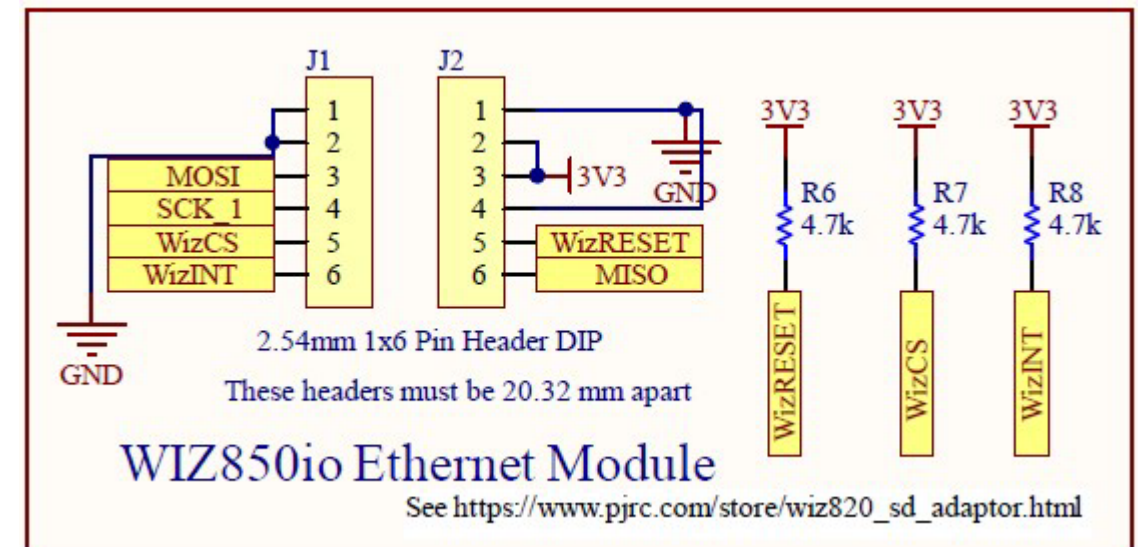
- PAC1934 is a four-channel power/energy monitor.
- 16-bit resolution
- 1 to 32 volts support.
- Up to 1024 samples per second
- I2C interface
- Arduino Library
 - https://github.com/SystemsCyber/Microchip_PAC1934x.git



Hardware Design

Wiz850io Ethernet module

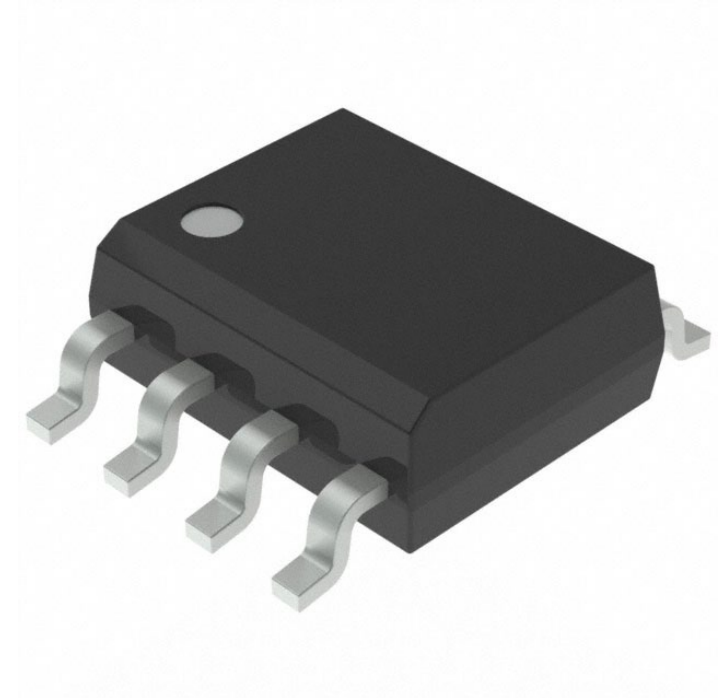
- Plugin Network Module
- Powered by W5500 chip
 - Hardwired TCP/IP embedded ethernet controller
- High speed SPI interface



Hardware Design

Microchip ATECC608B

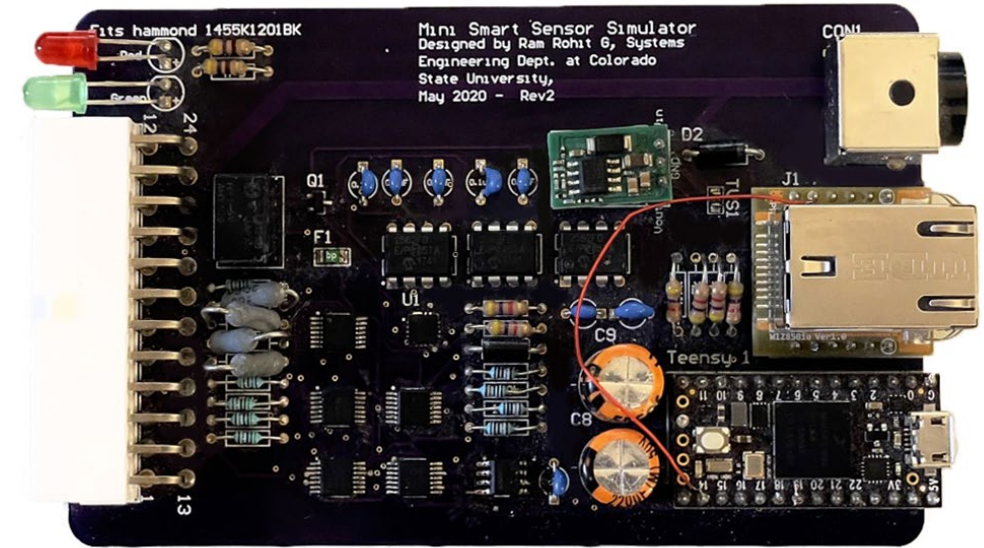
- Cryptographic co-processor with secure hardware-based key storage
- 16 key storage slots with 256-bit key size
- NIST standard P256 elliptic curve support (ECC)
- True Random Number Generator (TRNG)
- I2C or SWI interface
- [ArduinoECCX08](#)
- Teensy Compatibility ([Github Issue](#))



Hardware Design

PCB Design

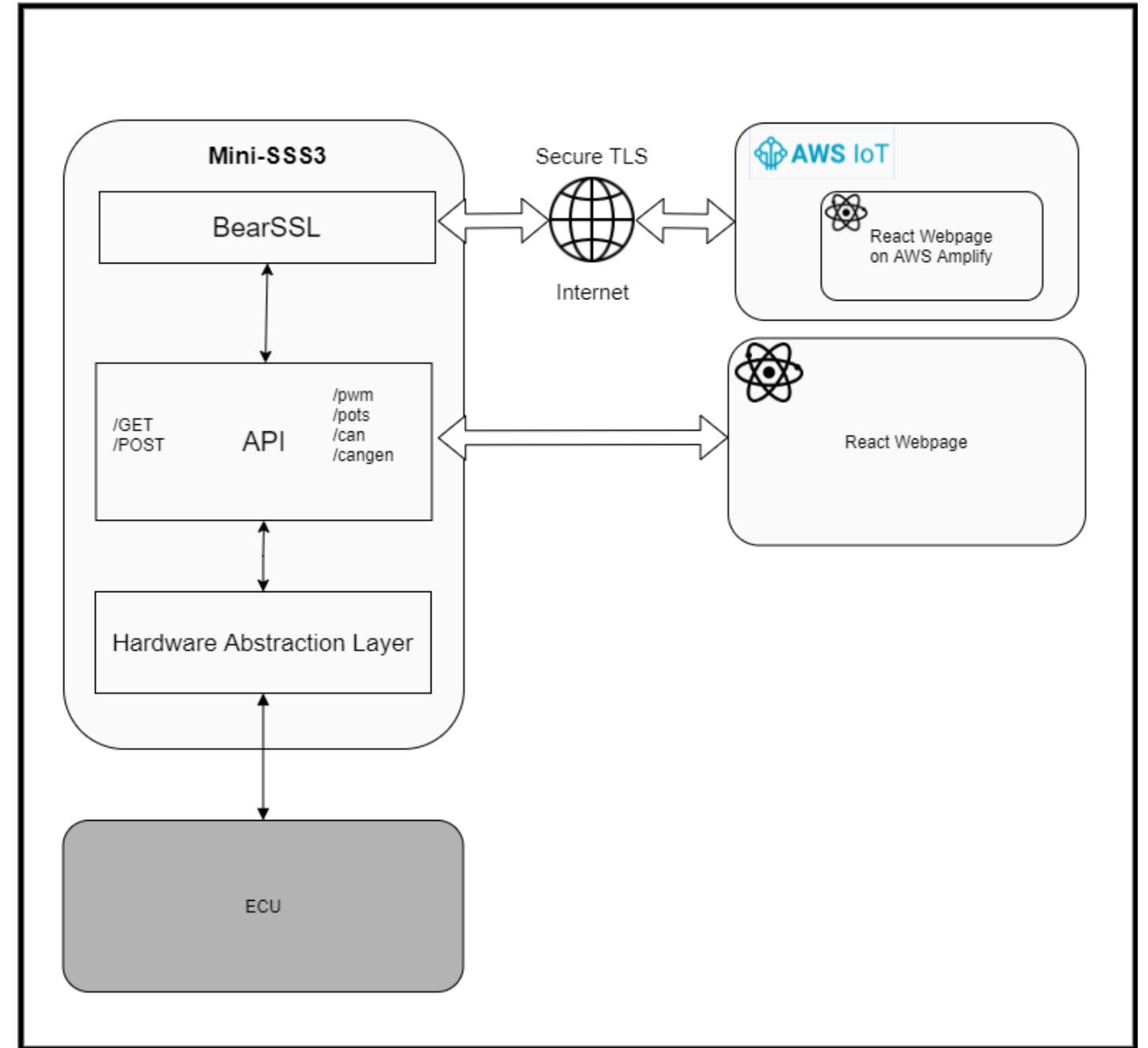
- Designed using Altium Designer
- Teensy 4.0 was integrated into a printed circuit board (PCB) along with other peripherals such as :
 - MCP41HV51 digital potentiometer
 - Microchip PAC1934
 - Voltage regulator and power protection
 - Microchip ATECC608B HSM
 - Op-amp for PWM outputs



Agenda

- Introduction & Motivation
- Background and Related work
- System Design
- Hardware Design
- **Software Design**
- Securing Cloud and external communication
- Conclusion and Future work

Software Design



Software Design

Hardware Abstraction Layer

Pots - SPI

Set Wiper Register
Set TCON Register

PWM

Set Frequency
Set Duty Cycle

CAN

Read Messages
Write Messages

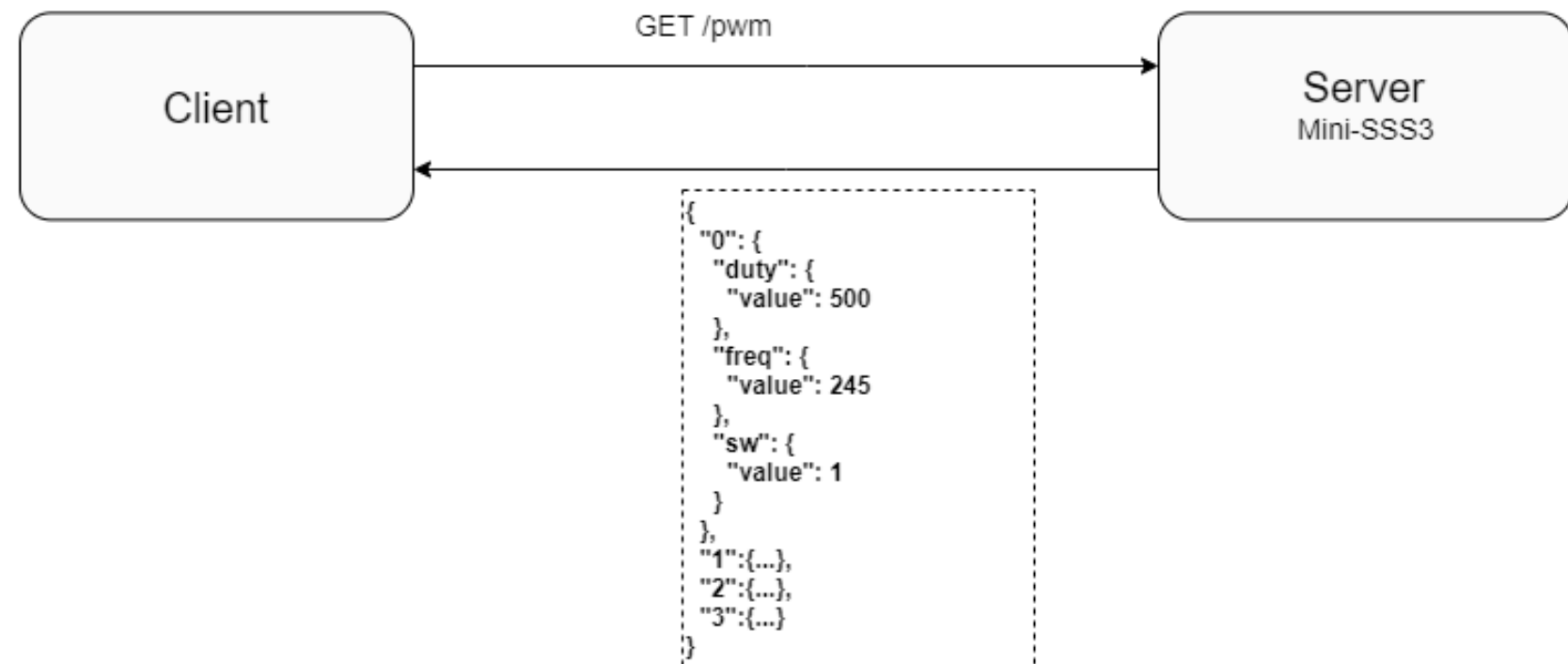
PAC – I2C

Read Voltage

Software Design

HTTP API Design

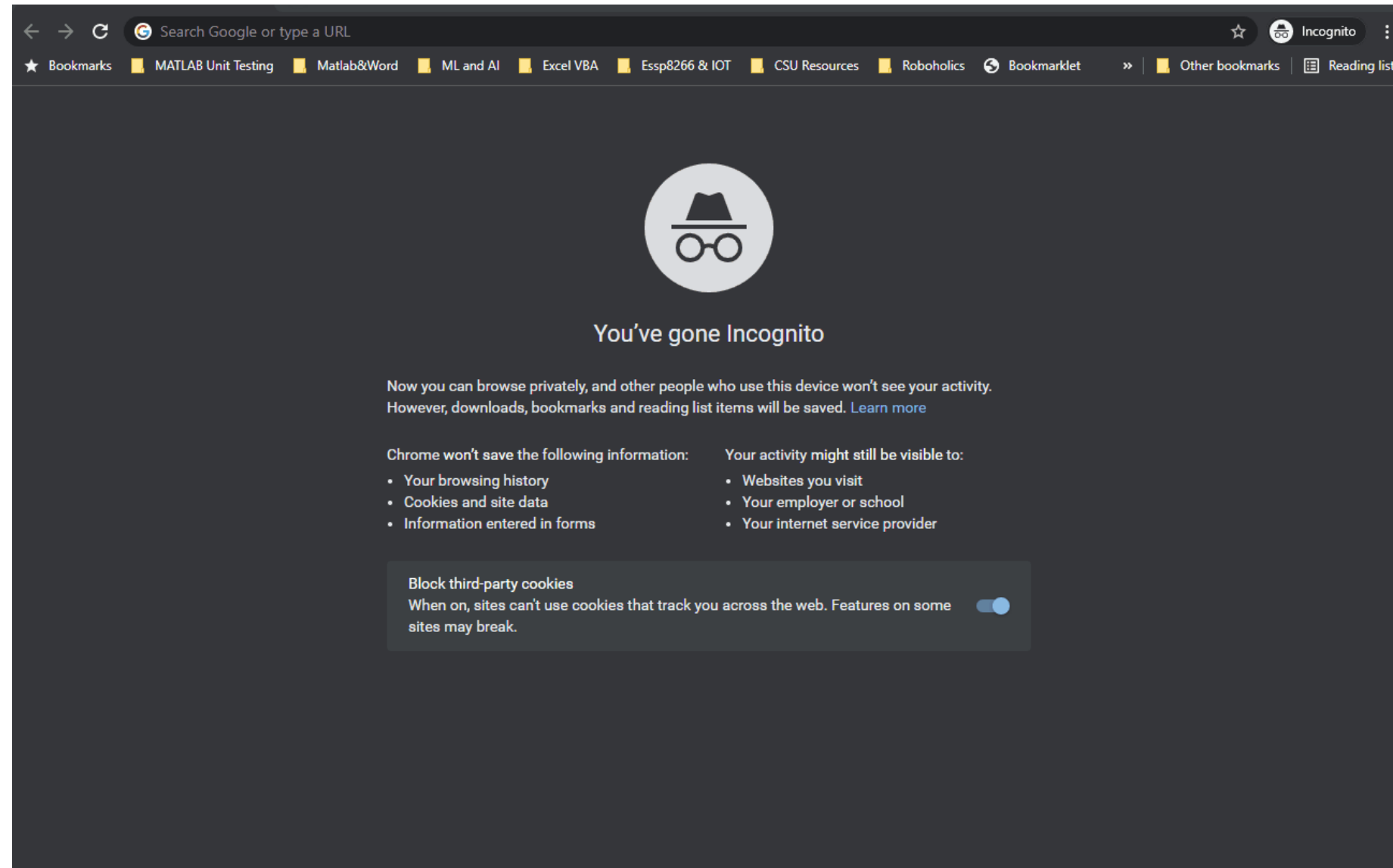
- API endpoint for the different peripherals
 - /pwm
 - /pots
 - /voltage
 - /can
 - /cangen



Software Design

Graphical User Interface

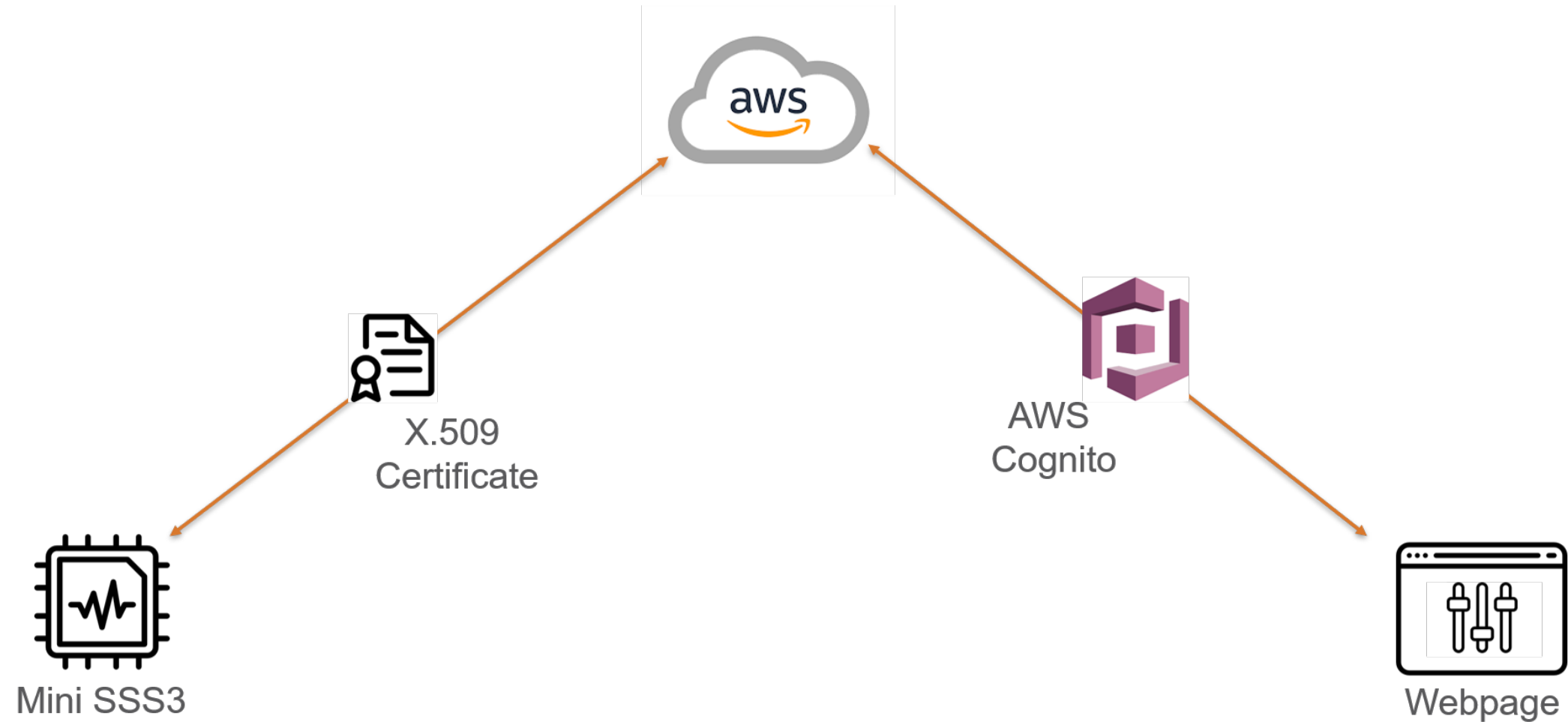
- Web based GUI
- React framework
- Served directly from Teensy 4.0
- [aWOT Arduino library](#)



Agenda

- Introduction & Motivation
- Background and Related work
- System Design
- Hardware Design
- Software Design
- Securing Cloud and external communication
- Conclusion and Future work

Securing the Cloud Communication



X.509 Certificates

- Digital certificates are the most common approach used to distribute public keys.
 - User's public key along with other identification information of a user is cryptographically signed by the third party using the third party's secret signature key
 - certificate binds this user identity to its public key
 - Clients must have the public key of the root CA in order to verify the public key
- BearSSL is used to implement TLS communication on the Mini-SSS3 with AWS IoT

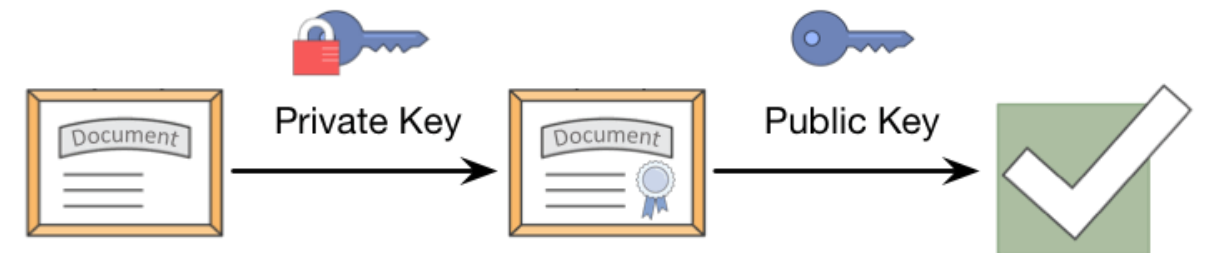
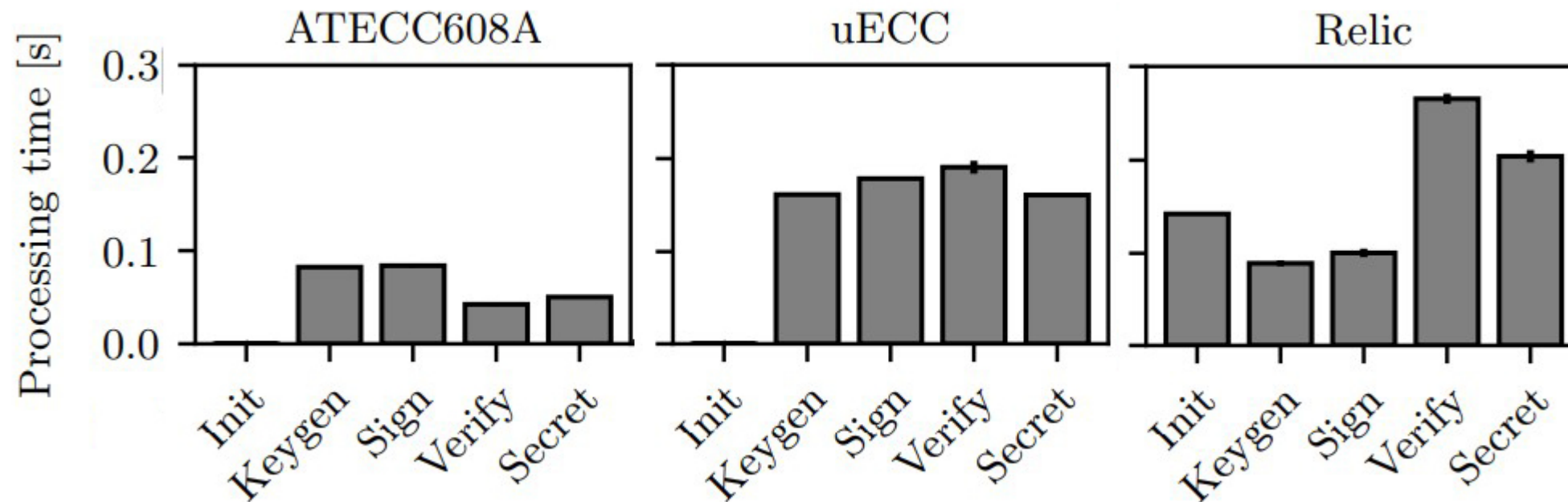


Image source: [Understanding the AWS IoT Security Model | The Internet of Things on AWS – Official Blog \(amazon.com\)](https://aws.amazon.com/blogs/iot/understanding-the-aws-iot-security-model/)

Offloading ECDSA sign/verify to ATECC608

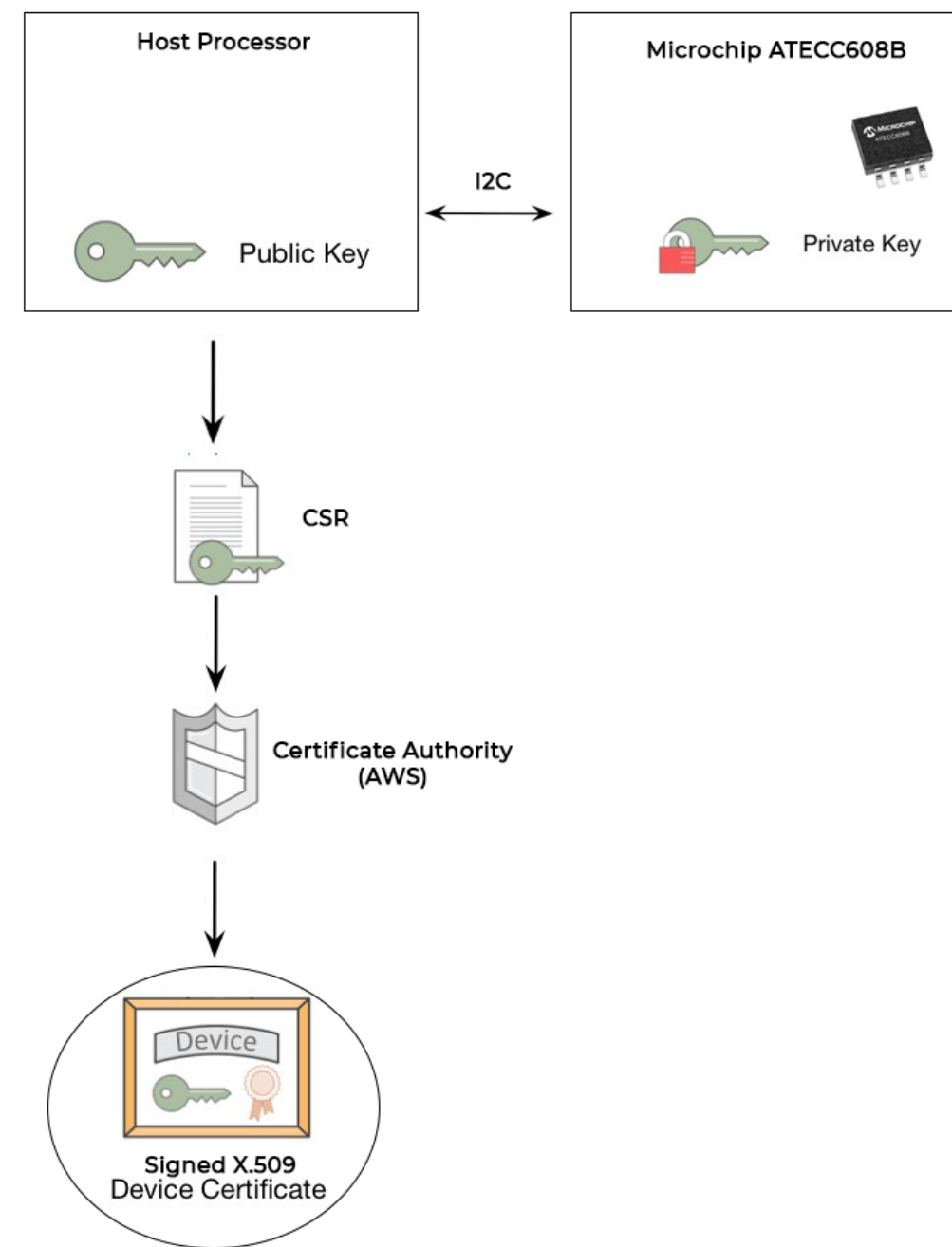


P. Kietzmann, L. Boeckmann, L. Lanzieri, T. C. Schmidt, and M. Wählisch, "A Performance Study of Crypto-Hardware in the Low-end IoT," 058, 2021. Accessed: Dec. 10, 2021. [Online]. Available: <http://eprint.iacr.org/2021/058>

Provisioning the ATECC608B

- Generate a private key Internally on the ATECC608B and lock the device
- Generate a Certificate Signing Request (CSR) of the publickey along with other user information from the private key locked in ATECC608B
- Provide CSR to a Certificate Authority (AWS IoT)
- Download the signed X.509 Certificate to the device

Private key never leaves the ATECC608 during the provisioning process



Communication with AWS IoT

- MQTT is a lightweight, publish-subscribe network protocol that transports messages between devices
- Changes on the device and the webpage and are relayed onto a common MQTT Topic

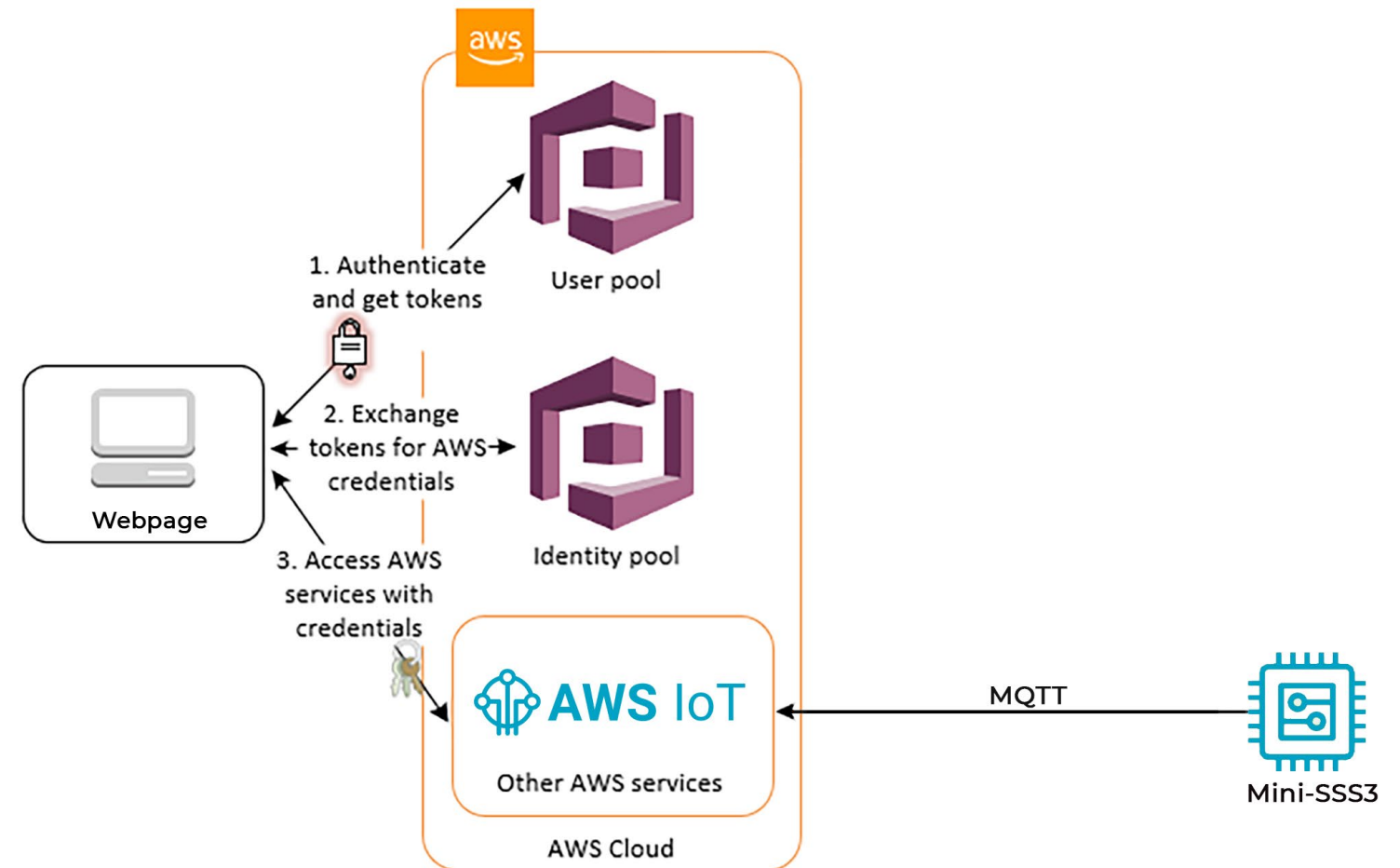


Image source: [Understanding the AWS IoT Security Model | The Internet of Things on AWS – Official Blog \(amazon.com\)](https://aws.amazon.com/blogs/iot/understanding-the-aws-iot-security-model/)

Demonstration



Image source: [Understanding the AWS IoT Security Model | The Internet of Things on AWS – Official Blog \(amazon.com\)](https://aws.amazon.com/blogs/iot/understanding-the-aws-iot-security-model/)

Contribution and Conclusion

- Hardware design of the Mini-SSS3
- Contributed to open-source Arduino libraries for PAC1934 and ArduinoECCX08 to make them compatible with Teensy 4.0
- Designed a Web-based GUI which is served directly from the Mini-SSS3 device.
- Implemented a Cloud based backend and front end for Interacting with the device remotely.

Limitations and Future Work

- Utilize ATECC608B-TNGTLS
 - Pre provisioned variant of ATECC608B
 - Signed Thumbprint certificate pre provisioned on the device which can be used to authenticate with AWS IoT, Azure and Google Cloud.
 - Eliminate the provisioning steps.
- Transition the HTTP API to HTTPS
- Serve HTTPS web-pages directly from the device.
- Implement Authentication mechanism on the device.

Thank you



Colorado State University

References

- [1] “Overview of the 2018 Crash Investigation Sampling System,” NHTSA. [Overview of the 2018 Crash Investigation Sampling System \(dot.gov\)](#)
- [2] P. Kietzmann, L. Boeckmann, L. Lanzieri, T. C. Schmidt, and M. Wählisch, “A Performance Study of Crypto-Hardware in the Low-end IoT,” 058, 2021. Accessed: Dec. 10, 2021. [Online]. Available: <http://eprint.iacr.org/2021/058>
- [3] S. A. B. van Nooten and J. R. Hrycay, “The Application and Reliability of Commercial Vehicle Event Data Recorders for Accident Investigation and Analysis,” SAE International, Warrendale, PA, SAE Technical Paper 2005-01–1177, Apr. 2005. doi: [10.4271/2005-01-1177](https://doi.org/10.4271/2005-01-1177).
- [4] D. Plant, T. Austin, and B. Smith, “Data Extraction Methods and their Effects on the Retention of Event Data Contained in the Electronic Control Modules of Detroit Diesel and Mercedes-Benz Engines,” SAE Int. J. Passeng. Cars – Mech. Syst., vol. 4, no. 1, pp. 636–647, 2011.

Appendix

- PWM Submodule on iMXRT1062

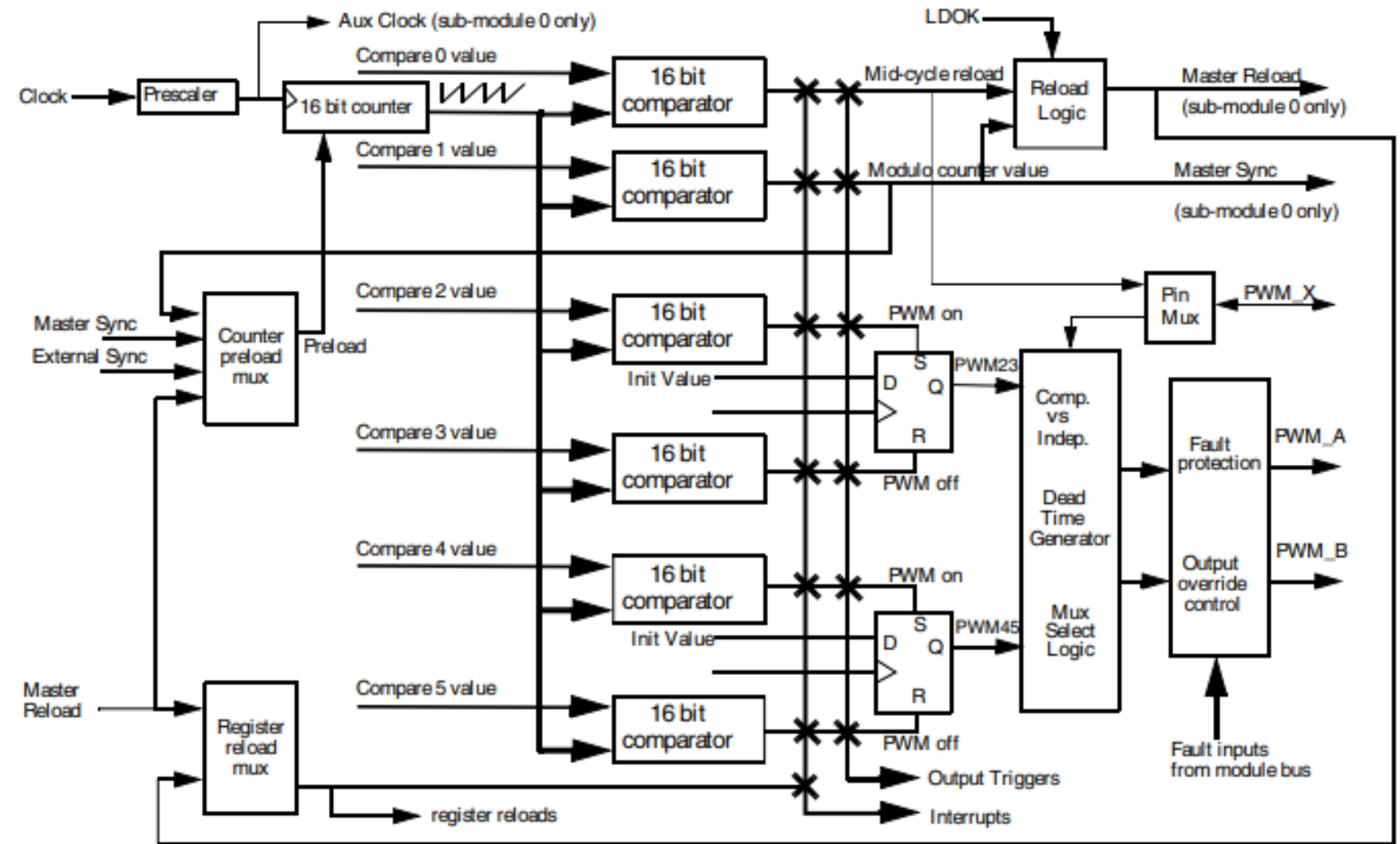


Figure 55-2. PWM Submodule Block Diagram

Client Hello

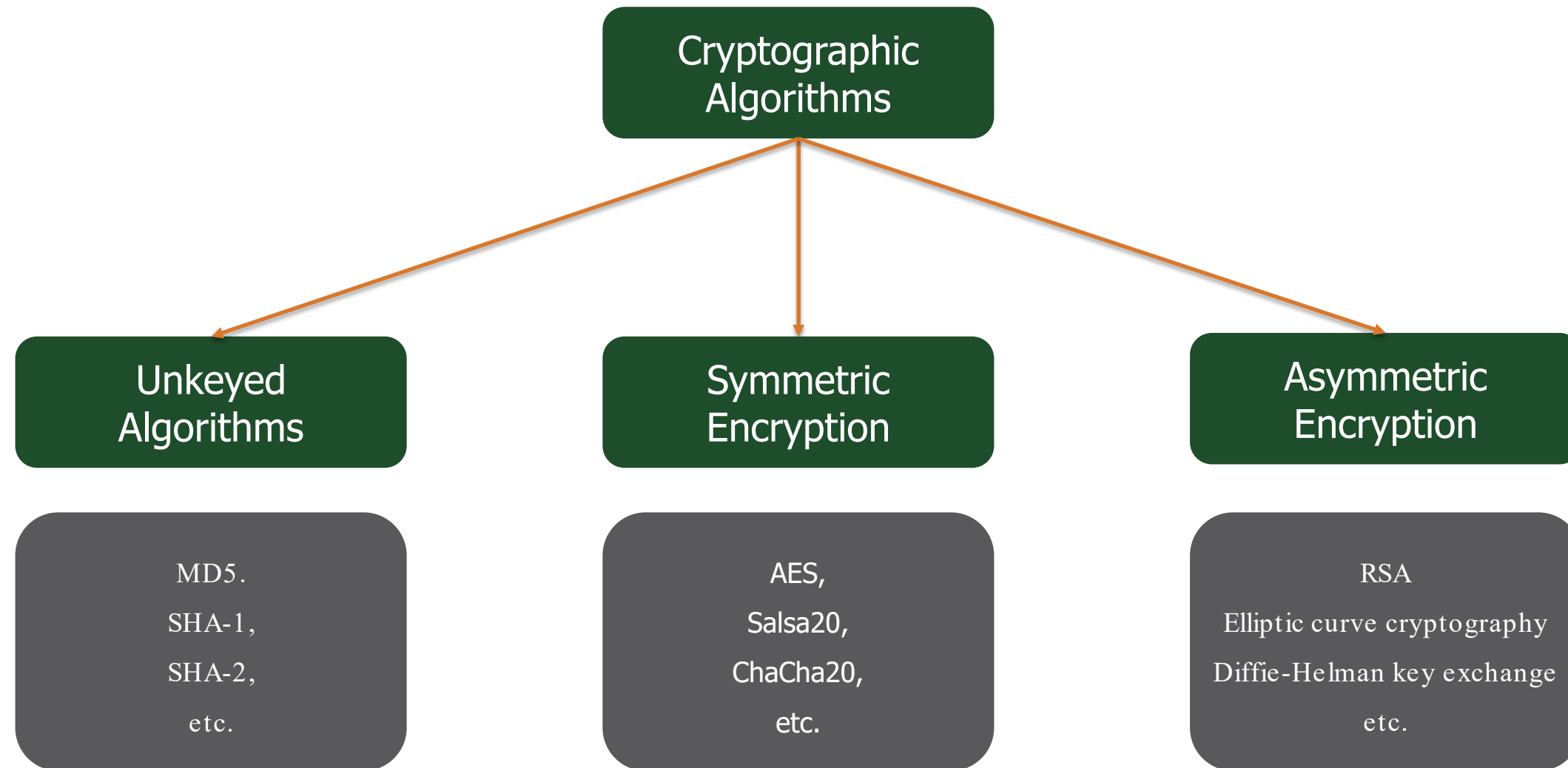
▼ Cipher Suites (45 suites)

```
Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CCM (0xc00ac)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CCM (0xc00ad)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 (0xc00ae)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8 (0xc00af)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02d)
Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256 (0xc031)
Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02e)
Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384 (0xc032)
Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 (0xc025)
Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 (0xc029)
Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 (0xc026)
Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 (0xc02a)
Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xc004)
Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (0xc00e)
Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xc005)
Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xc00f)
Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
Cipher Suite: TLS_RSA_WITH_AES_128_CCM (0xc009)
Cipher Suite: TLS_RSA_WITH_AES_256_CCM (0xc00d)
Cipher Suite: TLS_RSA_WITH_AES_128_CCM_8 (0xc00a)
Cipher Suite: TLS_RSA_WITH_AES_256_CCM_8 (0xc00b)
Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c)
Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003d)
Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc008)
Cipher Suite: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (0xc012)
Cipher Suite: TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc003)
Cipher Suite: TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA (0xc00d)
Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
```

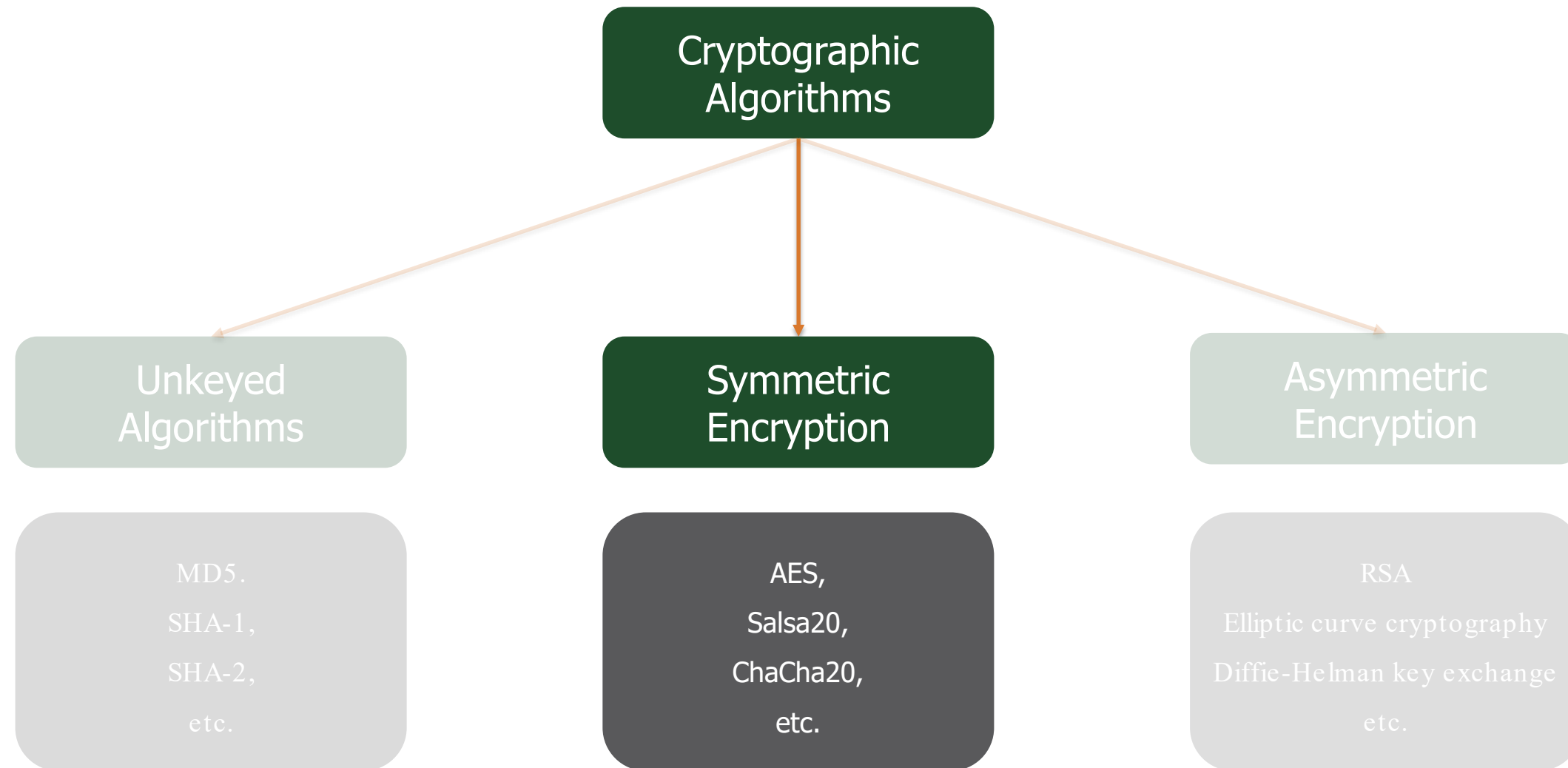
FLEXCAN Module

- Version 2.0B
 - Standard data and remote frames
 - Extended data and remote frames
 - Zero to eight bytes data length
 - Programmable bit rate up to 1 Mb/sec
 - Content-related addressing
- Flexible Mailboxes of eight bytes data length
- Each Mailbox is configurable as Rx or Tx, all supporting standard and extended messages •
- Individual Rx Mask Registers per Mailbox
- Full featured Rx FIFO with storage capacity for 6 frames and internal pointer handling

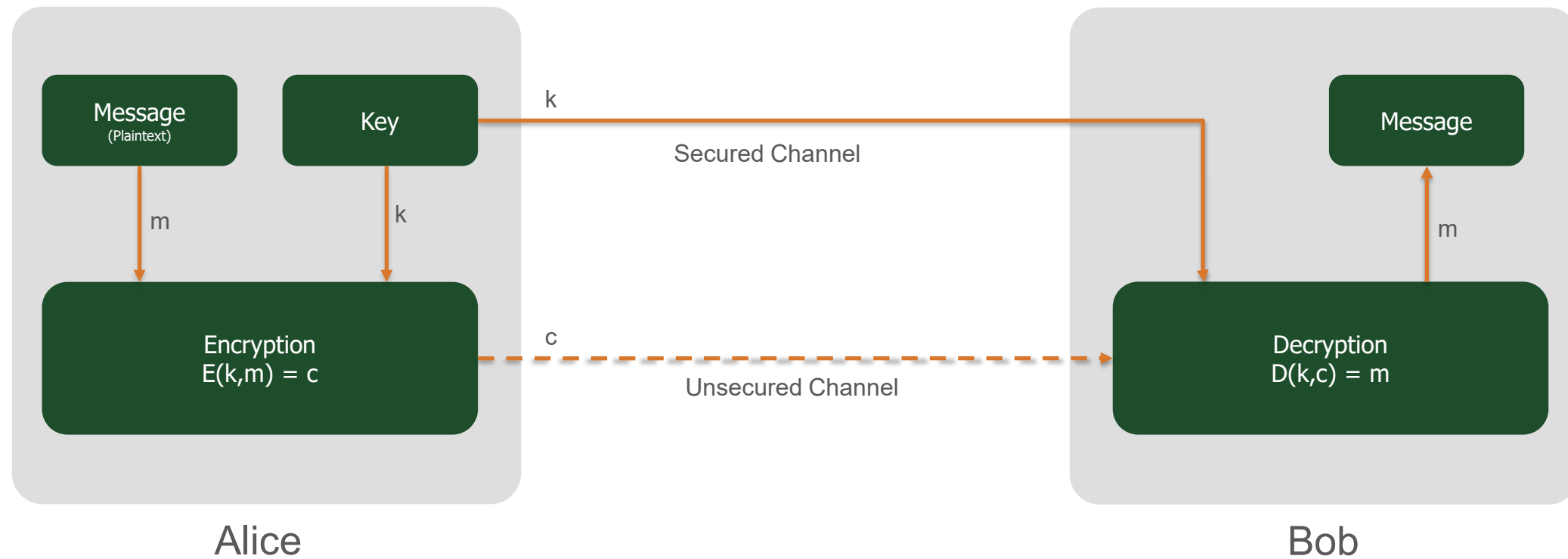
Basics of Cryptography



Basics of Cryptography



Symmetric Encryption



Symmetric Encryption

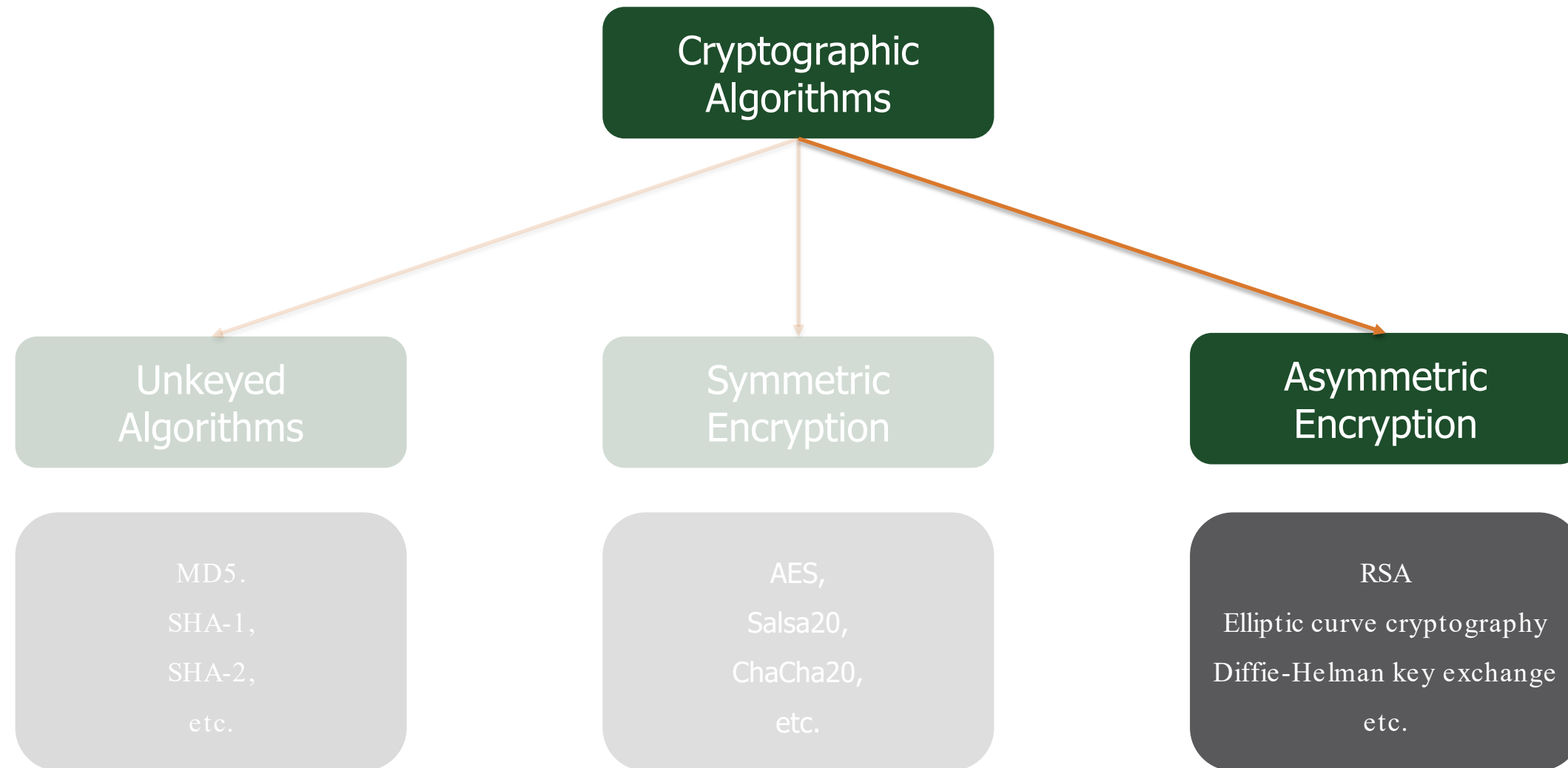
Advantages

- Simple
- Fast
- Less computing resources

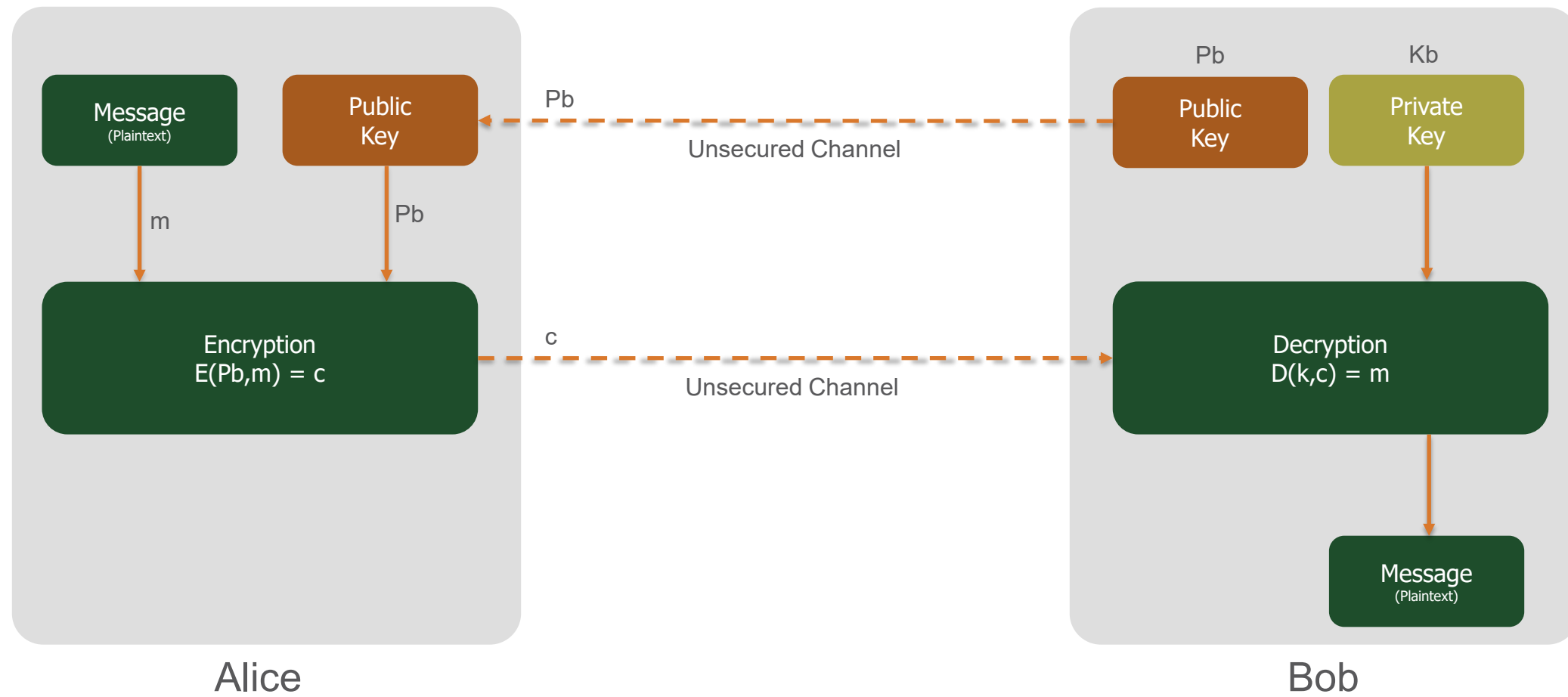
Disadvantages

- Secure channel for secret key exchange
- Multiple keys for multiple clients
- Lacks message authenticity.

Basics of Cryptography



Asymmetric Encryption



Asymmetric Algorithms

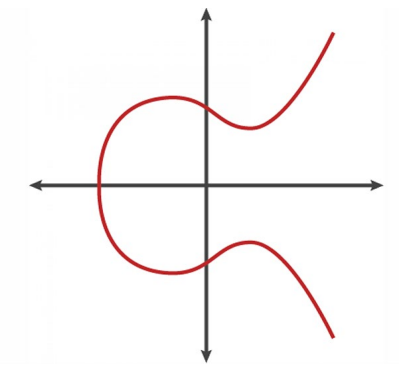
Advantages

- Confidentiality/access control
- Message authentication
- Tamper detection

Disadvantages

- Public keys should/must be authenticated
- Slower than symmetric encryption
- Require more computing resources

Elliptic Curve Cryptography

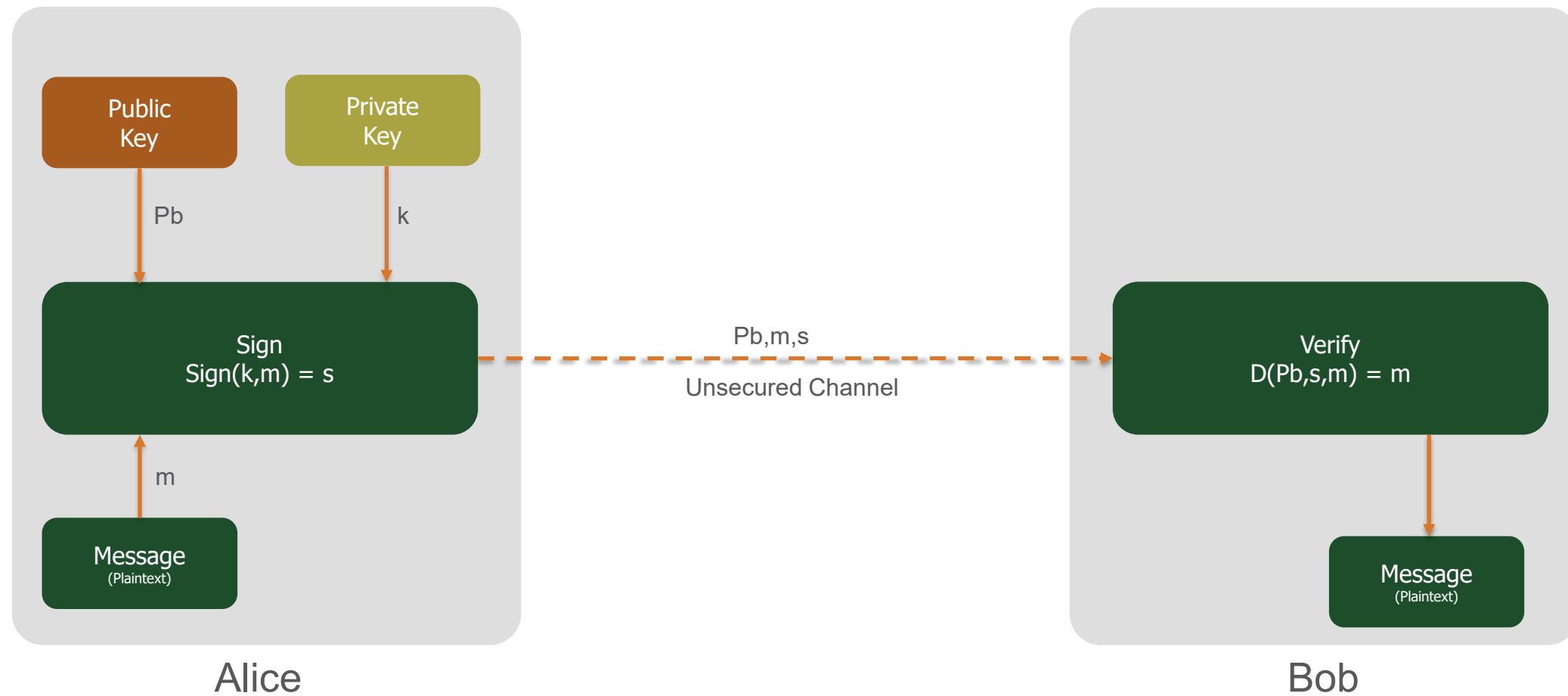


- Public key cryptography is based on special mathematical functions that are relatively easy to calculate but very difficult to invert.
- According to NIST SP 800-57, a 224-bit ECC key used for digital signatures is equivalent in strength to RSA-2048.
- Advantages
 - Better Security strength efficiency compared to RSA
 - Mathematics is more complex than RSA
 - Smaller Key size
- Disadvantages
 - Complex Implementations

Cryptographic strength comparison

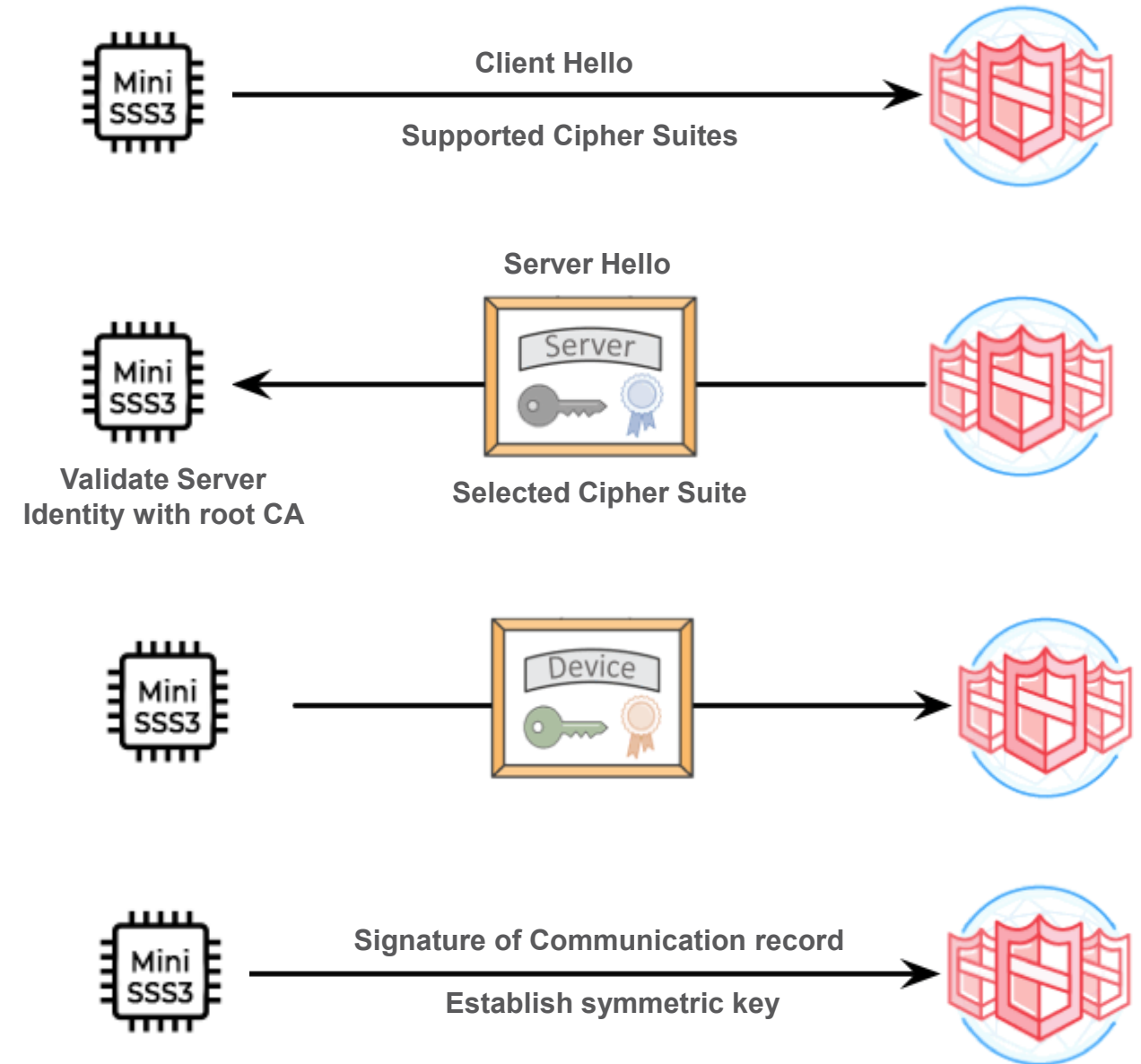
Symmetric Key Size	RSA	ECC
80	1024	160
112	2048	224
128	3072	256
192	7680	384

Elliptic Curve Digital Signature Algorithm



Transport Layer Security

- BearSSL is an implementation of the SSL/TLS protocol ([RFC 5246](https://tools.ietf.org/html/rfc5246)) written in C
- SSL/TLS has many defined cipher suites and extensions. BearSSL implements most of them.
- ArduinoBearSSL



Wireshark Trace

No.	Time	Source	Destination	Protocol	Length	Info
3166	242.884824	192.168.137.95	52.8.204.58	TLSv1.2	303	Client Hello
3168	242.934240	52.8.204.58	192.168.137.95	TLSv1.2	150	Server Hello
3172	242.935594	52.8.204.58	192.168.137.95	TLSv1.2	675	Certificate
3173	242.935639	52.8.204.58	192.168.137.95	TLSv1.2	438	Server Key Exchange, Certificate Request, Server Hello Done
3185	242.984991	192.168.137.95	52.8.204.58	TLSv1.2	571	[Certificate Fragment]
3189	243.150482	192.168.137.95	52.8.204.58	TLSv1.2	375	Certificate, Client Key Exchange, Certificate Verify
3190	243.150482	192.168.137.95	52.8.204.58	TLSv1.2	60	Change Cipher Spec
3191	243.150482	192.168.137.95	52.8.204.58	TLSv1.2	99	Encrypted Handshake Message
3195	243.198730	52.8.204.58	192.168.137.95	TLSv1.2	105	Change Cipher Spec, Encrypted Handshake Message
3198	243.199821	192.168.137.95	52.8.204.58	TLSv1.2	113	Application Data
3200	243.295465	52.8.204.58	192.168.137.95	TLSv1.2	87	Application Data
3203	243.296659	192.168.137.95	52.8.204.58	TLSv1.2	106	Application Data
3205	243.394629	52.8.204.58	192.168.137.95	TLSv1.2	88	Application Data
3208	243.395947	192.168.137.95	52.8.204.58	TLSv1.2	100	Application Data
3210	243.493957	52.8.204.58	192.168.137.95	TLSv1.2	88	Application Data
3213	243.495249	192.168.137.95	52.8.204.58	TLSv1.2	100	Application Data
3215	243.593640	52.8.204.58	192.168.137.95	TLSv1.2	88	Application Data
3218	243.594915	192.168.137.95	52.8.204.58	TLSv1.2	101	Application Data
3220	243.700346	52.8.204.58	192.168.137.95	TLSv1.2	88	Application Data
3223	243.701535	192.168.137.95	52.8.204.58	TLSv1.2	133	Application Data
3225	243.797950	52.8.204.58	192.168.137.95	TLSv1.2	88	Application Data
3228	243.799145	192.168.137.95	52.8.204.58	TLSv1.2	136	Application Data
3230	243.894933	52.8.204.58	192.168.137.95	TLSv1.2	88	Application Data

ECDSA

- ▼ Handshake Protocol: Certificate
 - Handshake Type: Certificate (11)
 - Length: 674
 - Certificates Length: 671
 - Certificates (671 bytes)
- ▼ Handshake Protocol: Client Key Exchange
 - Handshake Type: Client Key Exchange (16)
 - Length: 66
 - ▼ EC Diffie-Hellman Client Params
 - Pubkey Length: 65
 - Pubkey: 04ed9933301ba49136df6259aba5060f0d31e07ace65073672898775561d53c04edbf68...
- ▼ Handshake Protocol: Certificate Verify
 - Handshake Type: Certificate Verify (15)
 - Length: 76
 - Signature Algorithm: ecdsa_secp256r1_sha256 (0x0403)
 - Signature length: 72
 - Signature: 3046022100cb4c7ff7011efa2f3666fb7cd184ae0b1e8c22a9e5b950a01193779d1da92e...

ECDSA Sign

```
#include <ArduinoECCX08.h>

byte signature[64];

byte message[32] = {
  0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
  0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
  0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
  0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F
};

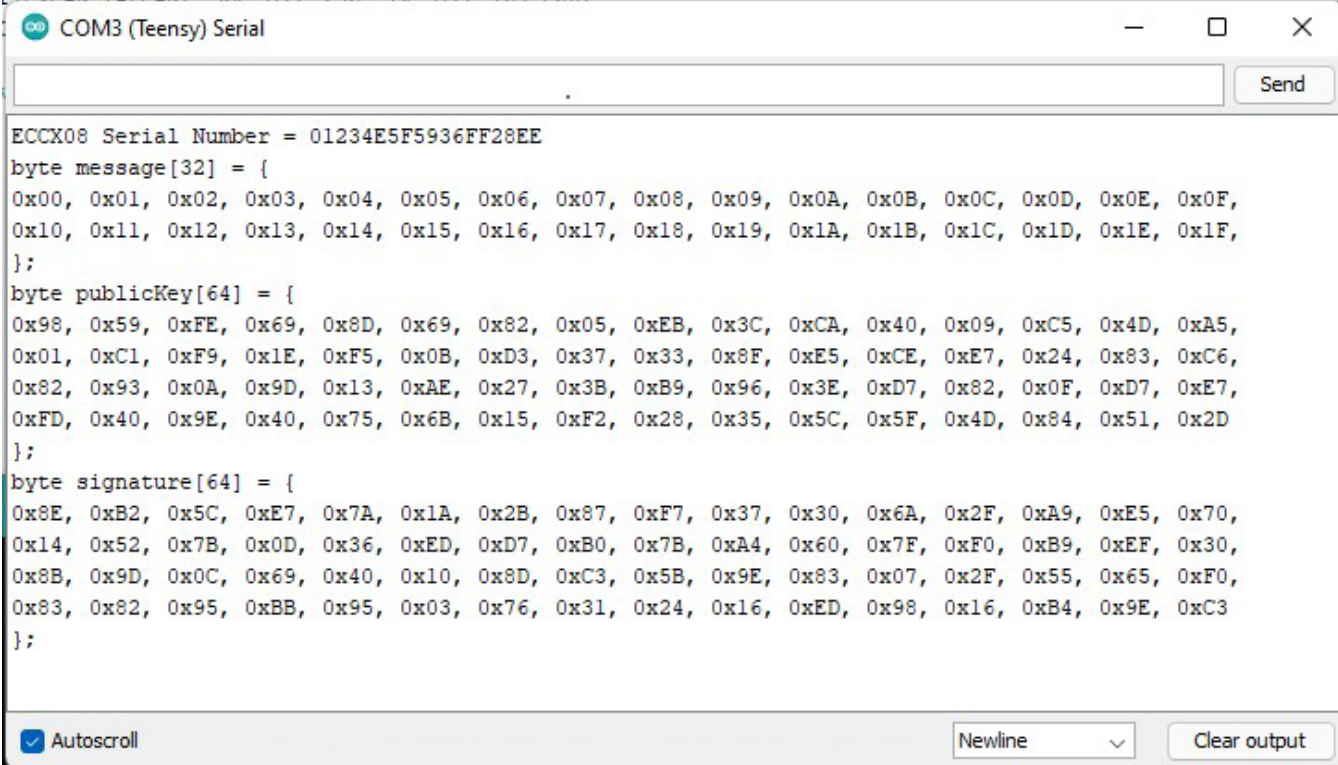
byte publicKey[64];
void setup()
{
  ECCX08.begin();
  ECCX08.ecSign(0, message, signature);

  ECCX08.generatePublicKey(0, publicKey);

  printMessage();

  printPublicKey();

  printSignature();
}
```



```
COM3 (Teensy) Serial

ECCX08 Serial Number = 01234E5F5936FF28EE
byte message[32] = {
  0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
  0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
};
byte publicKey[64] = {
  0x98, 0x59, 0xFE, 0x69, 0x8D, 0x69, 0x82, 0x05, 0xEB, 0x3C, 0xCA, 0x40, 0x09, 0xC5, 0x4D, 0xA5,
  0x01, 0xC1, 0xF9, 0x1E, 0xF5, 0x0B, 0xD3, 0x37, 0x33, 0x8F, 0xE5, 0xCE, 0xE7, 0x24, 0x83, 0xC6,
  0x82, 0x93, 0x0A, 0x9D, 0x13, 0xAE, 0x27, 0x3B, 0xB9, 0x96, 0x3E, 0xD7, 0x82, 0x0F, 0xD7, 0xE7,
  0xFD, 0x40, 0x9E, 0x40, 0x75, 0x6B, 0x15, 0xF2, 0x28, 0x35, 0x5C, 0x5F, 0x4D, 0x84, 0x51, 0x2D
};
byte signature[64] = {
  0x8E, 0xB2, 0x5C, 0xE7, 0x7A, 0x1A, 0x2B, 0x87, 0xF7, 0x37, 0x30, 0x6A, 0x2F, 0xA9, 0xE5, 0x70,
  0x14, 0x52, 0x7B, 0x0D, 0x36, 0xED, 0xD7, 0xB0, 0x7B, 0xA4, 0x60, 0x7F, 0xF0, 0xB9, 0xEF, 0x30,
  0x8B, 0x9D, 0x0C, 0x69, 0x40, 0x10, 0x8D, 0xC3, 0x5B, 0x9E, 0x83, 0x07, 0x2F, 0x55, 0x65, 0xF0,
  0x83, 0x82, 0x95, 0xBB, 0x95, 0x03, 0x76, 0x31, 0x24, 0x16, 0xED, 0x98, 0x16, 0xB4, 0x9E, 0xC3
};

Autoscroll Newline Clear output
```


ECDSA verify

```
#include <ArduinoECCX08.h>

byte message[32] = {
  0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
  0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F};

byte publicKey[64] = {0x98, 0x59, 0xFE, 0x69, 0x8D, 0x69, 0x82, 0x05, 0xEB, 0x3C, 0xCA, 0x40,
  0x09, 0xC5, 0x4D, 0xA5, 0x01, 0xC1, 0xF9, 0x1E, 0xF5, 0x0B, 0xD3, 0x37, 0x33, 0x8F, 0xE5, 0xCE,
  0xE7, 0x24, 0x83, 0xC6, 0x82, 0x93, 0x0A, 0x9D, 0x13, 0xAE, 0x27, 0x3B, 0xB9, 0x96, 0x3E, 0xD7,
  0x82, 0x0F, 0xD7, 0xE7, 0xFD, 0x40, 0x9E, 0x40, 0x75, 0x6B, 0x15, 0xF2, 0x28, 0x35, 0x5C, 0x5F,
  0x4D, 0x84, 0x51, 0x2D };

byte signature[64] = {0x8E, 0xB2, 0x5C, 0xE7, 0x7A, 0x1A, 0x2B, 0x87, 0xF7, 0x37, 0x30, 0x6A,
  0x2F, 0xA9, 0xE5, 0x70, 0x14, 0x52, 0x7B, 0x0D, 0x36, 0xED, 0xD7, 0xB0, 0x7B, 0xA4, 0x60, 0x7F,
  0xF0, 0xB9, 0xEF, 0x30, 0x8B, 0x9D, 0x0C, 0x69, 0x40, 0x10, 0x8D, 0xC3, 0x5B, 0x9E, 0x83, 0x07,
  0x2F, 0x55, 0x65, 0xF0, 0x83, 0x82, 0x95, 0xBB, 0x95, 0x03, 0x76, 0x31, 0x24, 0x16, 0xED, 0x98,
  0x16, 0xB4, 0x9E, 0xC3 };

void setup(){
  ECCX08.begin(0x35);

  if(ECCX08.ecdsaVerify(message, signature, publicKey)) Serial.println("Signature Verified");
  else Serial.println("Signature Failed");
}
```

