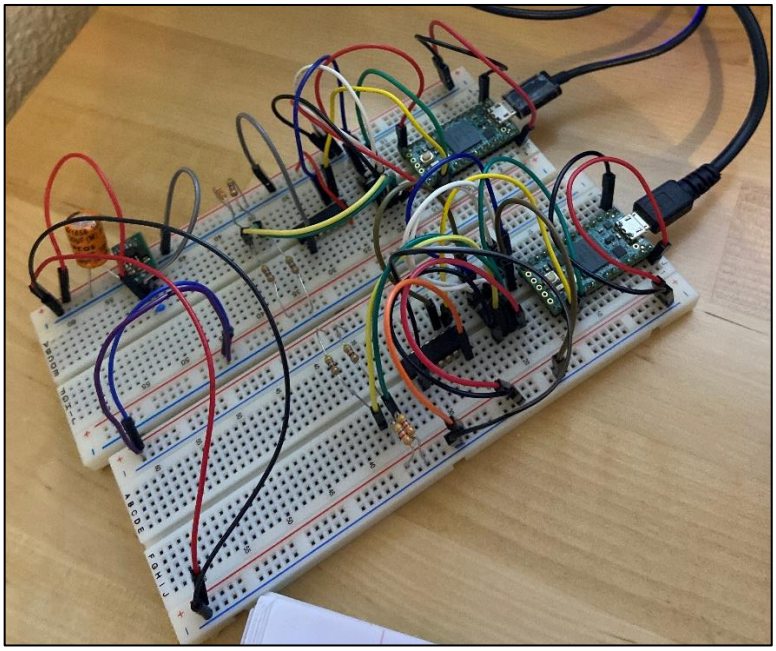
Final Report
Applied Cryptography in Legacy Vehicle Networks



Cover Image: Two Connected Demo ECUs

David Nnaji
ENGR 580A: Secure Vehicle and Industrial Networking
Due December 17th, 2020

1. Introduction

Commercial trucks and trailers rely on numerous microcomputer systems called electronic control units (ECU) to exchange system information. Much of this information pertains to the operation of the vehicle and its components (i.e. performance, maintenance, and diagnostics). In order for modules to communicate effectively between each other, manufacturers have agreed on a standard practice to promote hardware and software compatibility.

Although standardization of network communication has numerous benefits overall, it has unintentionally allowed potential cybersecurity risks to become broadly applicable to commercial vehicles. In fact, academic and industry research has already demonstrated network vulnerabilities in commercial vehicles that can allow an actor to disrupt regular in-vehicle communications and functions.

While the majority of data exchanged takes place between ECUs on the truck network, semi-trailers also generate information that is routinely monitored. However, most semi-trailers on the road rely on a legacy communications standard that has been stabilized. In addition to the aforementioned vulnerabilities, the legacy communication standard has its own set of unique security challenges. As academic researchers and industry engineers work on improving the security posture of commercial vehicle communications systems, there is a need for novel demonstrations of secure communications between modules still based on legacy standards.

2. Background

Related Work

Currently, investigations into commercial vehicle network vulnerabilities, countermeasures, and other related topics are numerous, broad, and active. For example, one team of researchers used empirical experiments to demonstrate that ECMs could be disrupted through request overloads and false request-to-send (RTS) messages [1]. Recent research by the NMFTA has exposed a vulnerability within trailer power line communications (PLC) signals which indicate the possibility of remote network surveillance at distances of up to 6 to 8 feet [2][3]. A team of Michigan-based researchers developed a clock-based intrusion detection system called CIDS to identify compromised modules on the network [4].

Despite the wealth of publications available, suggested mitigations for modules based on legacy protocol are rarely mentioned or thoroughly discussed. However, many mitigations in the context of modern systems could be extended or adapted for these modules.

SAE J1939

Embedded communications in modern commercial vehicles are largely defined by the SAE J1939 network protocol [5]. As shown in **Fig 1**. The SAE J1939 network is modeled on the ISO/OSI protocol stack. A single message from and ECU starts at the application layer traverses down this protocol stack until it reaches the CAN-based physical layer. It is at this lower layer that messages are framed, exchanged, and ultimately traverse back up the protocol stack of another ECU to be unpacked and interpreted.

Application	J1939/71 J1939/73
Presentation Session	Null
Transport	J1939/21
Network	J1939/31
Data Link	J1939/21
Physical	J1939/11 J1939/12 J1939/14 J1939/15
OSI Reference Model	**Modern SAE Model**

Figure 1: J1939 Relationship to ISO/OSI Model

A summary of notable features is provided here to avoid needless descriptions of the many documents that define SAE J1939:

Network Speed: Most often 250 kbit/s or 500 kbit/s
Network Topology: Multi-master serial bus connected via twisted wire pair with terminating resistors. Based on CAN Protocol.
Message Format: 29-bit Identifier and 8-byte data payload. As shown in **Fig. 2**, the identifier is broken into priority, reserved, data page, PDU format, PDU specific, and source address fields.

Priority	Reserved	Data page	PDU Format	PDU Specific	Source Address
3 bits	1 bit	1 bit	8 bits	8bits	8 bits

Figure 2: Structure of 29-bit ID

Address Claiming: Most addresses are pre-assigned to an ECU depending on its function by SAE 1939. However, an address claiming procedure has also been defined for ECU with functions beyond the scope of the standard.
Transmission: All messages sent by an ECU are visible to every node on the network. However, it is up to the receiving ECU to determine if a message can be ignored.
Transport Protocol: The procedure in place for ECUs wishing to transmit payloads larger than 8-bytes. The payload is segmented by the transmitter and put back together by the receiving ECU.

SAE J1708 and J1587

Embedded communication in commercial vehicles that predate the publication of SAE J1939 adhere to SAE J1708 and SAE J1587. These standards were stabilized in 2016 and their relationship to the ISO/OSI reference model is shown in **Fig. 3.**

OSI Reference Model	SAE Model
Application Presentation	J1587
Session Transport Network	Null
Data Link	J1587/J1708
Physical	J1708

Figure 3: Legacy Standard Relationship to ISO/OSI Model

Notable features of SAE J1708/1587 include:

Network Speed: Most often 9600 kbit/s
Network Topology: Multi-master serial bus connected via twisted wire pair. Based on RS-485 Protocol.
Message Format: As shown in **Fig. 4**. 1-byte Message Identifier (MID), 1-byte Parameter Identifier (PID), 18-byte (max) data payload, and 1-byte Checksum (CHK).

MID	PID	Data	Checksum
1 byte	1 byte	18 byte (max)	1 byte

Figure 4: Standard Structure of J1587 Message

Address Claiming: Nearly all MIDs are pre-assigned to an ECU depending on its function. A dynamic address claim exists only for trailer gateway devices.
Transmission: All messages sent by an ECU are visible to every node on the network. However, it is up to the receiving ECU to determine if a message can be ignored.
Transport Protocol: The procedure in place for ECUs wishing to transmit payloads larger than 8-bytes. The payload is segmented by the transmitter and put back together by the receiving ECU.

Feature Definition

Description: A J1587 compliant approach for generating cryptographic keys, exchanging public keys over the network, and encrypting sensitive parameters with them. The approach should be demonstrated in-part on an analogous J1708 network.

Threat Analysis and Risk Assessment

In the event of system failure or malfunction, the following stakeholders are likely to be affected: driver, technician, law enforcement, researcher, general public, fleet operator and telematics. **Table 1**, provides a list of risks that could be attributed directly to deployment of a cryptographic approach. It is broken into columns containing the risk ID, description, and key stakeholders that would be affected.

Risk ID	Risk	Key stakeholder(s)
a	An ECU cannot obtain needed parameters	Driver, Telematics
b	Bus overload	Driver, General Public
c	Critical forensic data cannot be captured by ECU	Law Enforcement
d	Critical forensic data cannot be decrypted by law enforcement	Law Enforcement
e	Diagnostic data cannot be obtained	Technician
f	ECU failure leads to crash	Driver, Fleet Operator, General Public
g	Key management become unwieldy	OEM, Technician
h	Increased power consumption by ECU	OEM
i	Increased memory consumption by ECU	OEM
j	Private keys are extracted	OEM
k	Encrypted session is deciphered	OEM, Researcher
m	Increased message latency	Telematics, Driver
n	Encrypted session is recorded	Fleet Operator, Telematics, Researcher

Table 1: Table of Risks

These risks were then plotted on the matrix shown in **Fig. 5**. depending on their expected likelihood and impact.

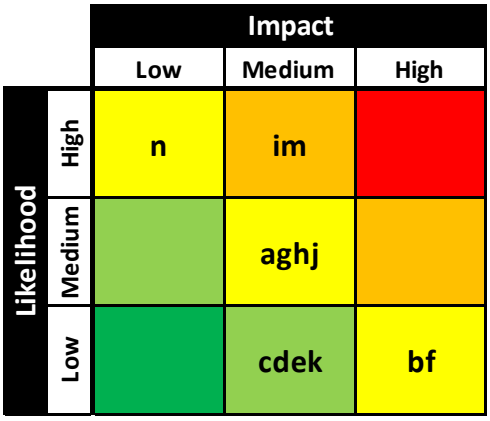


Figure X: Risk Matrix

The matrix shows that risk "I" and "M" were deemed the highest threats. An attack tree for risk I shows that the potential storage of numerous keys and larger messages will lead to increased memory consumption (**Fig. 6**). To mitigate this risk, it may be necessary to increase on board memory or to use a hardware security module capable of managing many keys. Leveraging algorithms that use small keys is also recommended. An attack tree for risk m would show that larger dependency on the transport protocol for sending keys and cryptographic functions performed by an ECU could ultimately lead to increased message latency.

Cybersecurity Goals and Concept

1. Decrease ECU memory consumption by using 32-byte public keys generated with elliptic curve cryptography.
2. Decrease latency by incorporating a hardware security module for cryptographic functions.

System Design Process

A systems "V" diagram for the project with the anticipated steps and milestones is included in the appendices.

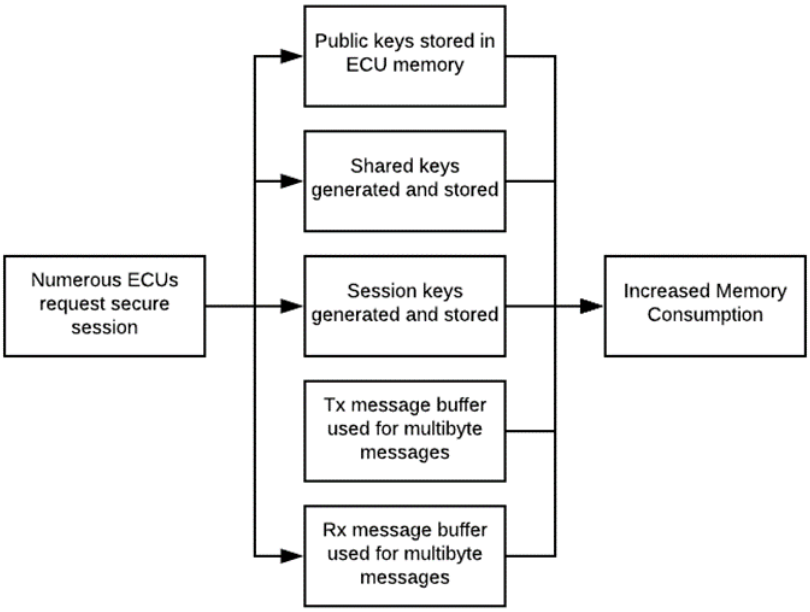


Figure 6: Attack Tree for "Risk I"

3. Requirements

The system requirements evolved as the project was further defined. The initial design alternative incorporated J1939 hardware, a software GUI, and other components. However, due to challenges with J1708 hardware leading to a diminished timeline, the following requirements we ultimately selected as the basis for the final design alternative.

Functional System Requirements:

SR1 System hardware shall comply with J1708 specifications.
SR2 Total hardware cost shall not exceed $200.
SR3 System communication shall comply with SAE J1587 recommended practice.
SR4 System software shall be well-documented and easy to read.

Functional Cybersecurity Requirements:

CR1 System shall integrate a hardware security module.
CR2 System shall demonstrate key-generation with HSM
CR3 System shall demonstrate key-exchange over J1587

4. Specific Project Focus

This project aimed to deliver a solution for cybersecurity goals 1 and 2 mentioned earlier in the report. To accomplish this, two prototype ECUs were made using identical hardware. The primary components of the ECUs were a Teensy 4.0, an RS485 chip, hex-inverter chip, and a 5V regulator. These were integrated on a breadboard according to the wiring diagram provided in

the appendices. A regulator allowed testing with an actual ECU and the Teensy allowed the device to be programmed using a C/C++ variant in the Arduino/Teensyduino IDE.

The ATECC608A was selected as the hardware security module. The Microchip TrustAuth platform development kit was used to interface with the ATECC608A via python. A diagram of the final hardware setup is provided in **Fig. 7**.

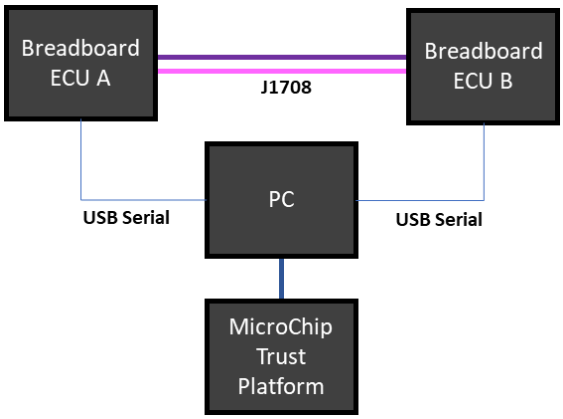


Fig 7: Final Hardware Setup

5. Implementation

Overall Goal

With suitable hardware in place, the two demo ECUs were intended to facilitate a demonstration of key generation, key exchange, and message encryption using the procedure shown in the appendices. This procedure produced the following deliverables:

1. One C/C++ library with useful functions for reading, and transmitting J1587 messages.
2. Two C/C++ sketches used to flash each demo ECU.
3. Two Jupyter Notebooks to generate ECC key pairs, and perform cryptographic functions using HSM and python cryptography libraries.

Completed Source Code

The source code that is functional and complete is the C/C++ library J1708.h and two C/C++ sketches used to flash each demo ECU. These files are available in the final submission. A general description of each function is provided in the README file. J1708Listen(), parseJ1708(), J1708Rx(), J1708Tx(), and J1708TransportTx() are among the most important functions. J1708Rx() and J1708Tx() functions were adapted from code originally written by Dr. Jeremy Daily [8].

The script written for the ECU allows it to effectively listen to messages on the serial bus, parse messages that pertain to it, queue a response, and send messages. A functional description of the code is provided in the appendices.

Unfinished Tasks

Two Jupyter Notebooks to generate ECC key pairs, and perform cryptographic functions using HSM and python cryptography libraries were not completed before the scheduled project completion.

6. Testing

SR1: **Satisfied!**

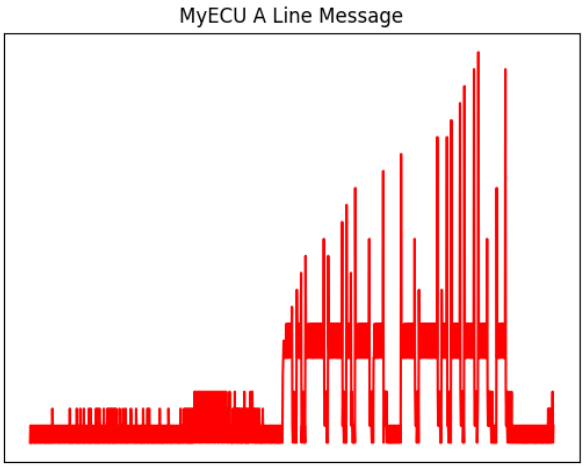


Fig 8: Oscilloscope Capture of RS-485 Pin A Output from Demo ECU

Despite the omission of two resistors used for signal filtering, the hardware setup for the demo ECUs satisfies SR1. This was tested by comparing an oscilloscope trace of the J7108 output signal to that of an actual ECM. This result is shown in **Fig. 8**. Additional testing using a Bendix EC60 and Caterpillar C15 module provided further verification of hardware compliance. In this test the demo ECU was connected to the J1708 pins of each ECM. The output messages that were read were then compared to a log capture with the DPA5. The wiring schematic for the demo ECUs is located in the appendices.

SR2: **Satisfied!**

A breakdown of the hardware costs to produce the two ECUs is provided in **Table 2**. The total hardware cost is roughly $76.11 which satisfies SR2.

	Item Price	Quantity	Sub Total
Teensy 4.0	19.95	2	39.9
Jumper Wire Bundle	1.95	1	1.95
Breadboard	6.95	2	13.9
Resistor Kit	8	1	8
(oki 78sr) V+ Regulator	3.7	1	3.7
220nF Capacitors	1	1	1
RS485	3.18	2	6.36
Hex Inverter	0.65	2	1.3
		Total:	**$ 76.11**

Table 2: Itemized Hardware Cost

SR3 Satisfied!
The key metrics that were used to determine J1587 were as follows:

1. Adherence to message structures defined in J1587 (MID, PID, DATA, CHECKSUM)
2. Avoidance of pre-defined MIDs and PIDs
3. Application of transport protocol to send large data packets
4. Use of request, PID pages, and other mechanism for other complex tasks
5. Appropriate message transmission rate.

The ECU code and J1708 library were written with these metrics in mind and were iteratively tested by printing their output to the USB serial port. The functions within the J1708 library in the final submission have been deemed acceptable.

SR4 Satisfied!
The key metrics that were used to determine that code has had been well written were:

1. The use of modularization
2. Absence of functional repetition
3. Consistent and relevant variable naming
4. Well-commented code

The ECU runtime loop consists of only 1 line of code. Key functions such as receive, transmit, and parse are separate functions that can be called the main ECU. Global variables, flags, buffers, and other data structures are named in relation to their uses and grouped at the top of the library. There are multiple instances where a third-party analyzer can read comments, and uncomment debugging code to analyze functionality.

CR1 Satisfied!
The hardware security module was interfaced using the ACES software.

CR2 Partially Satisfied
The hardware security module was interfaced using the ACES software to generate keys from the ATECC. However, these functions were not fully utilized since the final demonstration was not completed.

CR3 Not Satisfied
A complete run through of the software demonstration procedure was never completed. This was due to prolonged development of TransportTx() function.

7. Vulnerabilities
The most glaring vulnerability this specific application of cryptography to vehicle communication is verification of the public key source. Currently, any node could send its own public key with another MID. One way to minimize this vulnerability is to have a certificate authority. The certificate authority is an entity that issues digital certificates that certify the

ownership of a public key by the certificate owner. Actually, implementing this in a vehicle system would be challenging since any OEM could assume the role of CA.

8. Reflection

My project was ambiguously defined for an extended period of time which ultimately lead to a final result that was somewhat less than what I had hoped for. One of the glaring issues I had was identifying a specific problem I wanted to analyze. Aside from this, I was also learning a lot as my project evolved. Early on, I knew I wanted to use cryptography is some way but I did not have the technical examples to draw from until a lecture example was available. Despite this, I am glad I was challenged and didn't default to an easier project. Looking back, I have gained the following:

1. A better understanding of cryptographic tools, how to use them, and their applications (particularly to vehicle systems).
2. A chance to practice C++ on an actual project outside of Codecademy!
3. Strong understanding of SAE J1708/1587 based systems.

9. Works Consulted

[1] S. Mukherjee, H. Shirazi, I. Ray, J. Daily and R. Gamble. (2016). "Practical DoS Attacks on Embedded Networks in Commercial Vehicles" Internet: www.cs.colostate.edu. Dec. 16, 2020.

[2] K. Cho and K. Shin. (2016). "Fingerprinting Electronic Control Units for Vehicle Intrusion Detection" Internet: www.usenix.org. Dec. 16, 2020.

[3] Cybersecurity & Infrastructure Security Agency. "ICS Advisory (ICSA-20-219-01)".Internet: us-cert.cisa.gov. Dec. 16, 2020.

[3] Cybersecurity & Infrastructure Security Agency. "ICS Advisory (ICSA-20-219-01)".Internet: us-cert.cisa.gov. Dec. 16, 2020.

[4] DEFCONConference. "DEF CON Safe Mode ICS Village - Ben Gardiner - PowerLine Truck Hacking 2TOOLS4PLC4TRUCKS" Internet: www.youtube.com/DEFCONConference. Dec. 16, 2020.

[5] Society of Automotive Engineers. SAE J1939. (2018). Dec. 16, 2020.

[6] Society of Automotive Engineers. SAE J1708. (2016). Dec. 16, 2020.

[7] Society of Automotive Engineers. SAE J1587. (2013). Dec. 16, 2020.

[8] "J1708." Github. Internet: github.com/jeremydaily/J1708 Dec. 17, 2020.

System Level

- Define problem and timeline for project completion
- Obtain system and cybersecurity requirements from TARA, SAE J1708 and J1587, and related work.
- Broadly define approach for demonstrating key generation, exchanging public keys, and encrypting sensitive parameters.

▲ Project Proposal

Hardware Design

- Define system concept
- Define hardware requirements
- based on SAE J1708, course guidelines, and available class resources.
- Select hardware components
- Prototype and integrate on breadboard

Software Design

- Define system concept
- Define software requirements
- based on SAE J1587, course guidelines, and available class resources.
- Write J1708 library for sending, receiving, and parsing network messages. Write Jupyter notebook code.
- Integrate functions and write ECU code.

▲ Project Progress

Hardware Testing

- Test hardware can receive data from an ECU and compare to DPA5 logs.
- Test voltage regulation
- Test hardware allows message transmission

Software Testing

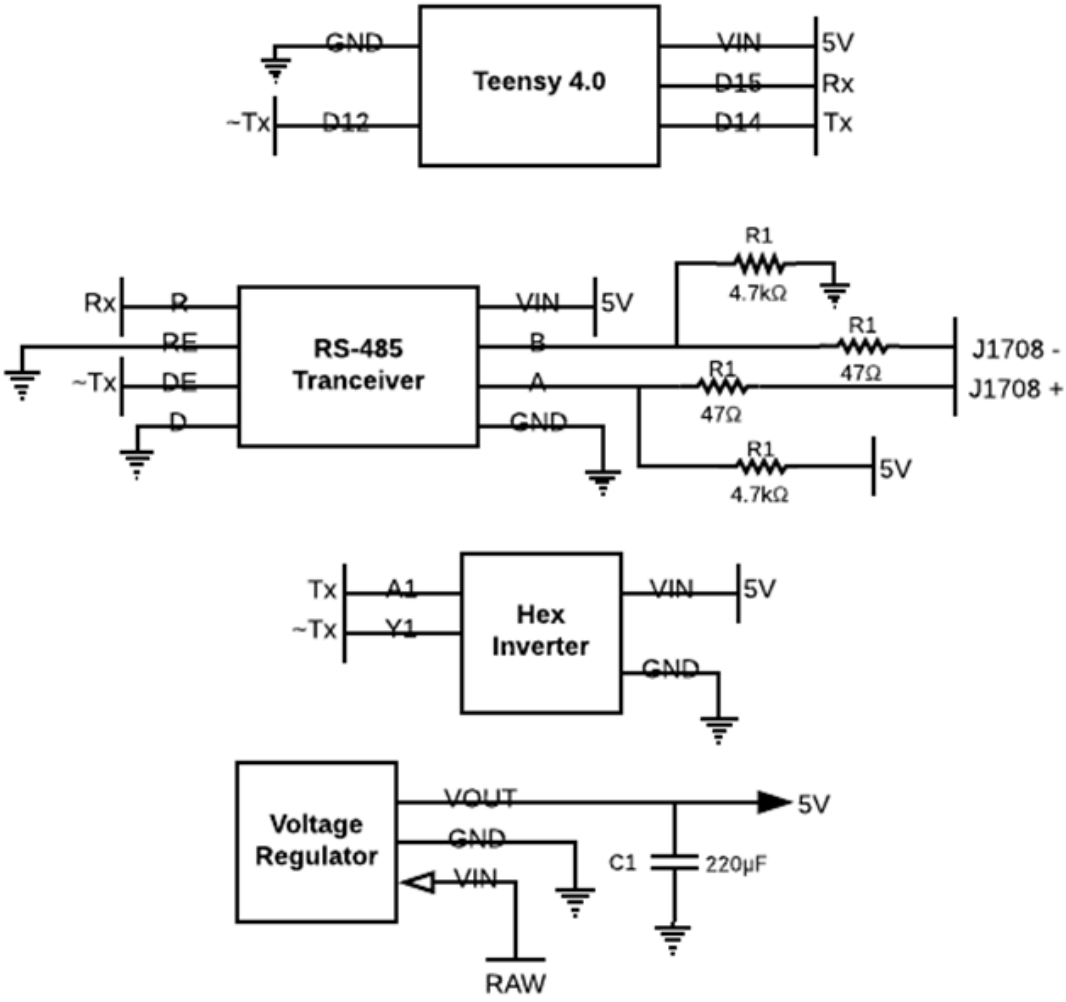
- Test Rx functionality with an ECU
- Test Tx functionality
- Test message parsing and handling functions
- Test transport protocol function
- Test key generation notebook
- Test message verify functions
- Test exchange algorithm
- Test EEPROM storage

▲ Final Presentation + Report

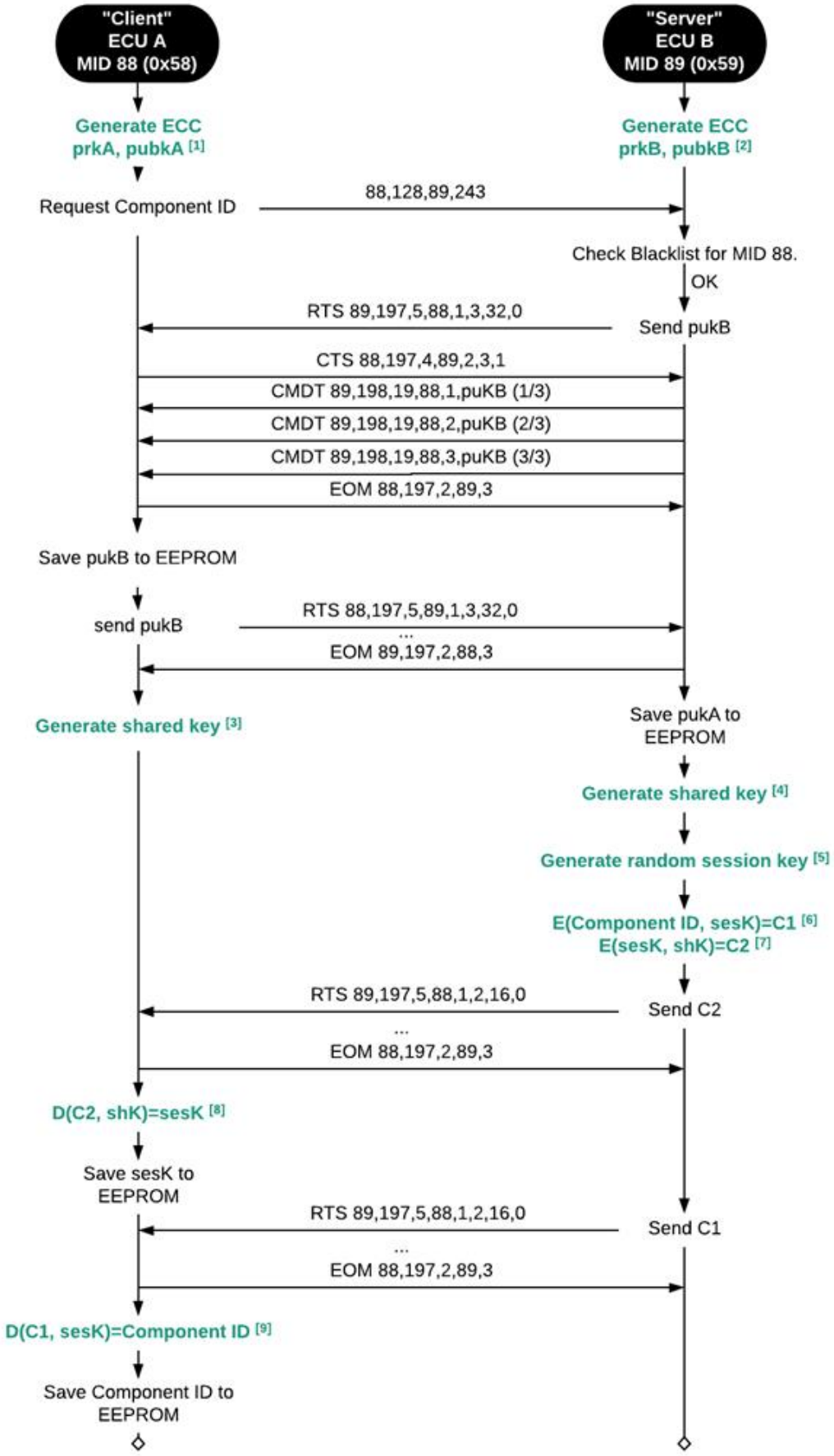
System Level

- Dry run of entire demonstration
- Vulnerability assessment

Wiring Schematic for ECUs



Software Demonstration Procedure



***

Indicates an accompanying procedure to be conducted in Jupyter Notebook

***

*1. ECC key pair generated with python cryptography function *.ed25519() and pre-stored in EEPROM.*

2. ECC key pair generated with HSM and pre-stored in EEPROM.

3. Shared keys generated with .exchange() python function.

4. Shared keys generated with ECDH() HSM function.

5. Use TRNG() function of HSM for genereating 16-byte sesK. Save to temp key slot.

6. Encrypt Component ID "ENGR580ECUB" bytes with sesK. This produces cipher 1 (C1)

7. Encrypt sesK with shK using AES() (16-byte output) (C2). Save (C1 and C2) to EEPROM of ECU B.

8. Use python Cipher() function to decrypt. Manually save to EEPROM.

9. Use fernet functions to decrypt C1 and manually save value to EEPROM.

ECU Runtime Algorithm