5 Testing

Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, power system, or software.

The testing plan should connect the requirements and the design to the adopted test strategy and instruments. In this overarching introduction, given an overview of the testing strategy and your team's overall testing philosophy. Emphasize any unique challenges to testing for your system/design.

In the sections below, describe specific methods for testing. You may include additional types of testing, if applicable to your design. If a particular type of testing is not applicable to your project, you must justify why you are not including it.

When writing your testing planning consider a few guidelines:

- Is our testing plan unique to our project? (It should be)
- Are you testing related to all requirements? For requirements you're not testing (e.g., cost related requirements) can you justify their exclusion?
- Is your testing plan comprehensive?
- When should you be testing? (In most cases, it's early and often, not at the end of the project)

5.1 Unit Testing

What units are being tested? How? Tools?

The mobile security CAN bus has two functions: encryption and decryption. How we have tested both of these is by creating a C program that uses AES encrypt and decrypt functions using the OpenSSL library. What we are specifically designing this algorithm to encrypt and decrypt is the CAN data being sent between two networks, so we have a CAN data log file that we are iterating through as a test subject and ensuring we can encrypt that data line by line, and decrypt it as well. The tools we are using for this are Red Hat Linux, Emacs, C, and Terminal.

5.2 Interface Testing

What are the interfaces in your design? Discuss how the composition of two or more units (interfaces) are being tested. Tools?

The main interfaces of our design are, the vehicle, the vehicle's can system, our CAN bridge device, our code within that device, and the OpenSSL library used within our code. Between our code and the OpenSSL library significant testing and research will be done to ensure that we are using the OpenSSL library correctly. We will be using Red Hat Linux and Emacs to write our code and perform our tests on our code. To test our code within the CAN bridge device we will attach it to a physical system, or a simulated one to validate that we are able to read and output the correct data with the device. Tools for creating a simulated environment will be done through matlab and simulink.

5.3 Integration Testing

What are the critical integration paths in your design? Justification for criticality may come from your requirements. How will they be tested? Tools?

Some critical integration paths include our encryption and decryption methods, and printing them out correctly. Receiving and distributing the keys within the encryption and decryption methods is critical for it to be correct. Some of the physical components that need to work for our integration paths to work would be physical parts of our vehicle (e.g. speedometer, engine, steering wheel, etc.). They'll be tested by checking if the key is correct on each end from our program in a simulation from the software socketCAN. Other tools we are using are OpenGarages, Vehicle ECU, CANSniffer, Linux RedHat, and programming language C.

5.4 System Testing

Describe system level testing strategy. What set of unit tests, interface tests, and integration tests suffice for system level testing? This should be closely tied to the requirements. Tools?

The most significant form of full system testing we will have is running our product on a physical vehicle (tractor). The most obvious way that these tests will be conducted is to confirm that the vehicle still operates as normal while our device is connected. We will also be reading the CAN FD frames to ensure the data is progressing through the system as normal while we also attempt to perform man in the middle attacks on the CAN bus. Other forms of system testing include simulating vehicles through matlab and simulink as well as using basic text files containing CAN data.

5.5 Regression Testing

How are you ensuring that any new additions do not break the old functionality? What implemented critical features do you need to ensure do not break? Is it driven by requirements? Tools?

The list of features that our product needs to have is encryption, decryption, not allow repeats of data, and to deny any Man in The Middle attacks. Right now, we have the encryption and decryption algorithm set, so moving forward, we are going to have to look at the CAN FD frame and see if we can “verify” what is a “good” packet and what is a “bad” packet by utilizing the bit fields of the frame. To ensure that we don’t break our code, we are going to frequently compile and ensure it still successfully encrypts and decrypts the data without a buffer overflow, as we are working with C. The list of tools we are using for regression testing are Linux Red Hat, C, Emacs, and Terminal.

5.6 Acceptance Testing

How will you demonstrate that the design requirements, both functional and non-functional are being met? How would you involve your client in the acceptance testing?

First, we will demo our algorithm with a laptop simulation using socket can, and a demo our client showed us early on in the semester. This demo will prove whether or not our program can survive a Man in The Middle attack, or detect duplicates of data being sent based on freshness values in the CAN FD frame. Once this is a success, our client has mentioned that we will be using our algorithm on a real tractor he will get on campus for us to see how successful we were.

5.7 Security Testing (if applicable)

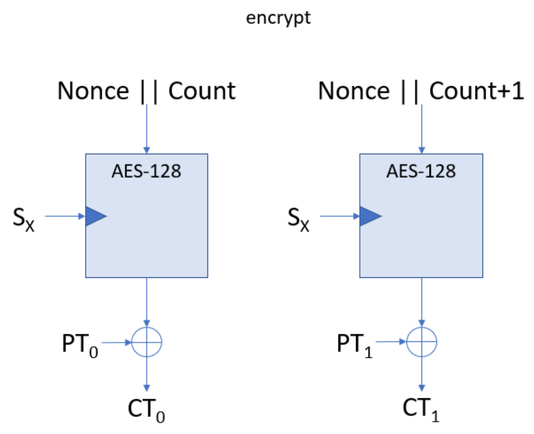
The main testing for the security of our application is going to ensure that the vehicles that are continuously running this code on their CAN busses won’t be susceptible to Man in the Middle attacks, nor will a malicious user be able to send repeats of data sniffed on the can bus network. Right now, we have the encryption and decryption algorithm set, so moving forward, we are going to have to look at the CAN FD frame and see if we can verify what is a “good” packet and what is a “bad” packet by utilizing the bit fields of the frame.

5.8 Results

What are the results of your testing? How do they ensure compliance with the requirements? Include figures and tables to explain your testing process better. A summary narrative concluding that your design is as intended is useful.

The main result we’ll be testing for is that we can send encrypted messages on the CAN network and that they are able to be decrypted when they arrive at their endpoint. For testing this, we can make sure a decrypted message is the same as the original message before encryption. We are planning on using AES128 CTR (counter mode) encryption, which can be seen in the following 2 diagrams:

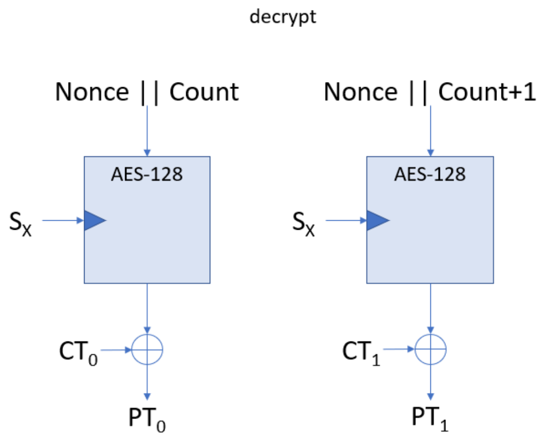
AES-128 CTR



S_X: symmetric key for entity “x”
PT: plaintext
CT: ciphertext

CTR: counter mode

AES-128 CTR



S_X: symmetric key for entity “x”
PT: plaintext
CT: ciphertext

CTR: counter mode

We’re using the OpenSSL library for encryption/decryption so knowing that the algorithms work is a big benefit since designing our own would make for way more testing of all possible inputs. With what we’re using, we have to be more careful of the inputs we’re giving and that they fit the constraints of the algorithms we’re using.

Our results will ensure compliance with the requirements because our tests will identify any problems within our project that need to be addressed so if we continuously test while we develop, we can debug code as we go. Any non code related tests, say on hardware components can be done when testing on the physical CAN network.