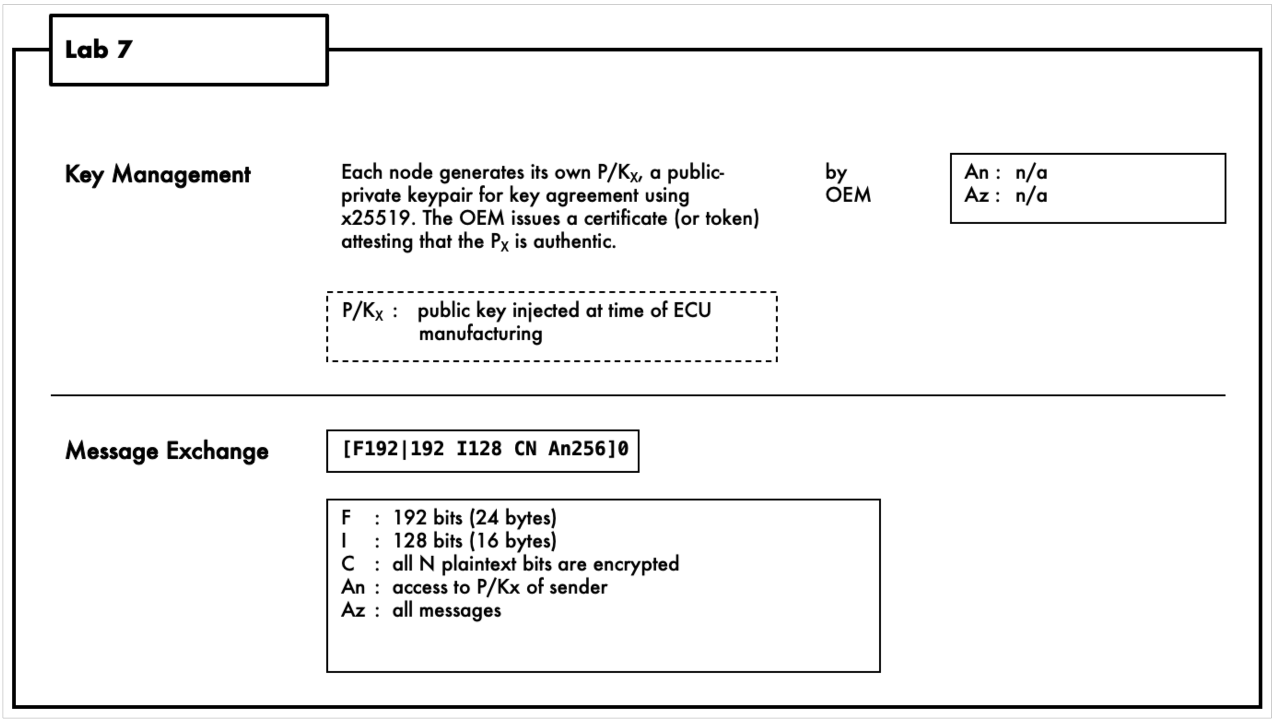

Lab 7

Key Management

Each node generates its own P/K_X, a public-private keypair for key agreement using x25519. The OEM issues a certificate (or token) attesting that the P_X is authentic.

by OEM

An : n/a
Az : n/a

P/K_X : public key injected at time of ECU manufacturing

Message Exchange

```
[F192|192 I128 CN An256]0
```

F	192 bits (24 bytes)
I	128 bits (16 bytes)
C	all N plaintext bits are encrypted
An	access to P/Kx of sender
Az	all messages

Scenario : It turns out there is a very inexpensive (as in free) (and also as in small) cryptography library that is easy to use correctly and hard to use incorrectly. It is probably not a good fit for vehicle networks for at least two reasons:

1) It uses elliptic curve cryptography on every message (which is slower than symmetric cryptography, but perhaps you are securing a message sent only once per second, so speed isn't an issue), and
2) It requires 40 bytes (320 bits) of overhead.

Nonetheless, it is worthwhile understanding this library as it has many uses, and perhaps one day the hardware accelerated versions of these primitives will make them better candidates for more kinds of communication.

The library we are referring to is the NaCl ("salt") library.

The framework above describes the security characteristics of the NaCl "box" concept.

Exercise 1 : [DEFEND] Create a PoC that uses NaCl to secure ground speed.

Hint: you only need to add code to 'adapterA.py' and 'adapterB.py'.

Exercise 2 : Comment on design decision related to nonce. Even though it is huge value (192 bits), how do you prevent a replay attack?