Securing Messages Without a New Format

Leaving individual message unchanged.

Lab5

Pretty Good Security(?): Protect a Stream with Low Overhead

or use existing in-place entropy

Remember

There is no 100% security

Security, like all engineering, involves tradeoffs
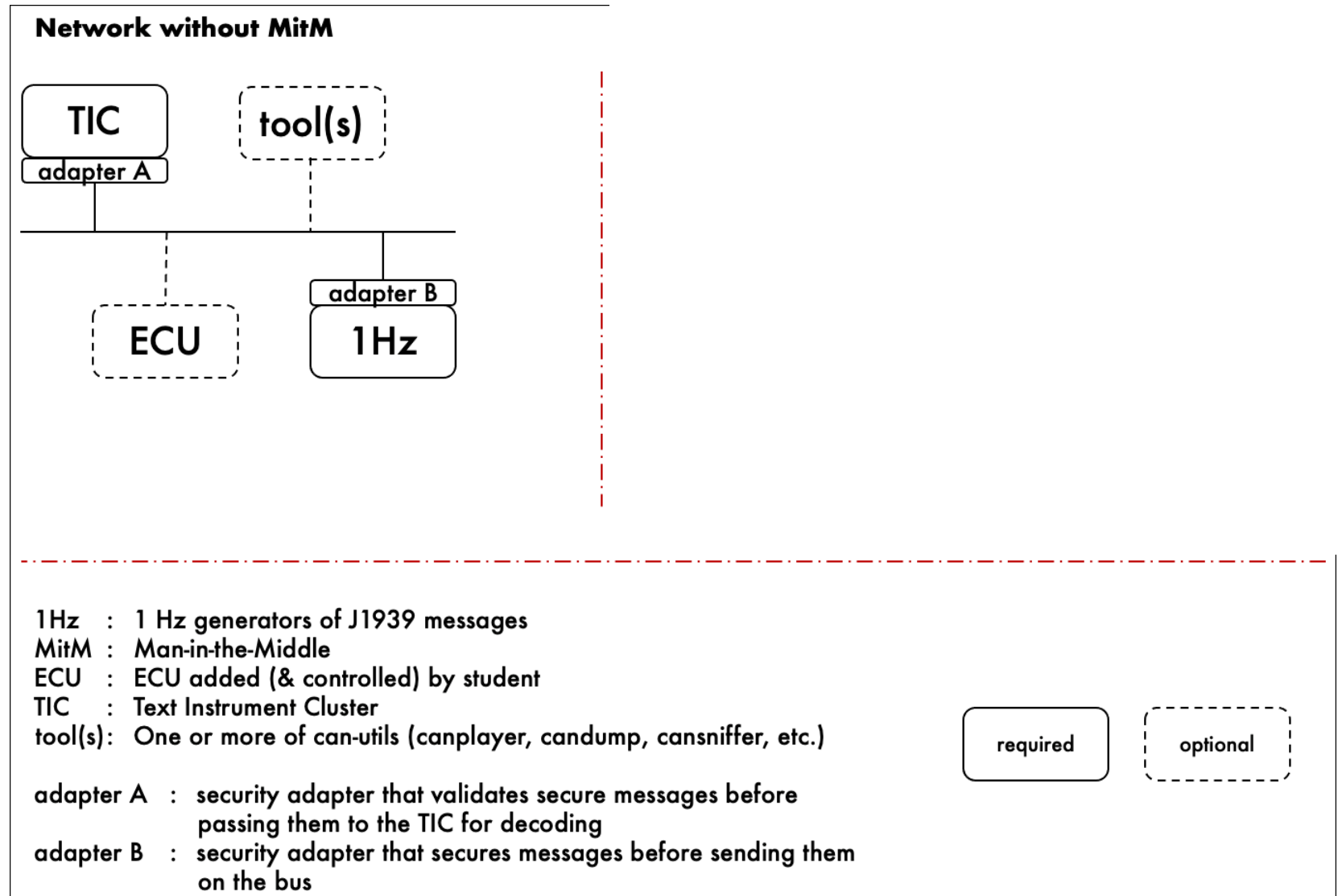
Know what you are trying to secure

The adversary...

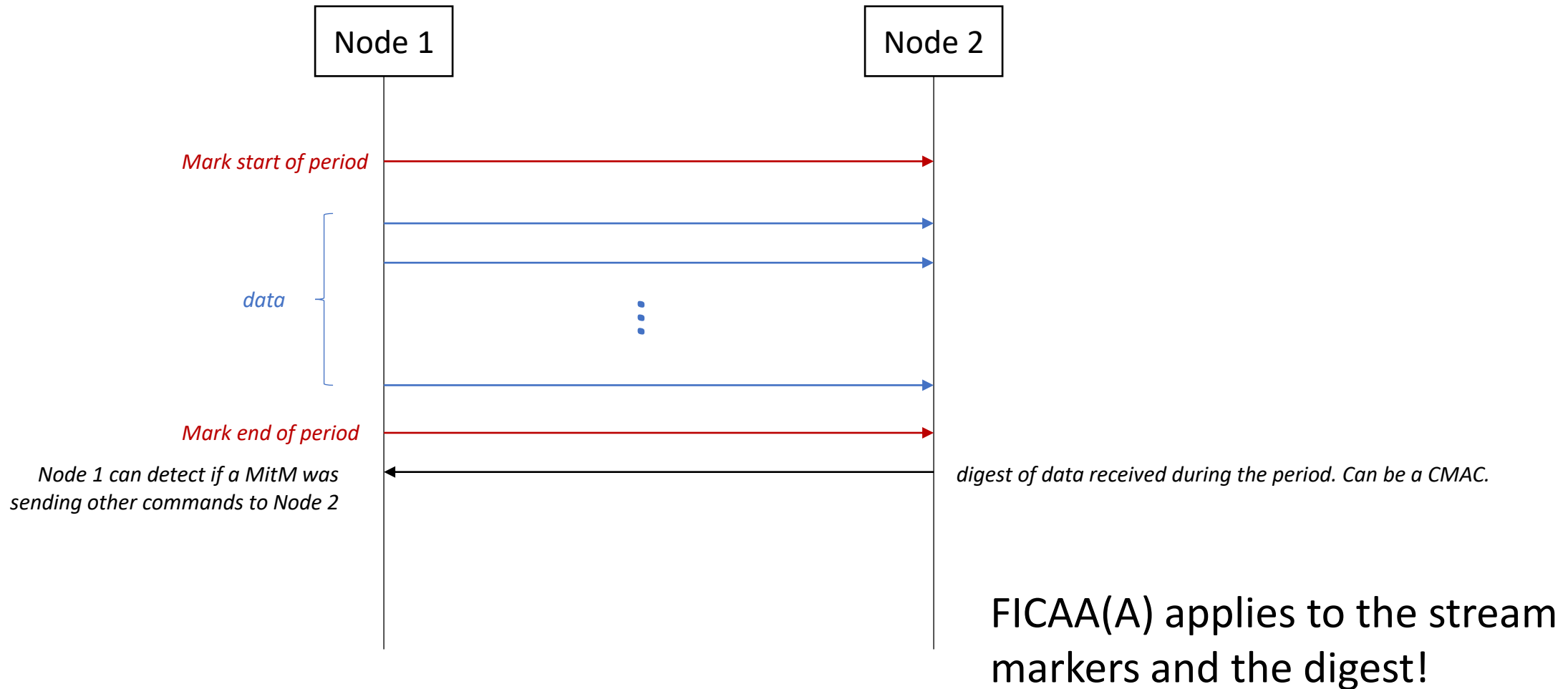**State Sponsored**

Network Configuration

Simple Network for this Lab

Network without MitM

TIC

adapter A

tool(s)

adapter B

ECU

1Hz

1Hz : 1 Hz generators of J1939 messages
MitM : Man-in-the-Middle
ECU : ECU added (& controlled) by student
TIC : Text Instrument Cluster
tool(s): One or more of can-utils (canplayer, candump, cansniffer, etc.)

adapter A : security adapter that validates secure messages before
             passing them to the TIC for decoding
adapter B : security adapter that secures messages before sending them
             on the bus

required

optional

Historical Reference

- Personal experience & academic papers
  - Wrap a stream of data with a CMAC that is validated periodically
  - Often make the simplifying assumption that messages won't be dropped or get out of order
  - Securing CAN Traffic on J1939 Networks
    https://www.researchgate.net/profile/David_Nnaji2/publication/353920347_Securing_CAN_Traffic_on_J1939_Networks/links/6132430cc69a4e487979bef1/Securing-CAN-Traffic-on-J1939-Networks.pdf
    - Intrusion Detection System (IDS)

- Another "free" (as in no overhead) security idea
  - Optimizing CAN Bus Security with In-Place Cryptography 2019-01-0098
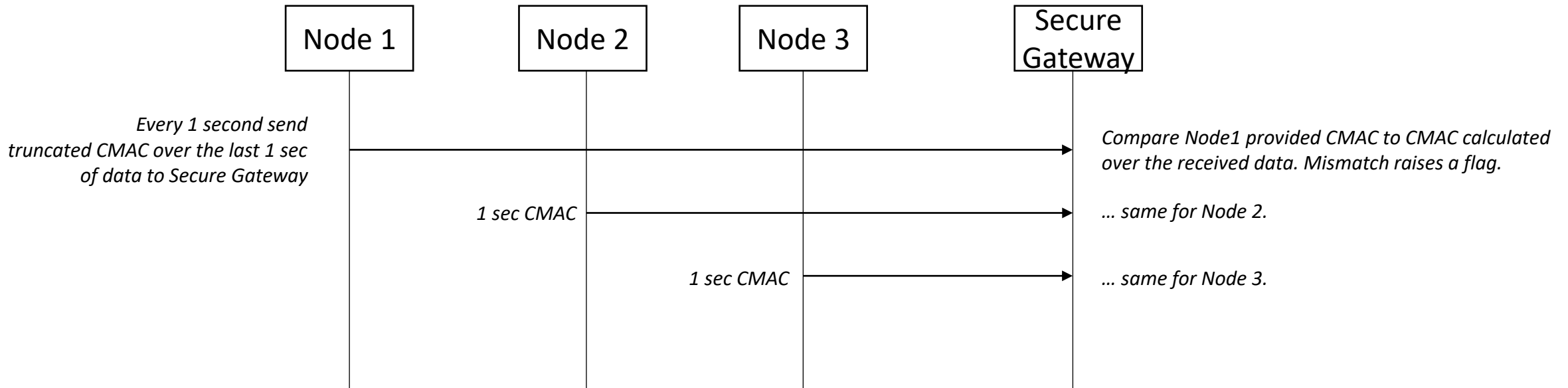 - doi:10.4271/2019-01-0098
  - $35 to download from SAE

Variant 1 : Point-to-Point

Node 1 | Node 2

Mark start of period

data

Mark end of period

Node 1 can detect if a MitM was sending other commands to Node 2

digest of data received during the period. Can be a CMAC.

FICAA(A) applies to the stream markers and the digest!

Variant 1 : Point-to-Point

- FICAA<u>A</u>?
  - This is an example where AVAILABILITY is relevant to the <u>protocol</u>.
  - We need to be careful and intentional about the kind of digest used.
  - Will the protocol fail often?
  - Ex1: Due to out-of-order (could be common, if heavy traffic and transmit queue uses priority instead of FIFO to pick next message to send)
    - In these cases, the application has no way to know what order the messages were actually placed on the bus. Which makes it hard for the application to calculate the digest.
  - Ex2: Due to lost messages (could be common, if bursts of heavy traffic overfill the receive queue of Node2)
    - In these cases, the application at Node2 will calculate a digest that doesn't match Node1.

Variant 2 : Multi-Node

```
                    Node 1      Node 2      Node 3      Secure
                                                        Gateway
```

*Every 1 second send
truncated CMAC over the last 1 sec
of data to Secure Gateway*

*Compare Node1 provided CMAC to CMAC calculated
over the received data. Mismatch raises a flag.*

1 sec CMAC

... same for Node 2.

1 sec CMAC

... same for Node 3.

In this simplified diagram the data messages are not shown

FICAA(A) applies!

From Daily, et. al.

Variant 2 : Multi-Node

- Daily paper, Av (availability) as "false-positives". (emphasis added)

The false positive rate is determined by a ratio of the reported CMAC mismatches to the number of CMACs calculated with no intrusions. Due to the way CMACs are calculated, the timing of the sentinel message containing the CMAC from the CAN Conditioner must provide sufficient gaps such that the same messages are used to calculate the CMAC on the Secure Gateway. If the order of a message is changed, then the CMAC comparison would fail, leading to a false positive. After some initial tuning of the timing, a 20 millisecond gap on either side of the sentinel message lead to a low false positive rate. In one bench test that went for 24 hours, there were 508,762 CMACs that matched and 12 that did not. ==This is a false positive rate of 0.00236%, which is arguably too much==. This would lead to false maintenance actions and operators would tend to ignore the warnings. However, this is a promising preliminary result and similar numbers were found when testing on vehicle. Additional logic regarding the number of messages and better tuning of the timing parameters should drop the false positive rate even lower.
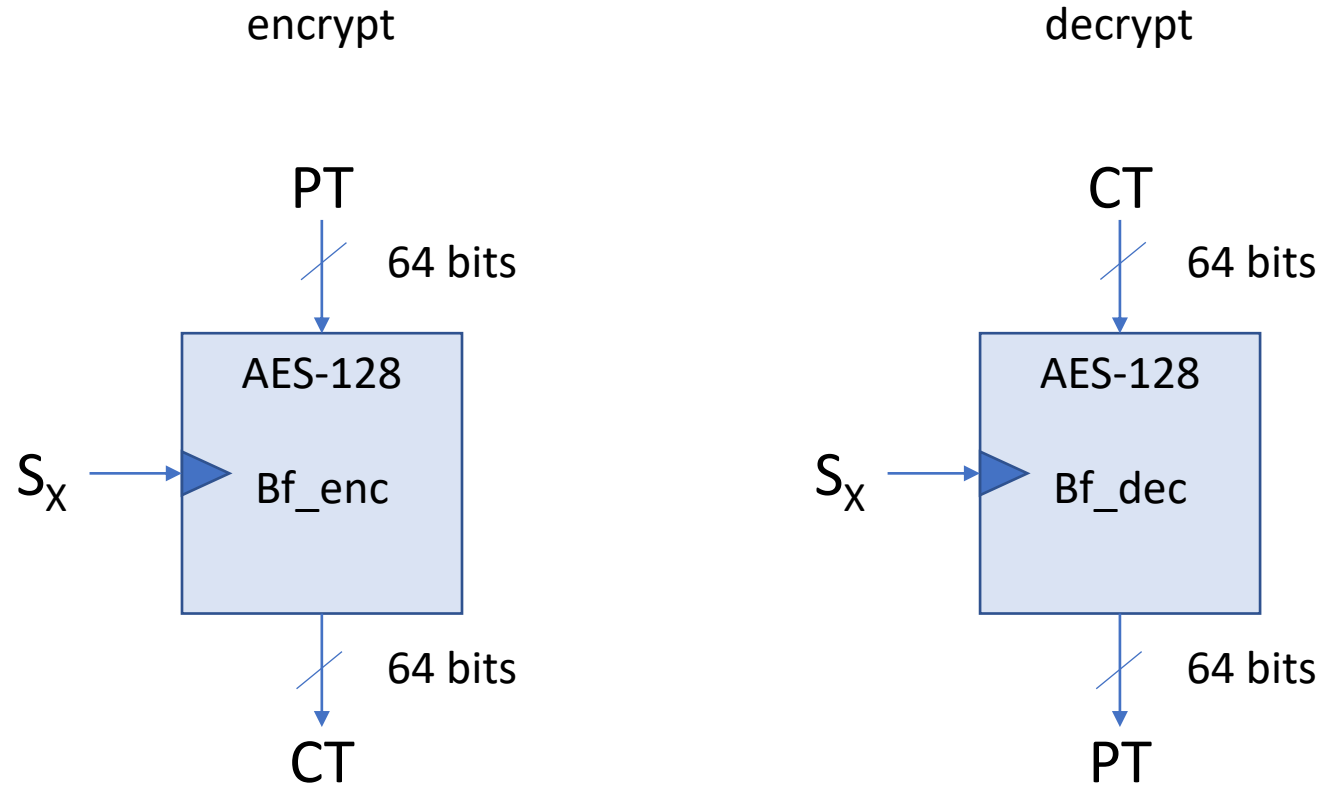
Something Completely Different
(and NOT General Use)

- Say you have a set of messages, M

- Say for each message $\mathbf{m_i}$ in M there are two bytes not being used AND those bytes have a fixed value, e.g., 0xFF.
  - The two bytes are likely to not be in the same byte position for each $\mathbf{m_i}$, so you have a table T that lists the byte positions and their fixed values

- Use the lack of entropy to create
  - `[F16|0 I16|0 C64]0`

Something Completely Different
(and NOT General Use)

- You can take advantage of the lack of entropy to secure each Tx msg:
    - FV = FV + 1   -- 16-bit freshness
    - PT = data XOR FV bytes (where XOR occurs at fixed value bytes)
    - CT = Encrypt the block (PT)

- Recipient
    - FV = FV + 1
    - PT = Decrypt the block (CT)
    - data = PT XOR FV bytes
    - Are fixed values the expected value? If yes, then no tampering

Blowfish (Block mode)

encrypt

decrypt

PT

CT

64 bits

64 bits

AES-128

AES-128

S_X

Bf_enc

S_X

Bf_dec

64 bits

64 bits

CT

PT

S_X: symmetric key for entity "x"
PT: plaintext
CT: ciphertext

block: ECB, but limited to 1 block

https://pypi.org/project/blowfish/#block

Issues

- Fragile FV
  - Requires some kind of sync scheme

Lab

- Use Sx = 00000000 11111111 22222222 33333333