Even fresh message can be replayed

Repurpose the AES block cipher to secure a stream.

Lab3

Encrypting Most Everything, but using Stream Cipher

Remember

There is no 100% security

Security, like all engineering, involves tradeoffs

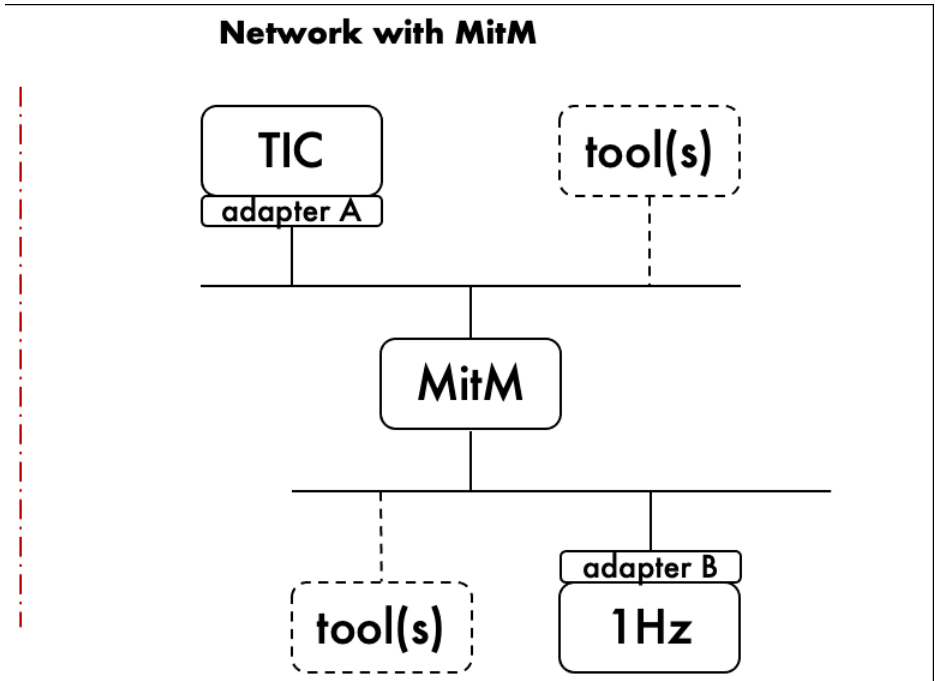
Know what you are trying to secure

The adversary…

State Sponsored

Network Configuration

Use just MitM configuration in this lab



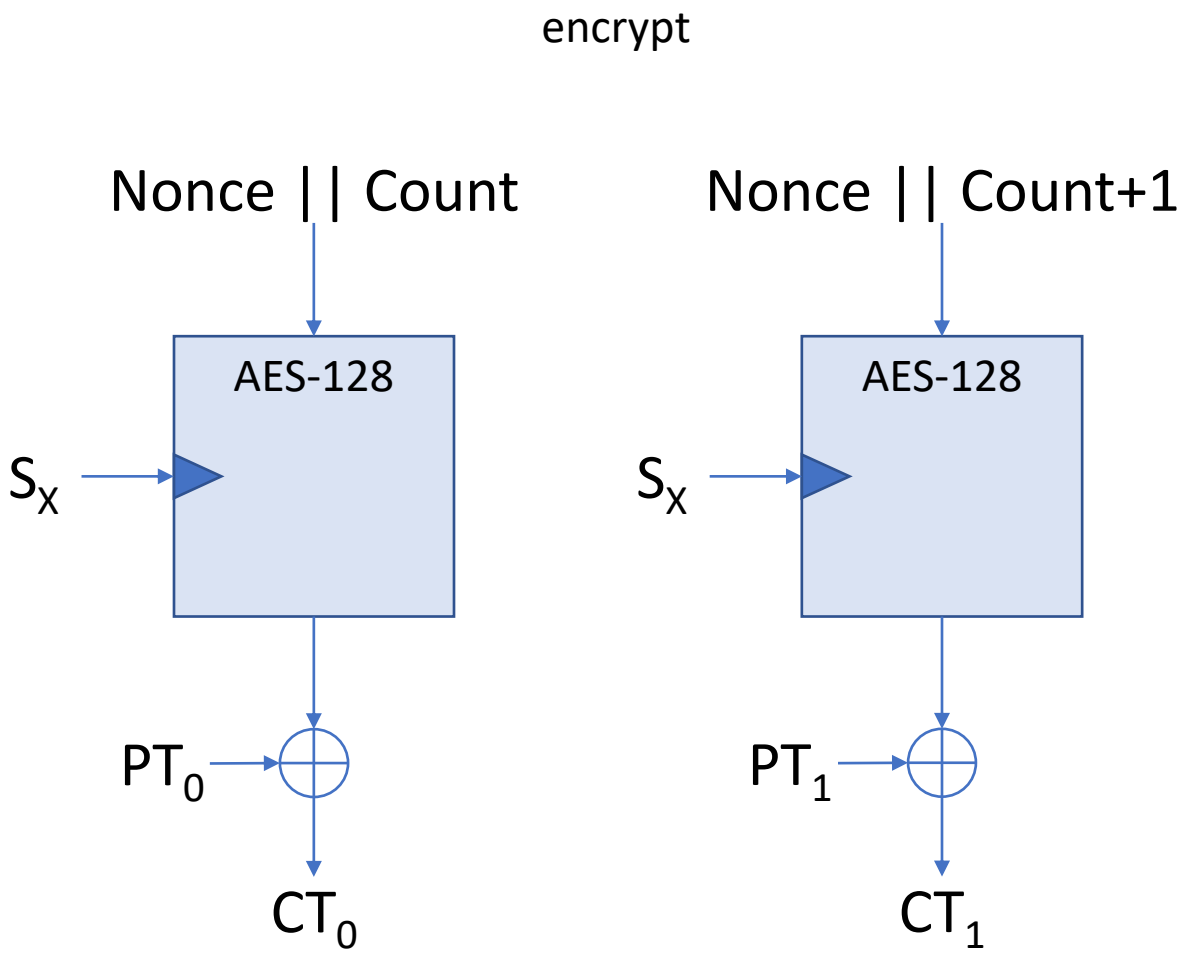
1Hz : 1 Hz generators of J1939 messages
MitM : Man-in-the-Middle
ECU : ECU added (& controlled) by student
TIC : Text Instrument Cluster
tool(s): One or more of can-utils (canplayer, candump, cansniffer, etc.)

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

Historical Reference

- Personal experience – brainstorming sessions early in the development of product security
  - Make it “easy”, “just encrypt everything”
  - “128-bit encryption is unbreakable”
  - It must fit in a standard CAN frame

AES-128 CTR

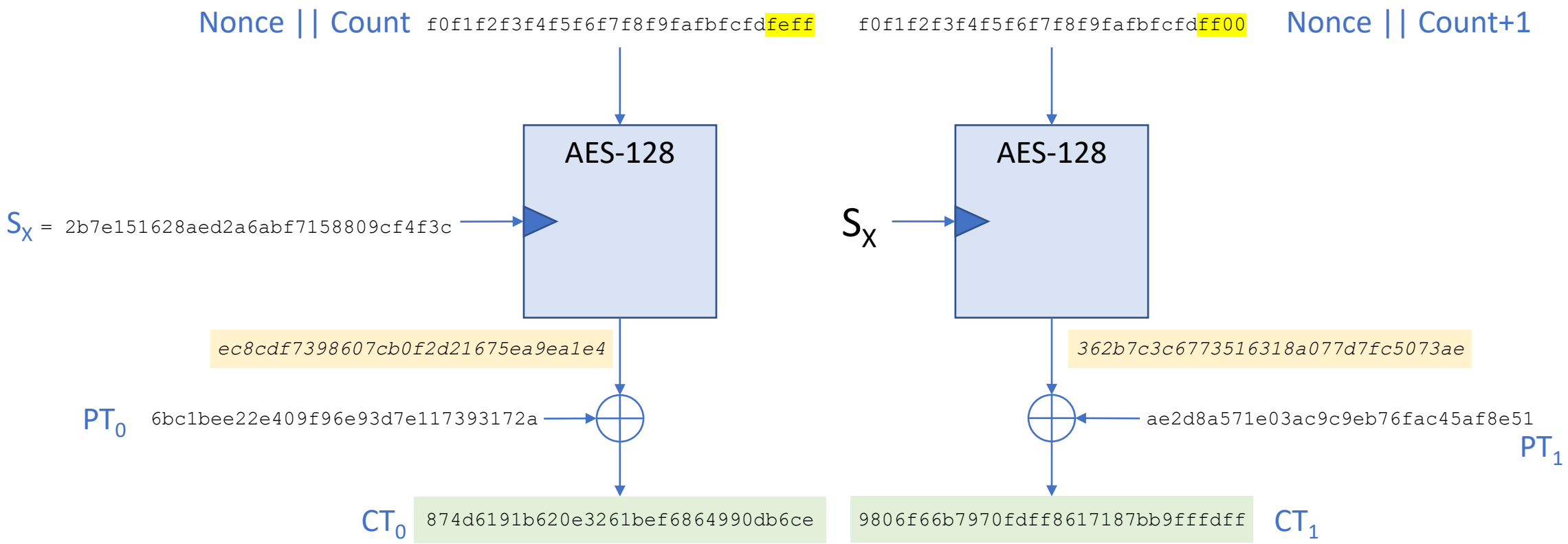


S_X: symmetric key for entity “x”
PT: plaintext
CT: ciphertext

CTR: counter mode

AES-128 CTR **NIST 800-38A**

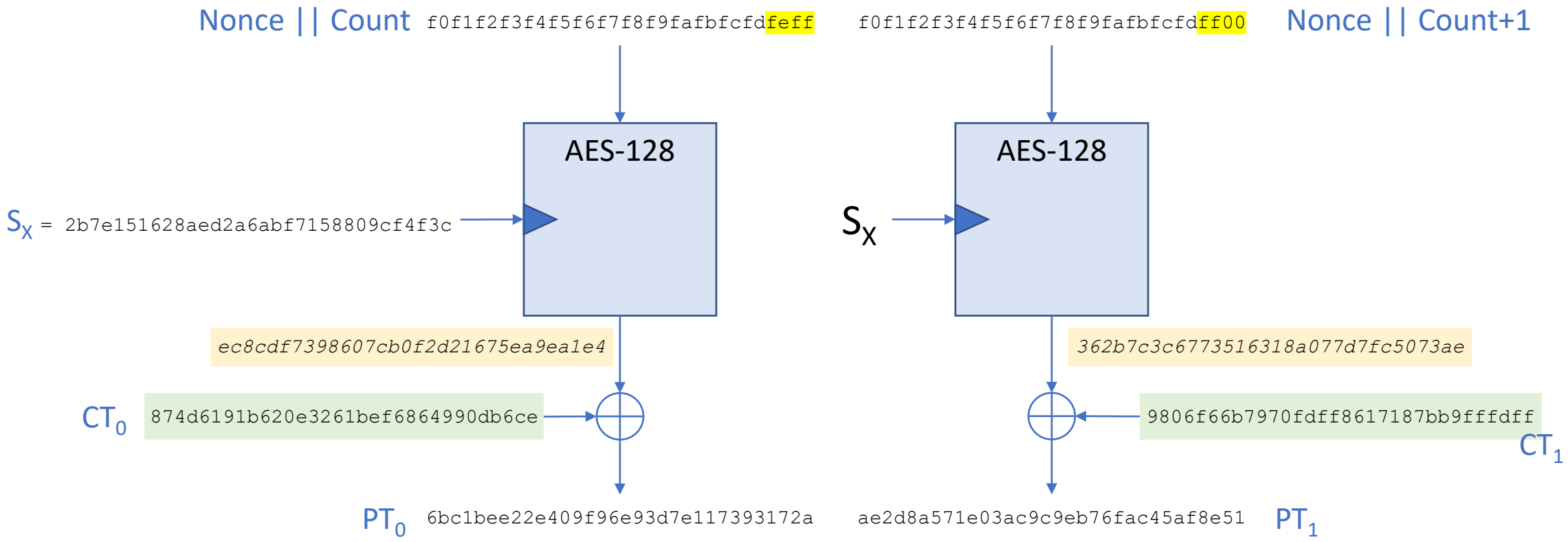
encrypt



https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

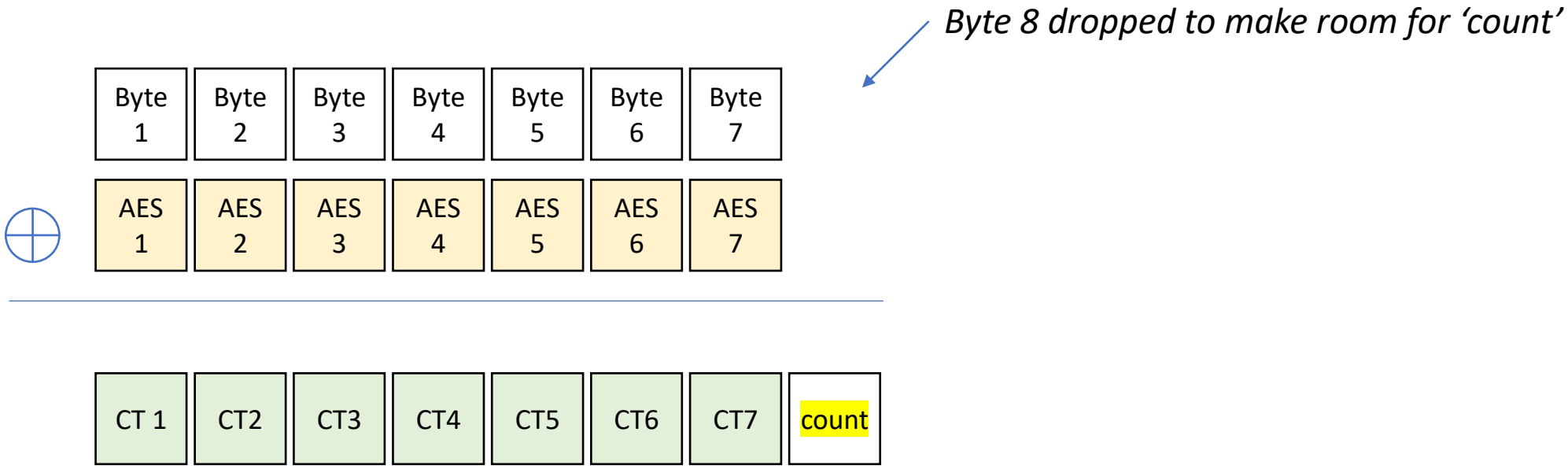
AES-128 CTR **NIST 800-38A**

decrypt

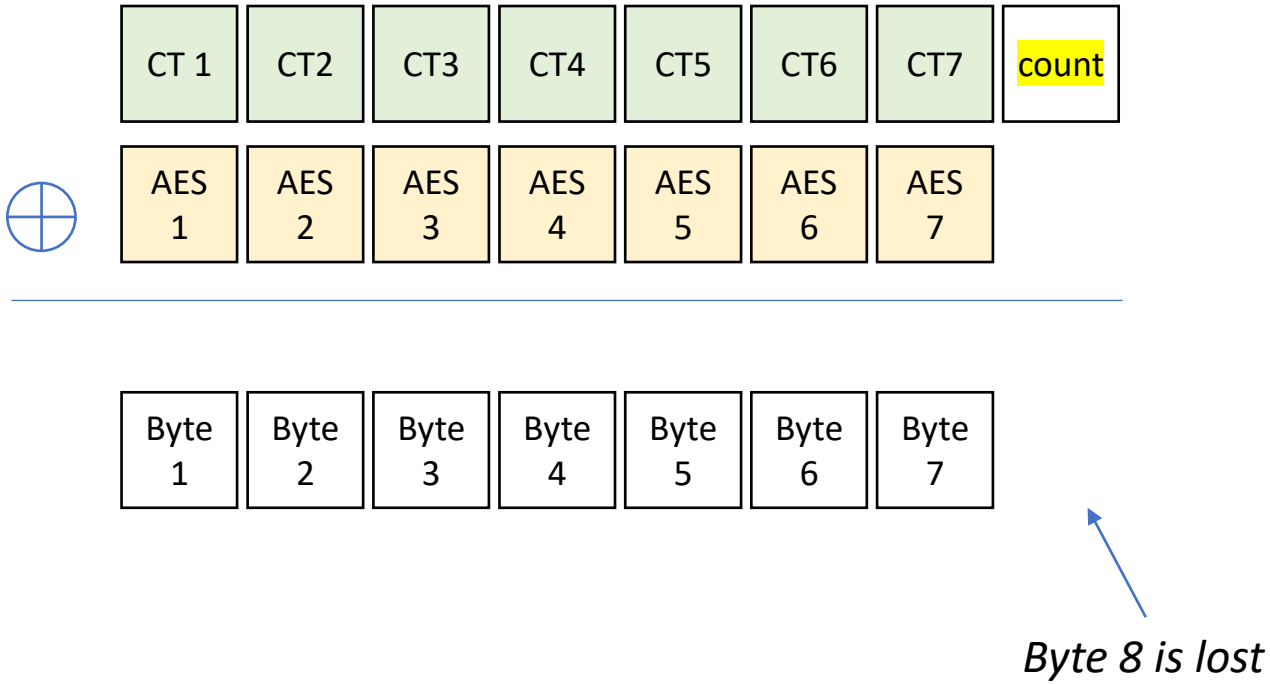


https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

CTR Example, Encrypt



CTR Example, Decrypt



CTR Example

```
> cat -n ctr.py
     1  #
     2  # reference:
     3  # https://cryptography.io/en/latest/hazmat/primitives/symmetric-encryption/
     4
     5  from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
     6
     7  count = 0  # will range thru 0..255
     8
     9  Sx =    bytes.fromhex("00000000 11111111 22222222 33333333")
    10  nonce = bytes.fromhex("00000000 00000000 00000000 000000")
    11  count = count + 1
    12  iv = bytes(nonce) + count.to_bytes(1, "big")
    13  print("iv")
    14  print(iv.hex(" ", 4))
    15
    16  cipher = Cipher(algorithms.AES(Sx), modes.ECB())
    17  encryptor = cipher.encryptor()
    18  ctiv = encryptor.update(iv) + encryptor.finalize()
    19  print("ctiv")
    20  print(ctiv.hex(" ", 4))
    21
    22  data = bytes.fromhex("dead cafe beef 0102")
    23  ct = bytes([a ^ b for a, b in zip(ctiv, data)])
    24  print("ct")
    25  print(ct.hex(" ", 4))
    26
    27  print("now, to decrypt...")
    28  pt = bytes([a ^ b for a, b in zip(ctiv, ct)])
    29  print("pt")
    30  print(pt.hex(" ", 4))
```

```
> python3 ctr.py | cat -n
     1  iv
     2  00000000 00000000 00000000 00000001
     3  ctiv
     4  b01731f7 55efc4f5 ed9a1e20 a9963816
     5  ct
     6  6ebafb09 eb00c5f7
     7  now, to decrypt...
     8  pt
     9  deadcafe beef0102
```

Lab

- Use Su = 00000000 11111111 22222222 33333333

- What is a valid message?
  - Established by Policy and the security needs
  - How do we detect invalid messages?
  - When we detect an issue, what do we do?
  - Without policies:
    - Unwanted data makes it through.
    - Sometimes it is malformed bytes that may exceed bounds during conversion to engineering units