Pretty Good Security

Stretching the value of a single byte.

Lab4

Pretty Good Security: Freshness and Integrity

Remember

There is no 100% security

Security, like all engineering, involves tradeoffs

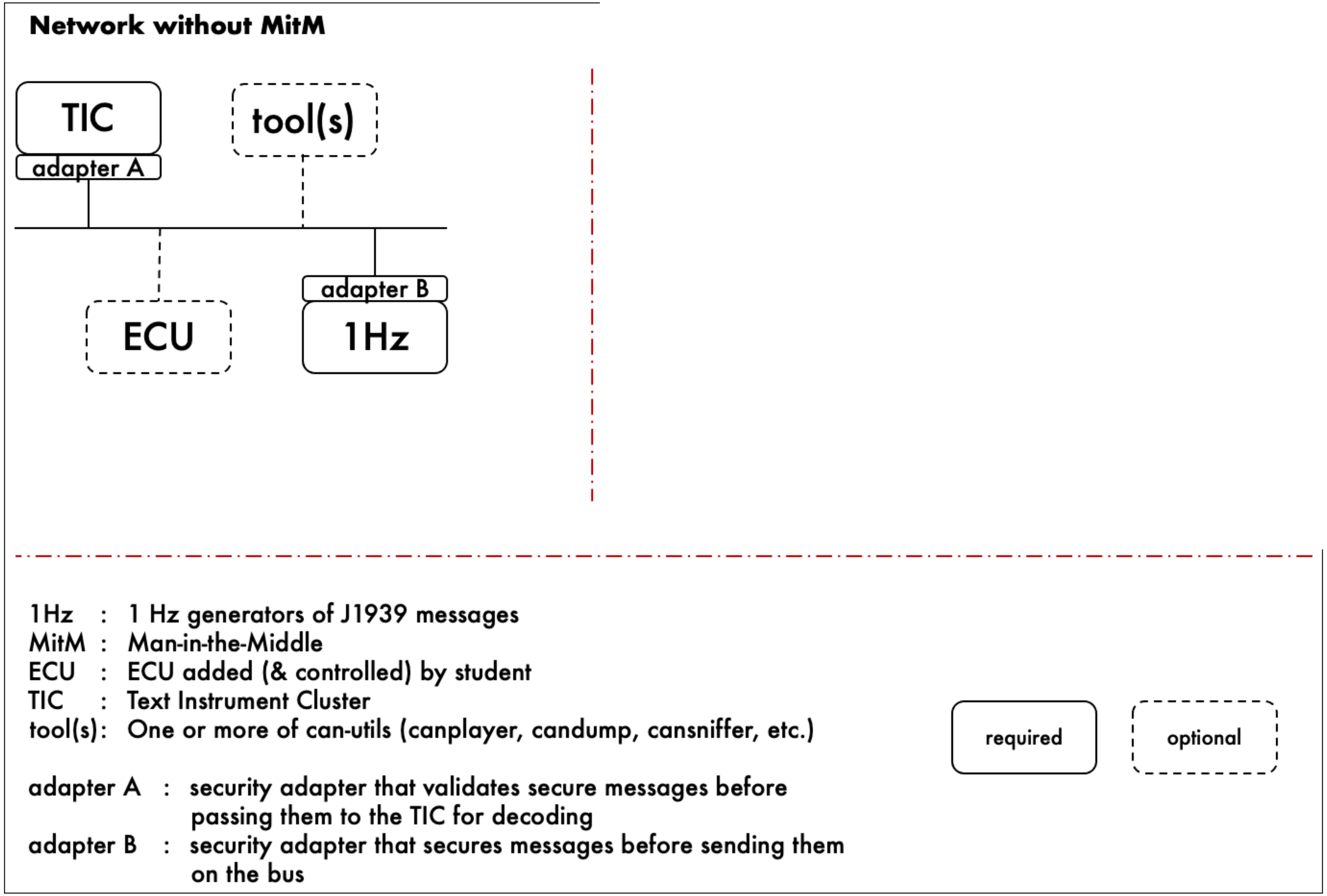
Know what you are trying to secure

The adversary…

State Sponsored

Network Configuration

Simple Network for this Lab



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

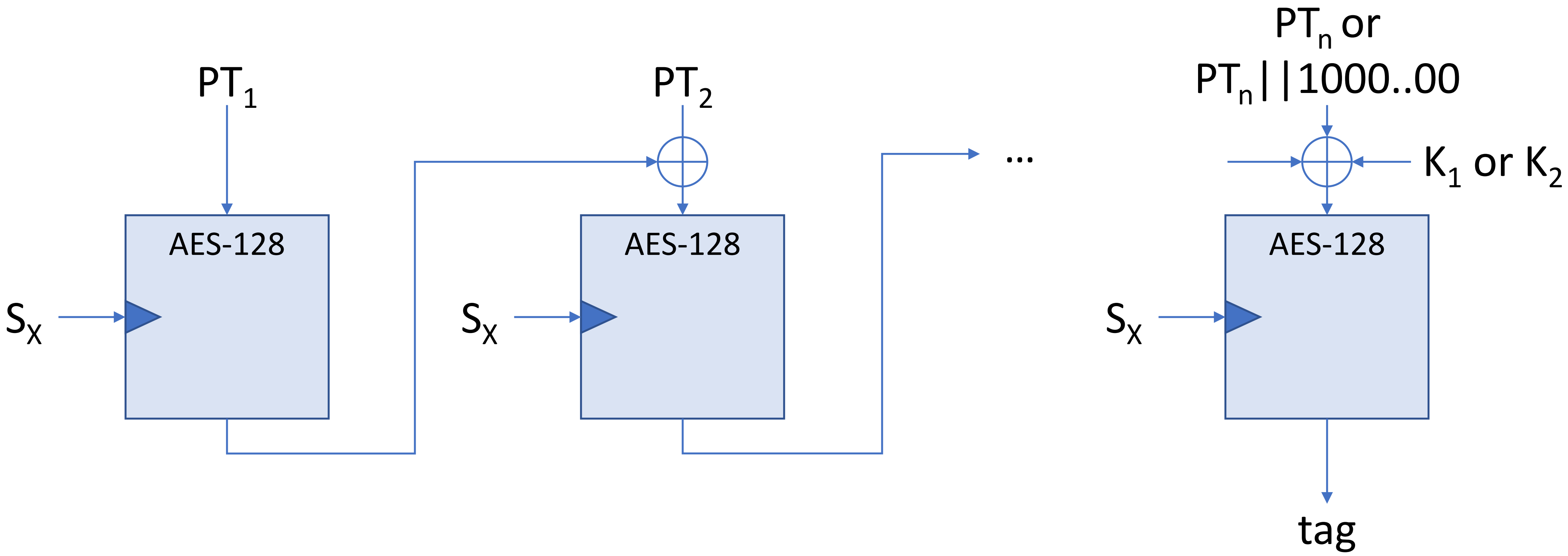
Historical Reference

- SAE J1939-91C
    - Integrity is extremely important
    - tag length (“Tlen”) is 31 bits

- AEF TIM “run time” security
    - Limited number of bits allocated to security
    - Existing method for discontinuing a session
    - tag length (“Tlen”) is 4 bits

AES-128 CMAC **RFC 4493**

Generate keyed tag

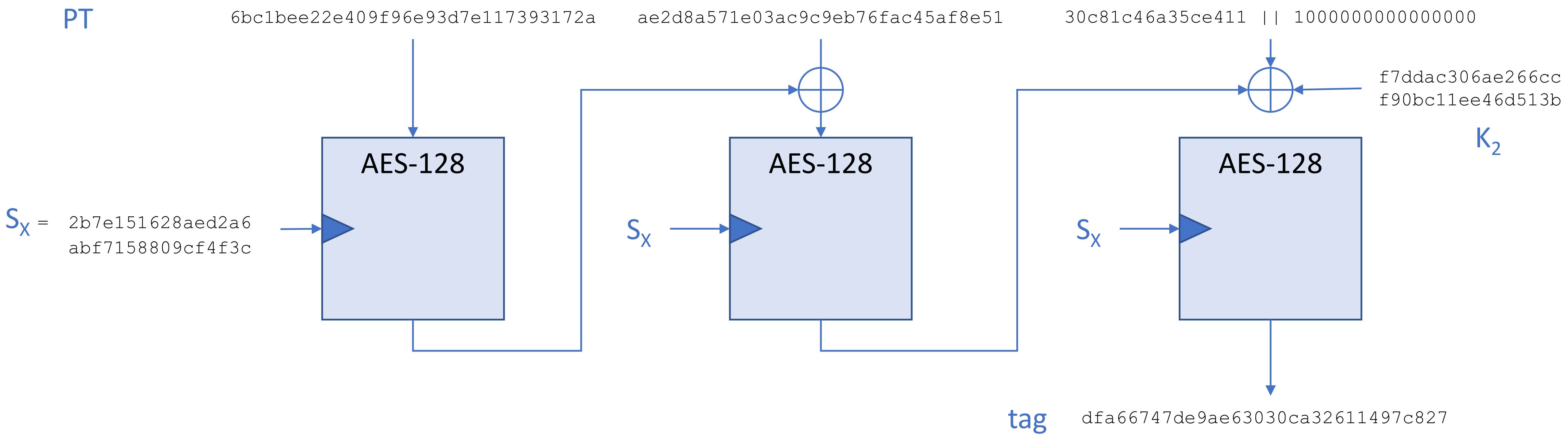


S_X: symmetric key for entity “x”
PT: plaintext
CT: ciphertext

CMAC: cipher message authentication code
Tag: fixed-size, keyed, cryptographic hash of plaintext (128 bits for AES-128 CMAC)

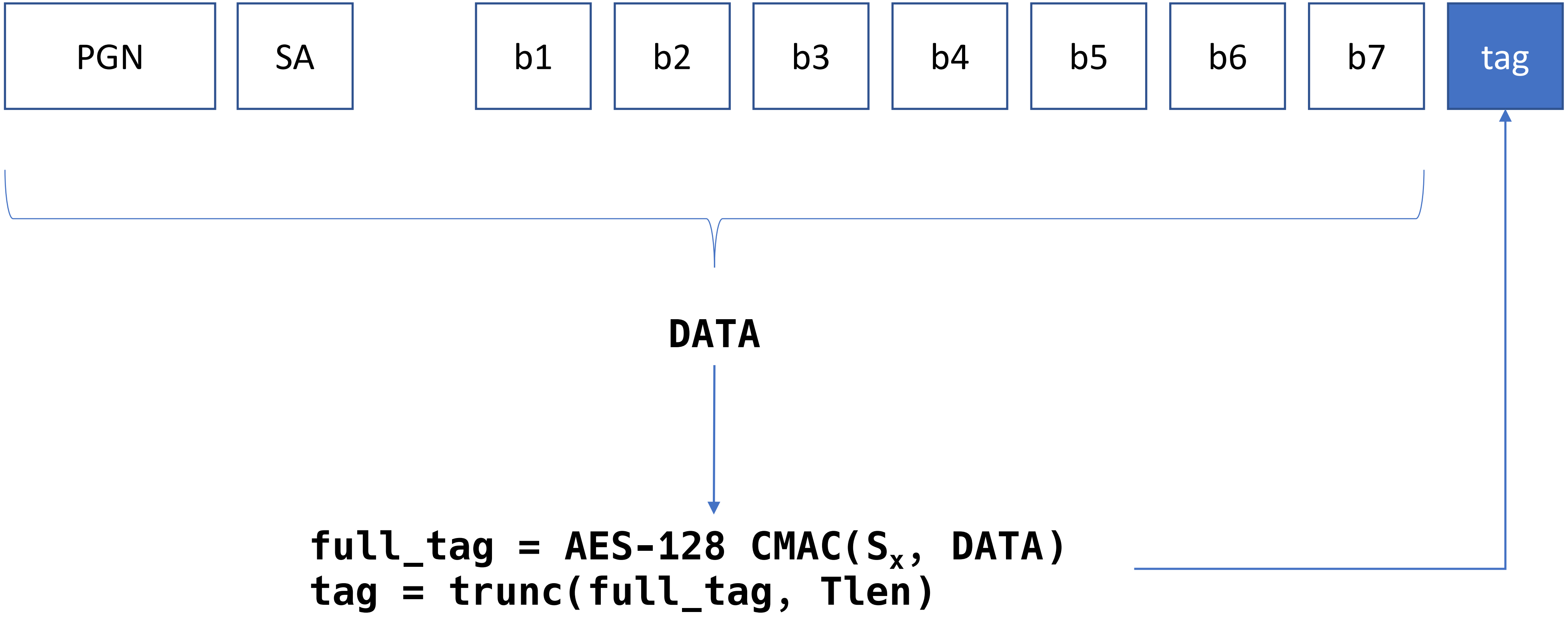
AES-128 CMAC **NIST 800-38B** **RFC 4493**

Generate keyed tag
Example 3 from RFC 4493


PT
6bc1bee22e409f96e93d7e117393172a
ae2d8a571e03ac9c9eb76fac45af8e51
30c81c46a35ce411 || 1000000000000000
f7ddac306ae266cc
f90bc11ee46d513b
K_2
AES-128
AES-128
AES-128
S_X = 2b7e151628aed2a6
abf7158809cf4f3c
S_X
S_X
tag
dfa66747de9ae63030ca32611497c827

https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-38b.pdf
https://www.rfc-editor.org/rfc/rfc4493.html

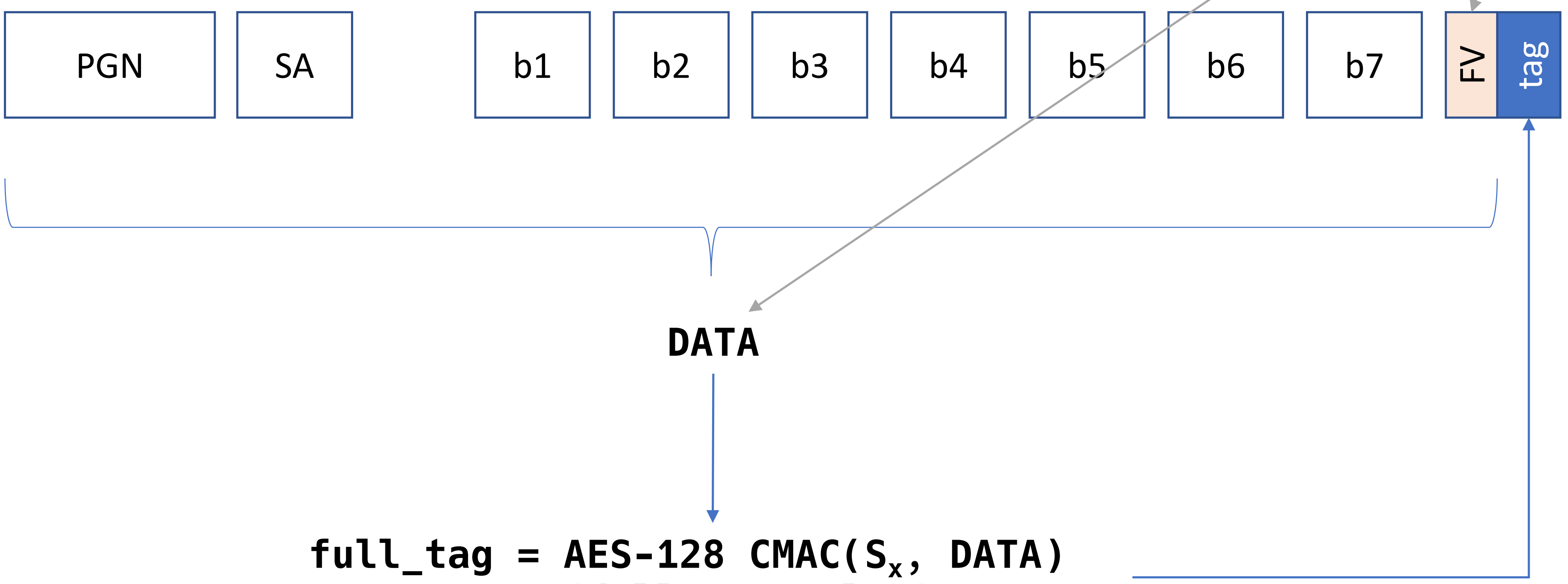
Using CMAC



full_tag = AES-128 CMAC(S_x, DATA)
tag = trunc(full_tag, Tlen)

Consider explicit vs implicit freshness

Using CMAC with Freshness



DATA

full_tag = AES-128 CMAC(S_x, DATA)
tag = trunc(full_tag, Tlen)

Using CMAC

- Transmitter calculates tag over the data
  - Sends **`DATA || tag`**
- Receiver calculates tag over the data it received and compares
  - **`tag' = CMAC(DATA)`**
  - **`tag' == tag → VALID`**
  - **`tag' != tag → INVALID`**

- If CMAC tag is INVALID – message **is not** trustworthy

- If CMAC tag is VALID – message **is probably** trustworthy
  - Attacker may have guessed the tag
  - Easier to do with shorter tags (smaller *Tlen*)

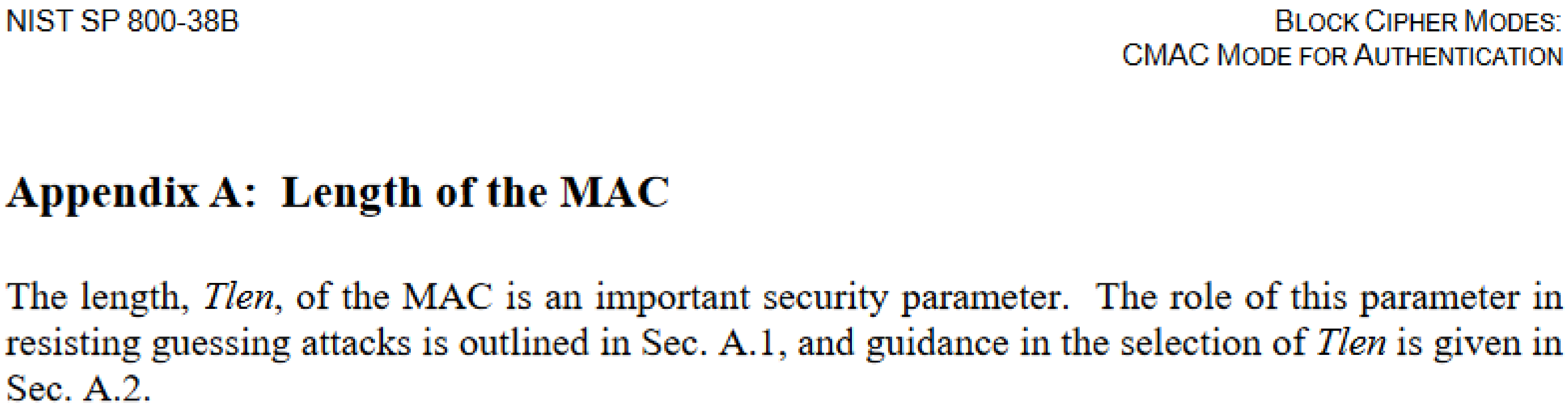
Tlen: The Tag Length

- AES-128 CMAC, full tag length is 128 bits
  - Often too much overhead for vehicle networks

- 2^{128} is huge space – ok to truncate the tag

- *Tlen* is number of bits in the truncated tag

Tlen tradeoffs



NIST SP 800-38B

BLOCK CIPHER MODES:
CMAC MODE FOR AUTHENTICATION

Appendix A: Length of the MAC

The length, *Tlen*, of the MAC is an important security parameter. The role of this parameter in resisting guessing attacks is outlined in Sec. A.1, and guidance in the selection of *Tlen* is given in Sec. A.2.

Tlen: The Tag Length

- Larger values of *Tlen*: greater protection against guessing attacks
- Smaller values of *Tlen*: less overhead for security
- Sizing
  - How lucky to you require the attacker to be? *Risk*
  - What is the limit for INVALID tgs? *MaxInvalids*”

$$\boxed{Tlen \geq \lg(MaxInvalids\,/\,Risk)}$$

Tlen: The Tag Length

$$\boxed{Tlen \geq \lg(MaxInvalids\,/\,Risk)}$$

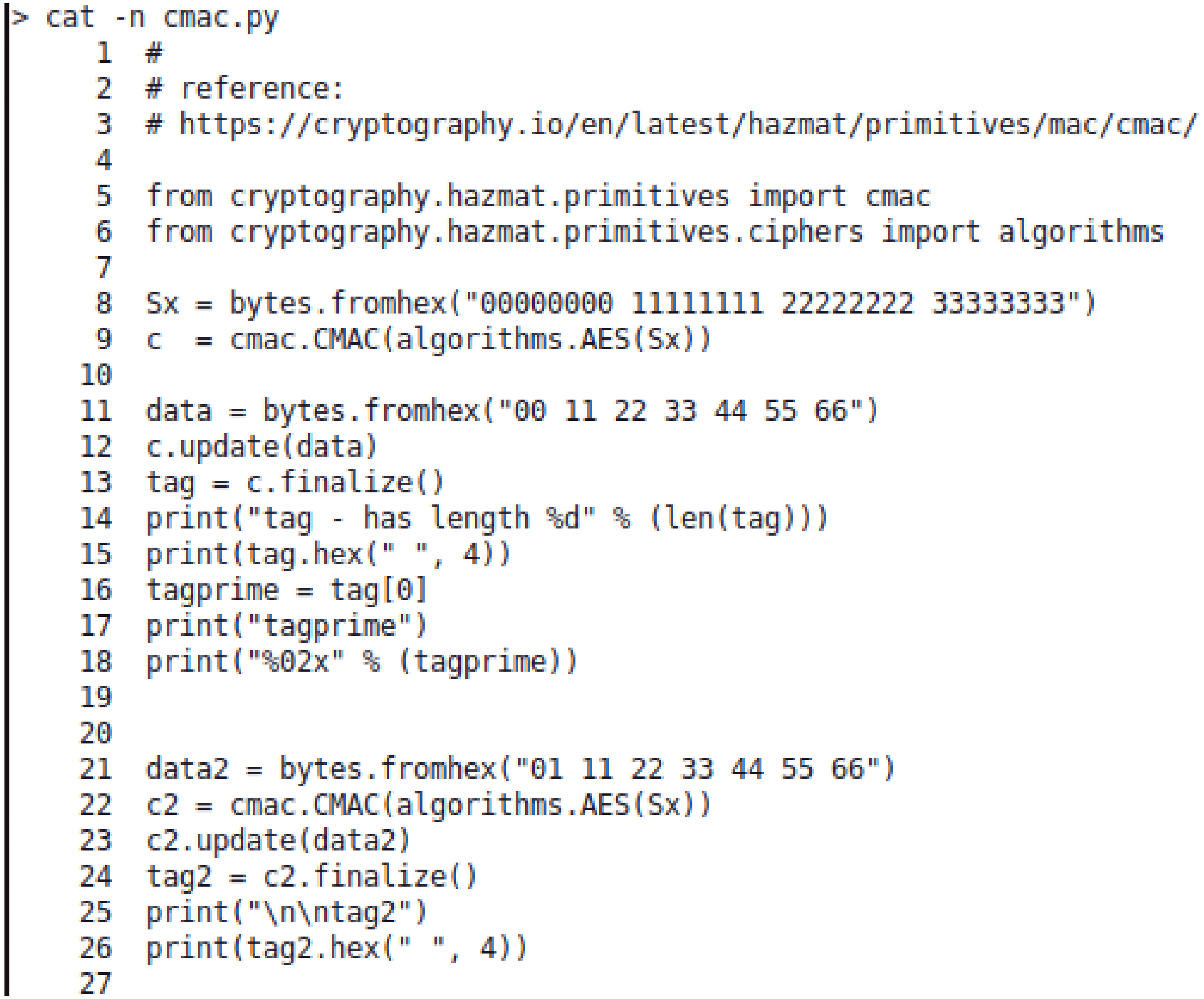
System A

Say we retire the key after 128 INVALID tags: *MaxInvalids* = 2^7
Say we can accept a 1 in 1,048,576 chance of guessing the tag: *Risk* = 2^{-20}
Then *Tlen* >= $\log_2(2^7 / 2^{-20}) = \log_2(2^{27})$ = **27 bits**

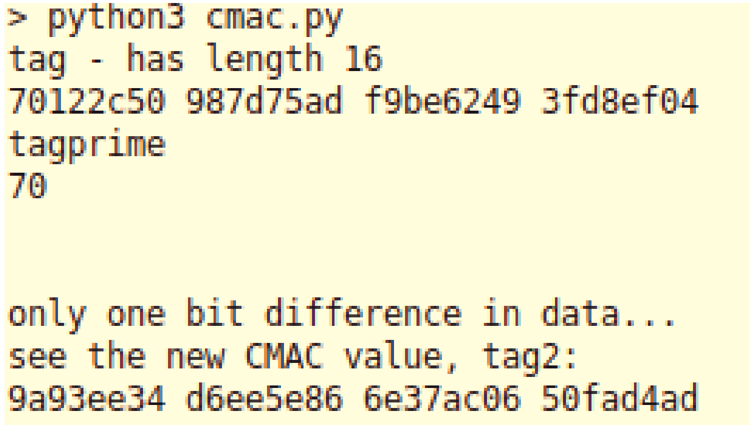
System B

Say we retire the key after 8 INVALID tags: *MaxInvalids* = 2^3
Say we can accept a 1 in 32 chance of guessing the tag: *Risk* = 2^{-5}
Then *Tlen* >= $\log_2(2^3 / 2^{-5}) = \log_2(2^8)$ = **8 bits**

CMAC example



```
> cat -n cmac.py
     1  #
     2  # reference:
     3  # https://cryptography.io/en/latest/hazmat/primitives/mac/cmac/
     4
     5  from cryptography.hazmat.primitives import cmac
     6  from cryptography.hazmat.primitives.ciphers import algorithms
     7
     8  Sx = bytes.fromhex("00000000 11111111 22222222 33333333")
     9  c  = cmac.CMAC(algorithms.AES(Sx))
    10
    11  data = bytes.fromhex("00 11 22 33 44 55 66")
    12  c.update(data)
    13  tag = c.finalize()
    14  print("tag - has length %d" % (len(tag)))
    15  print(tag.hex(" ", 4))
    16  tagprime = tag[0]
    17  print("tagprime")
    18  print("%02x" % (tagprime))
    19
    20
    21  data2 = bytes.fromhex("01 11 22 33 44 55 66")
    22  c2 = cmac.CMAC(algorithms.AES(Sx))
    23  c2.update(data2)
    24  tag2 = c2.finalize()
    25  print("\n\ntag2")
    26  print(tag2.hex(" ", 4))
    27
```



```
> python3 cmac.py
tag - has length 16
70122c50 987d75ad f9be6249 3fd8ef04
tagprime
70



only one bit difference in data...
see the new CMAC value, tag2:
9a93ee34 d6ee5e86 6e37ac06 50fad4ad
```

Freshness

- Explicit
  - All freshness values appear in the message
- Implicit
  - None of the freshness values appear in the message
  - (nodes keep track by counting messages or use some other value on the bus)
- Hybrid
  - Say freshness is 32-bit value, but only 4 bits appear in the message
  - Make the least-significant bits the explicit portion

Lab

- Use Su = 00000000 11111111 22222222 33333333