A Powerful and Easy to Use Lockbox

Easy to use correctly, hard to use incorrectly

Lab7

Easy to Use Correctly: NaCl "box" concept

Remember

There is no 100% security

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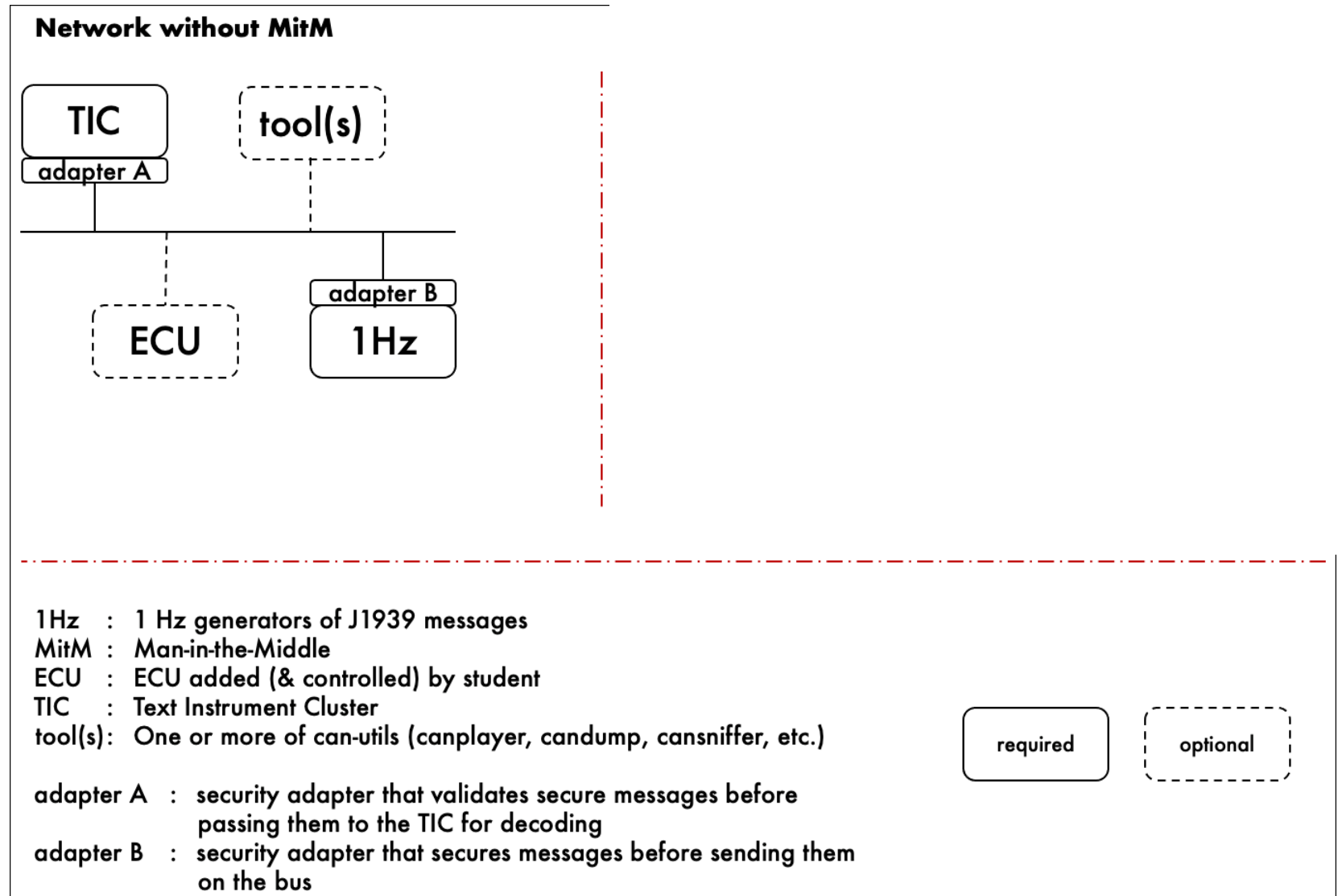
Know what you are trying to secure

The adversary…

State Sponsored

Network Configuration

Simple Network for this Lab

Network without MitM

TIC
adapter A

tool(s)

ECU

adapter B
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

Notes

Complexity is the enemy of security.

- Bruce Schneier

TweetNaCl demonstrates a full library in 100 tweets.

- All elliptic curve code

- All encryption and hashing code
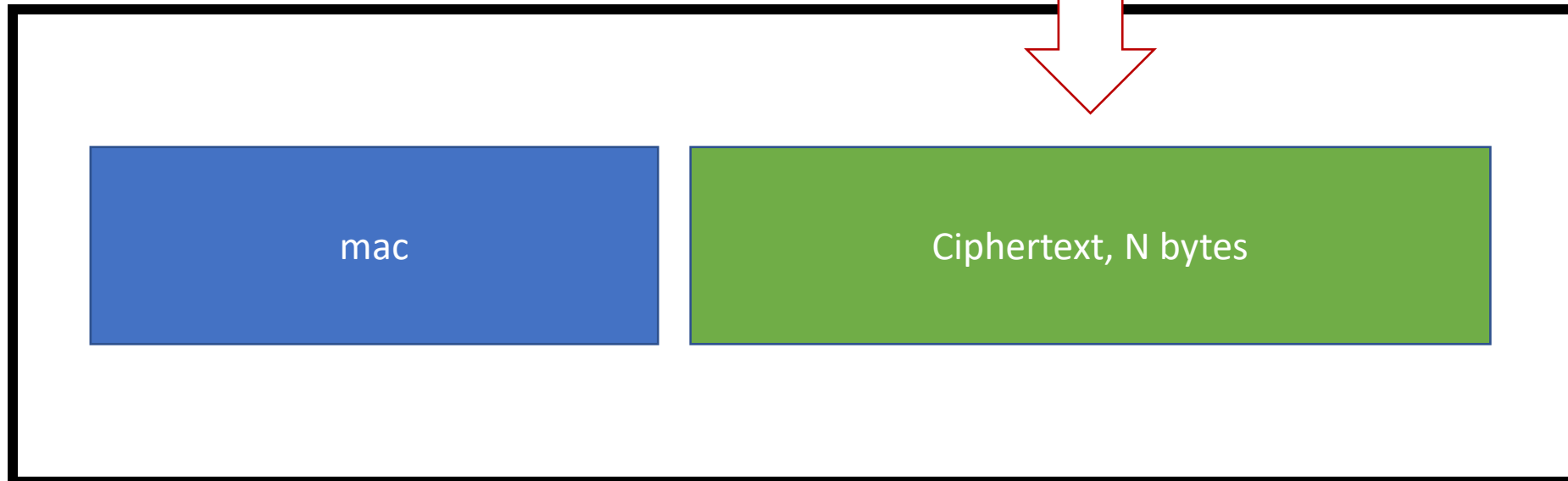

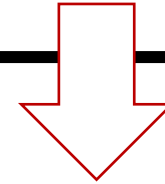NaCl – easy to use correctly, hard to use incorrectly

"Box"

To secure data being sent from Entity A to Entity B.

- Data
- Nonce (which *must* be shared with Entity B)
- Entity A private key
- Entity B public key

nonce	Data, N bytes

$Box(data, nonce, P_B, K_A)$

mac	Ciphertext, N bytes

"Box"

To secure data being sent from Entity A to Entity B.
- Every time data is sent, a <u>new nonce is required</u>.

$Box(data, nonce, P_B, K_A)$

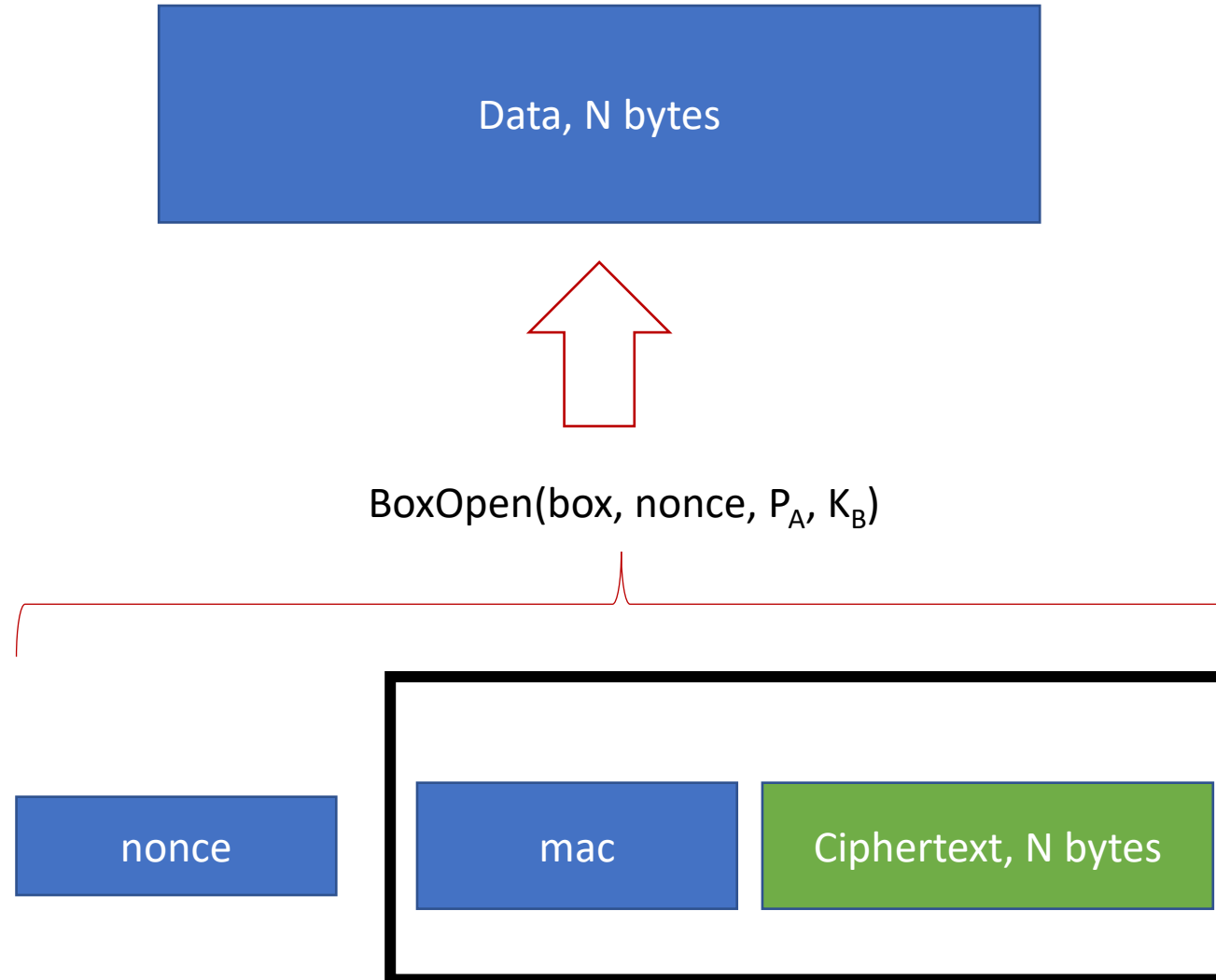| mac | Ciphertext, N bytes |

"BoxOpen"

To validate and decrypt data being sent from Entity A to Entity B.
- "box" (mac || ct)
- Nonce (which was provided by Entity A)
- Entity A public key
- Entity B private key

Data, N bytes

$BoxOpen(box, nonce, P_A, K_B)$

nonce

mac

Ciphertext, N bytes

Notes

- Every library seems to label the functions a little differently

- In "TweetNaCl" the functions are very straightforward:
  - nacl.box()
  - nacl.boxopen()

- In python library it takes a couple of steps.

Notes

Used to put data
into box and get
it out.

```
> cat -n using_nacl.py
     1  #
     2  # reference:
     3  # https://pynacl.readthedocs.io/en/latest/public/#nacl-public-box
     4
     5  import nacl.utils
     6  from nacl.public import PrivateKey, Box
     7
     8  # notes on key naming
     9  # P/Kx  -- Public/Private keypair for entity x
    10  # Sx    -- Symmetric key for entity x
    11
    12  # Generate Bob's private key (!! keep this secret !!)
    13  Kb = PrivateKey.generate()
    14
    15  # Use the private key to create the public key -- this must be shared
    16  Pb = Kb.public_key
    17
    18  # Alice does the same thing
    19  Ka = PrivateKey.generate()
    20  Pa = Ka.public_key
    21
    22  # <here Alice and Bob exchange public keys>
    23
    24  # Bob wants to box some data for Alice;
    25  # He needs his private key and Alice's public key
    26  boxerBA = Box(Kb, Pa)
    27
```

Notes

```
28  data = bytes.fromhex("00000000 11111111 22222222 33333333")
29  print("data to secure")
30  print(data.hex(" ", 4))
31
32  # the library automatically created a nonce for us.
33  # using boxerBA results in a box that is:
34  # mac || nonce || data   === 16 bytes || 24 bytes || len(data)
35  # so total length is 40 bytes longer than data
36  box = boxerBA.encrypt(data)
37
38
39  ### Alice is given the box, 'box', and needs to open it.
40
41  # create a box opener, she uses Bob's public key and her private key
42  boxopenerBA = Box(Ka, Pb)
43
44  # use the opener
45  pt = boxopenerBA.decrypt(box)
46
47  print("data received by Alice")
48  print(pt.hex(" ", 4))
```

'encrypt' puts data into box

'decrypt' unboxes the data

```
> python3 using_nacl.py
data to secure
00000000 11111111 22222222 33333333
data received by Alice
00000000 11111111 22222222 33333333
```

Notes

See "classroom/using_nacl_nonce.py" for example setting your own nonce.

```
data to secure
00000000 11111111 22222222 33333333
using nonce
00000000 00000000 00000000 00000000 00000000 00000001
size of box is 40 + len(data): 56
data received by Alice
00000000 11111111 22222222 33333333
```

Lab

- The only code required is for:
  - adapterA.py
  - adapterB.py

Lab

nonce

However, if we need to use an explicit nonce, it can be passed along with the message:

```
# This is a nonce, it *MUST* only be used once, but it is not considered
#   secret and can be transmitted or stored alongside the ciphertext. A
#   good source of nonces are just sequences of 24 random bytes.
nonce = nacl.utils.random(Box.NONCE_SIZE)

encrypted = bob_box.encrypt(message, nonce)
```