Sharing Keys in Plain View

Using Elliptic-Curve Diffie-Hellman.

Lab6

Key management: Elliptic Curve Diffie-Hellman

with

Key Derivation Function (KDF)

Remember

There is no 100% security

Security, like all engineering, involves tradeoffs

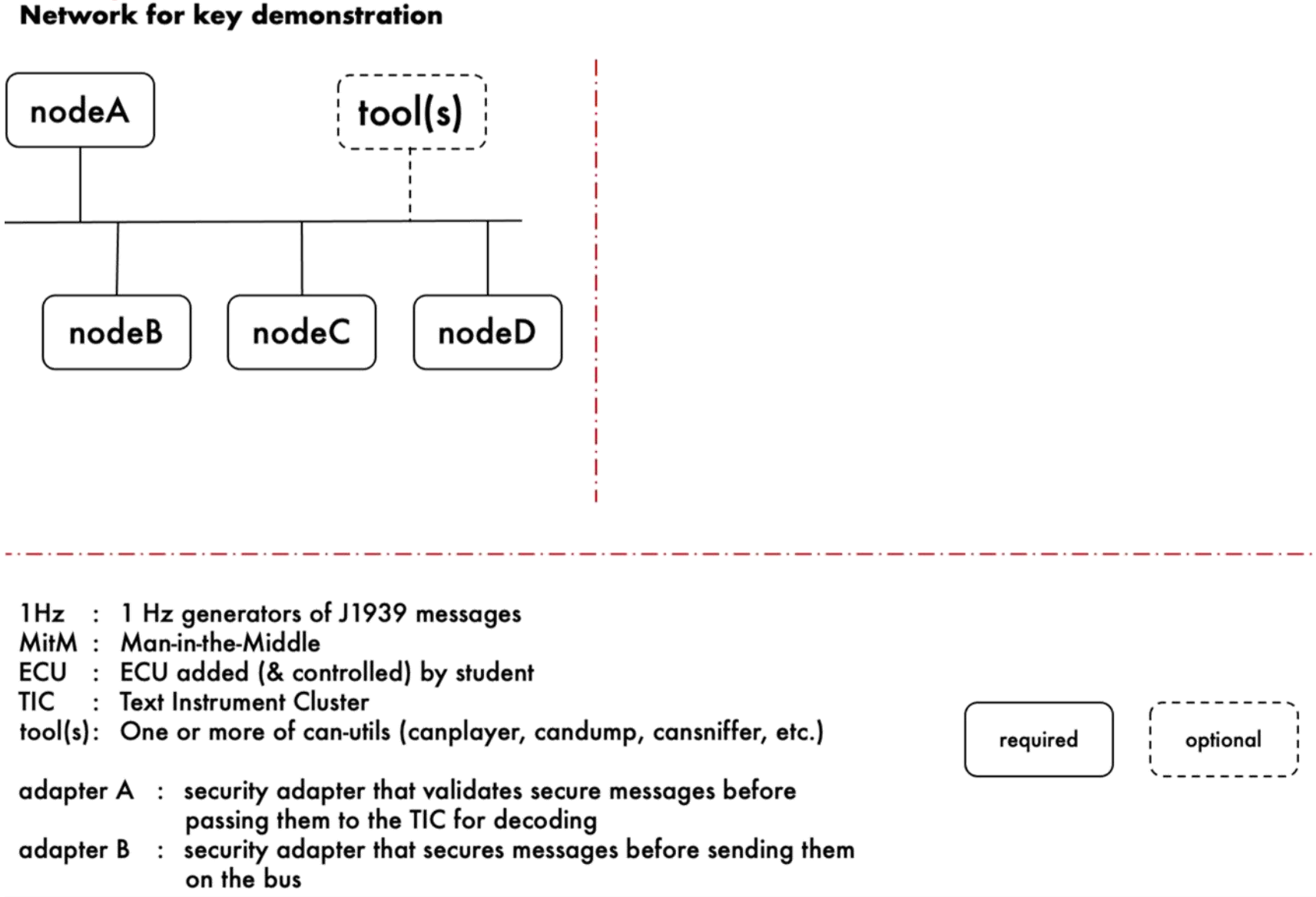
Know what you are trying to secure

The adversary…

State Sponsored

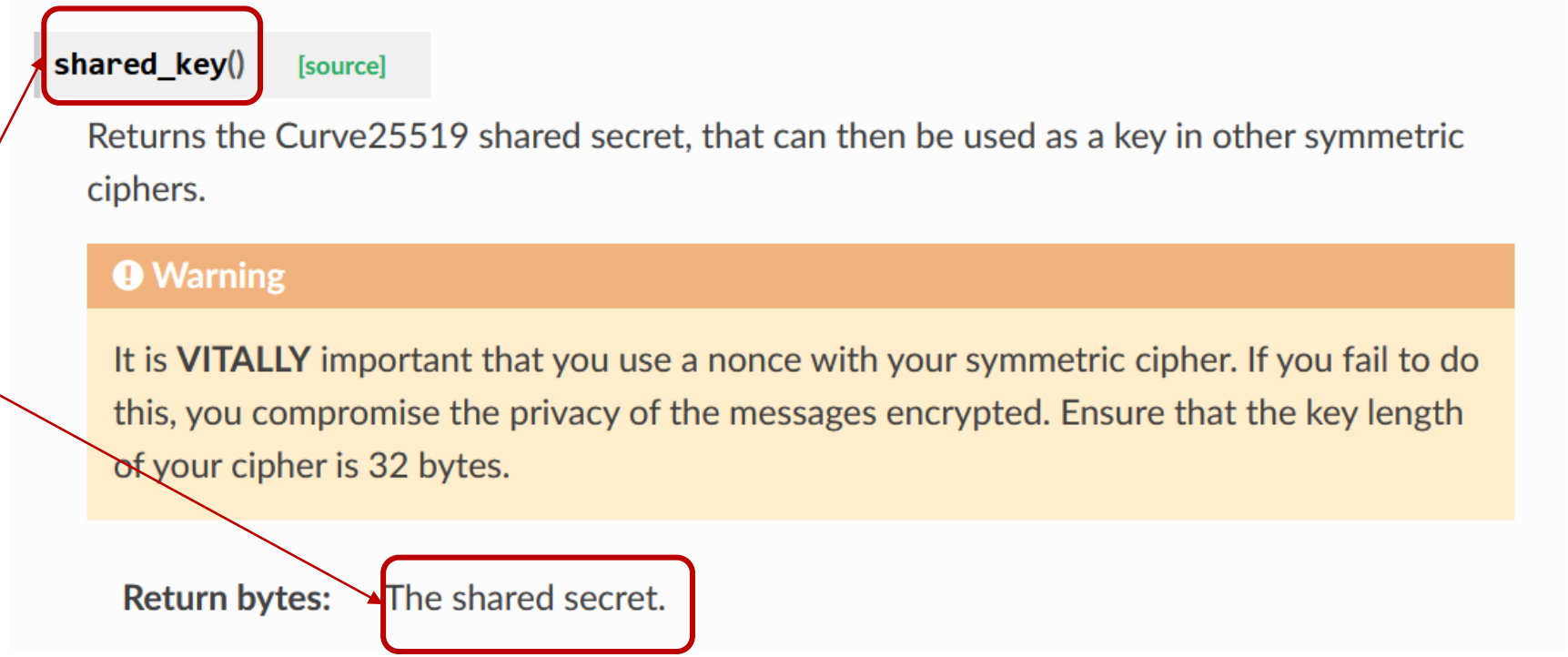
Network Configuration

Simple Network for this Lab



Notes – sometimes libraries are confusing

Mixing up terms!?



https://pynacl.readthedocs.io/en/latest/public/#nacl.public.Box

Notes – The Curve25519 paper

A hash of the shared secret Curve25519$(a, \text{Curve25519}(b, \underline{9}))$ is used as the key for a secret-key authentication system (to authenticate messages), or as the key for a secret-key authenticated-encryption system (to simultaneously encrypt and authenticate messages).

Can use a Key Derivation Function (KDF) as the hash

https://www.iacr.org/cryptodb/archive/2006/PKC/3351/3351.pdf

Notes

Danger

This is a "Hazardous Materials" module. You should **ONLY** use it if you're 100% absolutely sure that you know what you're doing because this module is full of land mines, dragons, and dinosaurs with laser guns.

Key derivation functions

Key derivation functions derive bytes suitable for cryptographic operations from passwords or other data sources using a pseudo-random function (PRF). Different KDFs are suitable for different tasks such as:

- Cryptographic key derivation

  Deriving a key suitable for use as input to an encryption algorithm. Typically this means taking a password and running it through an algorithm such as `PBKDF2HMAC` or `HKDF`. This process is typically known as key stretching.

- Password storage

  When storing passwords you want to use an algorithm that is computationally intensive. Legitimate users will only need to compute it once (for example, taking the user's password, running it through the KDF, then comparing it to the stored value), while attackers will need to do it billions of times. Ideal password storage KDFs will be demanding on both computational and memory resources.

https://cryptography.io/en/latest/hazmat/primitives/key-derivation-functions/#

Notes

ConcatKDF

class `cryptography.hazmat.primitives.kdf.concatkdf.ConcatKDFHash`(*algorithm*, *length*, *otherinfo*)
[source]

New in version 1.0.

ConcatKDFHash (Concatenation Key Derivation Function) is defined by the NIST Special Publication NIST SP 800-56Ar2 document, to be used to derive keys for use after a Key Exchange negotiation operation.

Warning

ConcatKDFHash should not be used for password storage.

Notes

- Use NaCl to derive sharedsecret
- Use concatKDF() to derive sharedkey
  - Use ‘otherinfo’ with jointly created nonce – allows same Public keys to create new symmetric keys.

Notes

```
> cat -n using_kdf.py
     1  #
     2  # reference:
     3  # https://pynacl.readthedocs.io/en/latest/public/#nacl.public.Box
     4  # https://cryptography.io/en/latest/hazmat/primitives/key-derivation-functions/#concatkdf
     5
     6  import nacl.utils
     7  from nacl.public import PrivateKey, Box
     8
     9  from cryptography.hazmat.primitives import hashes
    10  from cryptography.hazmat.primitives.kdf.concatkdf import ConcatKDFHash
    11
    12  # notes on key naming
    13  # P/Kx  -- Public/Private keypair for entity x
    14  # Sx    -- Symmetric key for entity x
    15
    16  # Generate Bob's private key (!! keep this secret !!)
    17  Kb = PrivateKey.generate()
    18
    19  # Use the private key to create the public key -- this must be shared
    20  Pb = Kb.public_key
    21
    22  # Alice does the same thing
    23  Ka = PrivateKey.generate()
    24  Pa = Ka.public_key
```

Notes

```
26  # By sharing just their public keys with each other,
27  # Alice and Bob have enough to derive a shared secret
28
29  # Alice knows her private key and Bob's public key
30  boxerAB = Box(Ka, Pb)
31  sharedsecret = boxerAB.shared_key()   # CONFUSING... the library docs say
32                                        # we are getting both the shared secret and a key.
33                                        # We are going to assume it is just the shared secret
34                                        # and that we still need to generate the key.
35
36  # since we want an AES-128 key we still need to do a KDF and get 16 bytes
37  ckdf = ConcatKDFHash(algorithm=hashes.SHA256(), length=16, otherinfo=b"00000000")
38  sharedkey = ckdf.derive(sharedsecret)
39
40  print("[Alice] the shared key is: %s" % (sharedkey.hex(" ", 4)), flush=True)
41
42  # Bob knows his private key and Alices's public key
43  boxerBA = Box(Kb, Pa)
44  sharedsecret = boxerBA.shared_key()
45  # since we want an AES-128 key we still need to do a KDF and get 16 bytes
46  ckdf = ConcatKDFHash(algorithm=hashes.SHA256(), length=16, otherinfo=b"00000000")
47  sharedkey = ckdf.derive(sharedsecret)
48
49  print("  [Bob] the shared key is: %s" % (sharedkey.hex(" ", 4)), flush=True)
```

secret → key

otherinfo

```
> python3 using_kdf.py
[Alice] the shared key is: 3bf536db 46788954 56450fb0 6e27514a
  [Bob] the shared key is: 3bf536db 46788954 56450fb0 6e27514a
```

Notes

HVOC_2023 / lab6 / classroom / randbits_and_kdf.py /

```python
from cryptography.hazmat.primitives import hashes                       # for SHA256 hash
from cryptography.hazmat.primitives.kdf.concatkdf import ConcatKDFHash  # for kdf
import secrets                                                          # for randbits

x = secrets.randbits(8)
print(x)

x = secrets.randbits(64)
xb = x.to_bytes(8, "big")        # converts 64 bits into 8 bytes
print(xb.hex(" ", 4))            # example: 61e015f7 920a8f0a

y = secrets.randbits(64)
yb = y.to_bytes(8, "big")        # example: 13a56ccd bed1a7c8

zb = xs + ys
print(zb.hex(" ", 4))            # example: 61e015f7 920a8f0a 13a56ccd bed1a7c8

Sn = bytes.fromhex("00000000 11111111 22222222 33333333")

keymaterial = Sn + zb
print(keymaterial.hex(" ", 4))   # example: 00000000 11111111 22222222 33333333 61e015f7 920a8f0a 13a56ccd bed1a7c8

sv = kdf.derive(keymaterial)
print(sv.hex(" ", 4))            # example: fb2e494f df1c65a1 01e8649f f59f7bd6
```

Lab

- The provided nodeA.py and nodeB.py MUST BE FIXED.
  - They need to include a KDF.