Sharing Keys in Plain View

Using Elliptic-Curve Diffie-Hellman.

Lab6

Key management: Elliptic Curve Diffie-Hellman

and

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

Concepts

- Entropy
- Elliptic-Curve Diffie-Hellman
- Key Derivation Function

Entropy

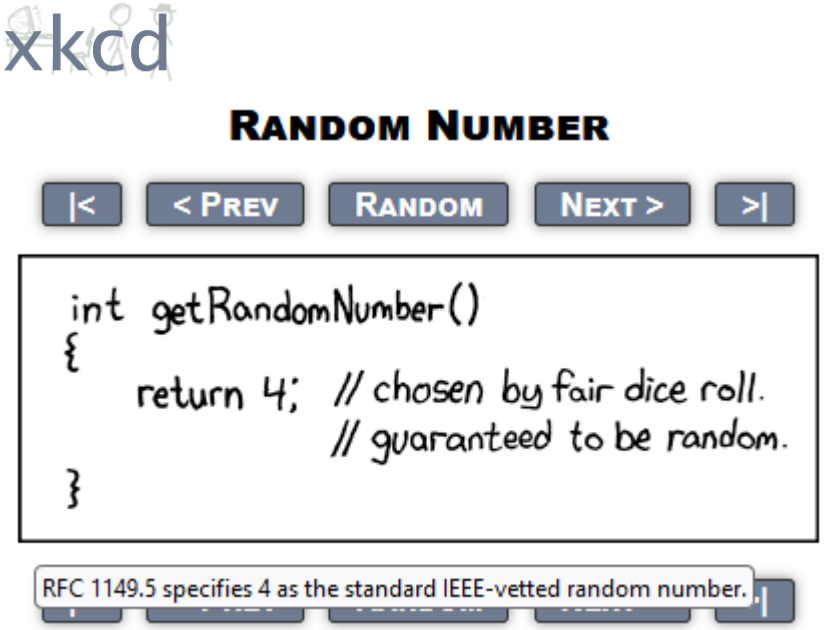
- Randomness

In information theory, the **entropy** of a random variable is the average level of "information", "surprise", or "uncertainty" inherent to the variable's possible outcomes. Given a discrete random variable X, which takes values in the alphabet $\mathcal{X}$ and is distributed according to $p : \mathcal{X} \to [0, 1]$:

$$\mathrm{H}(X) := -\sum_{x \in \mathcal{X}} p(x) \log p(x) = \mathbb{E}[-\log p(X)],$$

where Σ denotes the sum over the variable's possible values. The choice of base for log, the logarithm, varies for different applications. Base 2 gives the unit of bits (or

Quantities of the form $H = -\sum p_i \log p_i$ (the constant K merely amounts to a choice of a unit of measure) play a central role in information theory as measures of information, choice and uncertainty. The form of H will be recognized as that of entropy as defined in certain formulations of statistical mechanics[8] where p_i is the probability of a system being in cell i of its phase space. H is then, for example, the H in Boltzmann's famous H theorem. We shall call $H = -\sum p_i \log p_i$ the entropy of the set of probabilities $p_1 \ldots p_n$. If x is a



https://xkcd.com/221/
https://en.wikipedia.org/wiki/Entropy_(information_theory)
https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf

Shannon

Reprinted with corrections from *The Bell System Technical Journal*,
Vol. 27, pp. 379–423, 623–656, July, October, 1948.

A Mathematical Theory of Communication

By C. E. SHANNON

INTRODUCTION

THE recent development of various methods of modulation such as PCM and PPM which exchange bandwidth for signal-to-noise ratio has intensified the interest in a general theory of communication. A basis for such a theory is contained in the important papers of Nyquist[1] and Hartley[2] on this subject. In the present paper we will extend the theory to include a number of new factors, in particular the effect of noise in the channel, and the savings possible due to the statistical structure of the original message and due to the nature of the final destination of the information.

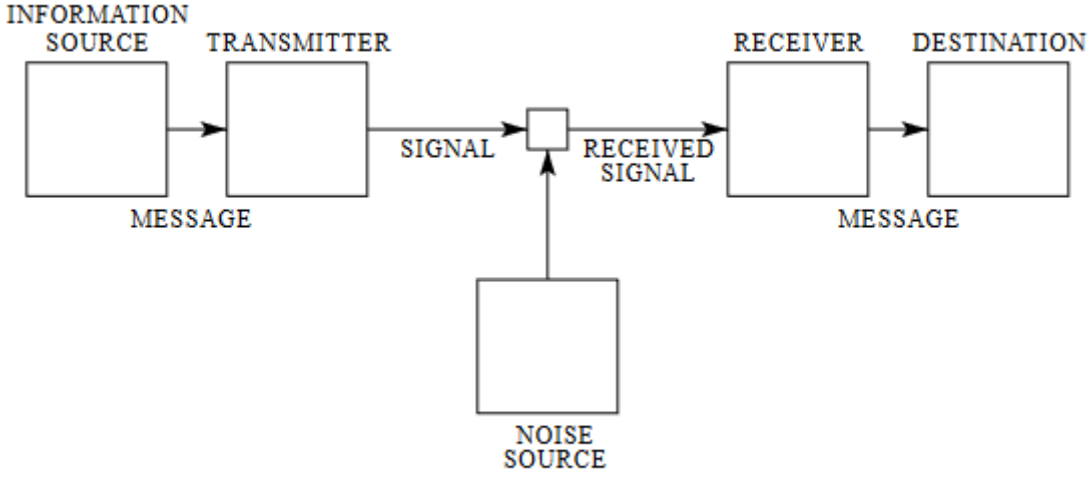


Fig. 1—Schematic diagram of a general communication system.

https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf

Shannon

The entropy in the case of two possibilities with probabilities p and $q = 1 - p$, namely

$$H = -(p \log p + q \log q)$$

is plotted in Fig. 7 as a function of p.

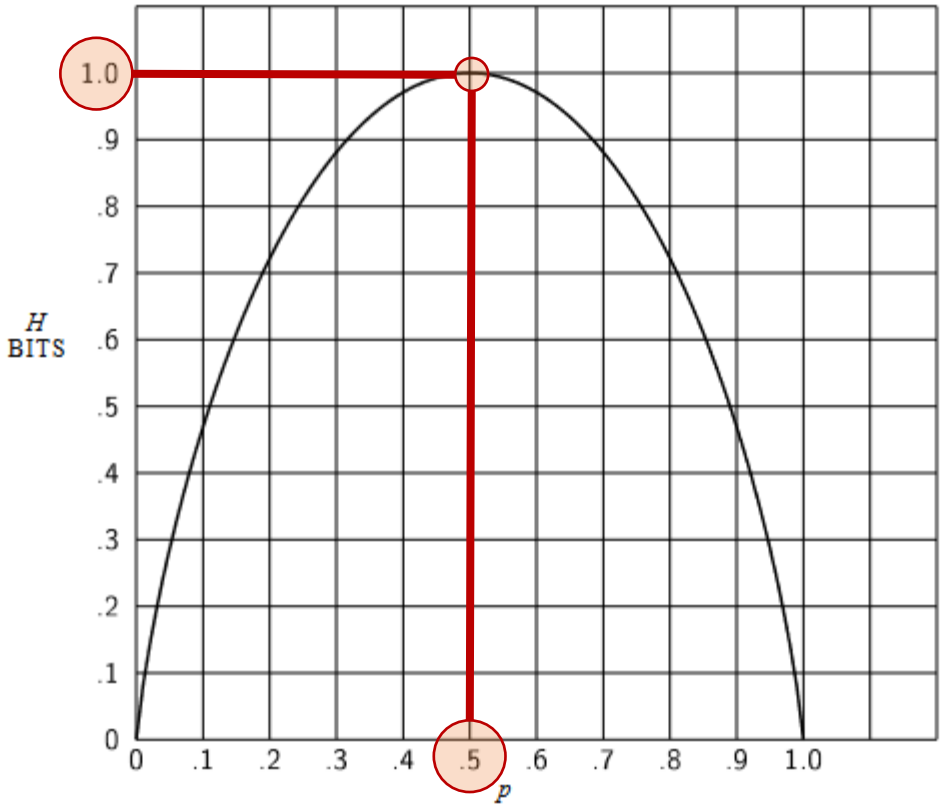


Fig. 7—Entropy in the case of two possibilities with probabilities p and $(1 - p)$.

The quantity H has a number of interesting properties which further substantiate it as a reasonable measure of choice or information.

1. $H = 0$ if and only if all the p_i but one are zero, this one having the value unity. Thus only when we are certain of the outcome does H vanish. Otherwise H is positive.

2. For a given n, H is a maximum and equal to $\log n$ when all the p_i are equal (i.e., $\frac{1}{n}$). This is also intuitively the most uncertain situation.

The Point

- For cryptography we need a good source of entropy
  - when creating keys, etc.

FYI

XKCD reference to RFC 1149.5

- RFC 1149, “A Standard for the Transmission of IP Datagrams on Avian Carriers”; https://www.rfc-editor.org/rfc/rfc1149.txt

Elliptic-Curve Diffie-Hellman (ECDH)

Alice and Bob agree ahead of time to use:

1) **Curve25519** for their key agreement elliptic curve.
   - This gives them common parameters (G, p).
   - $p = 2^{255}-19$ (prime number; this is where the name comes from Curve25519)
2) A specific hash function (say SHA512/256)

ALICE

Generate random number, is it as private key: K_A

Calculate public key: $P_A = K_A \bullet G \pmod p$

Alice knows K_A and got P_B from Bob

Multiply to get shared secret = $K_A \bullet K_B \bullet G \pmod p$

Key, S_{AB} = hash(shared secret)

BOB

Generate random number, is it as private key: K_B

Calculate public key: $P_B = K_B \bullet G \pmod p$

Bob knows K_B and got P_A from Alice

Multiply to get shared secret = $K_A \bullet K_B \bullet G \pmod p$

Key, S_{AB} = hash(shared secret)

References

Curve25519: new Diffie-Hellman speed records
https://cr.yp.to/ecdh.html
https://cr.yp.to/ecdh/curve25519-20060209.pdf

RFC for Curve25519
https://www.rfc-editor.org/rfc/rfc7748

Christof Paar (part of an entire semester course on cryptography)
https://www.youtube.com/watch?v=vnpZXJL6QCQ

Robert Pierce; Math intensive
https://www.youtube.com/watch?v=F3zzNa42-tQ

Bill Buchanan; Code in Python
https://www.youtube.com/watch?v=o9AdiGjOb_I

The Point

- Exchange public information
- Eve can observe complete exchange
- Arrive at a key that only Alice and Bob know

- One Hard Part: Knowing I have Alice's public key
  - Perhaps Eve is pretending to be Alice…

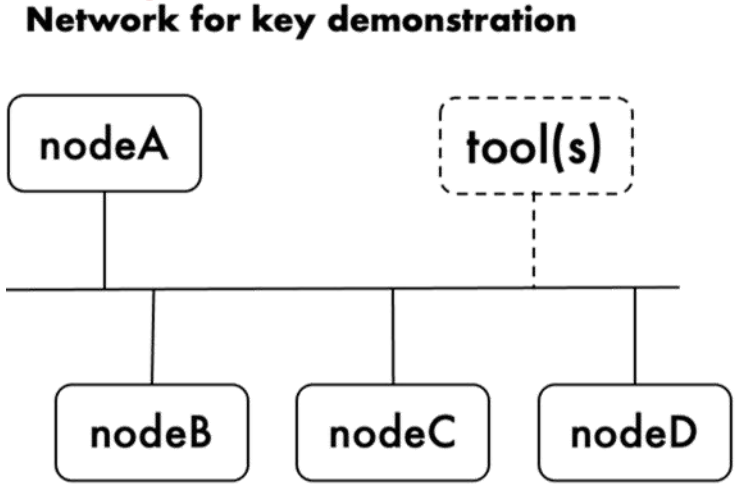
Key Derivation Function (KDF)

- After doing the heavy lifting to get a share key, S_N.
- Use S_N to derive a new key for each “session”

- **S_V = KDF(S_X, entropy || role || other info…)**

Network Configuration

Simple Network for this Lab

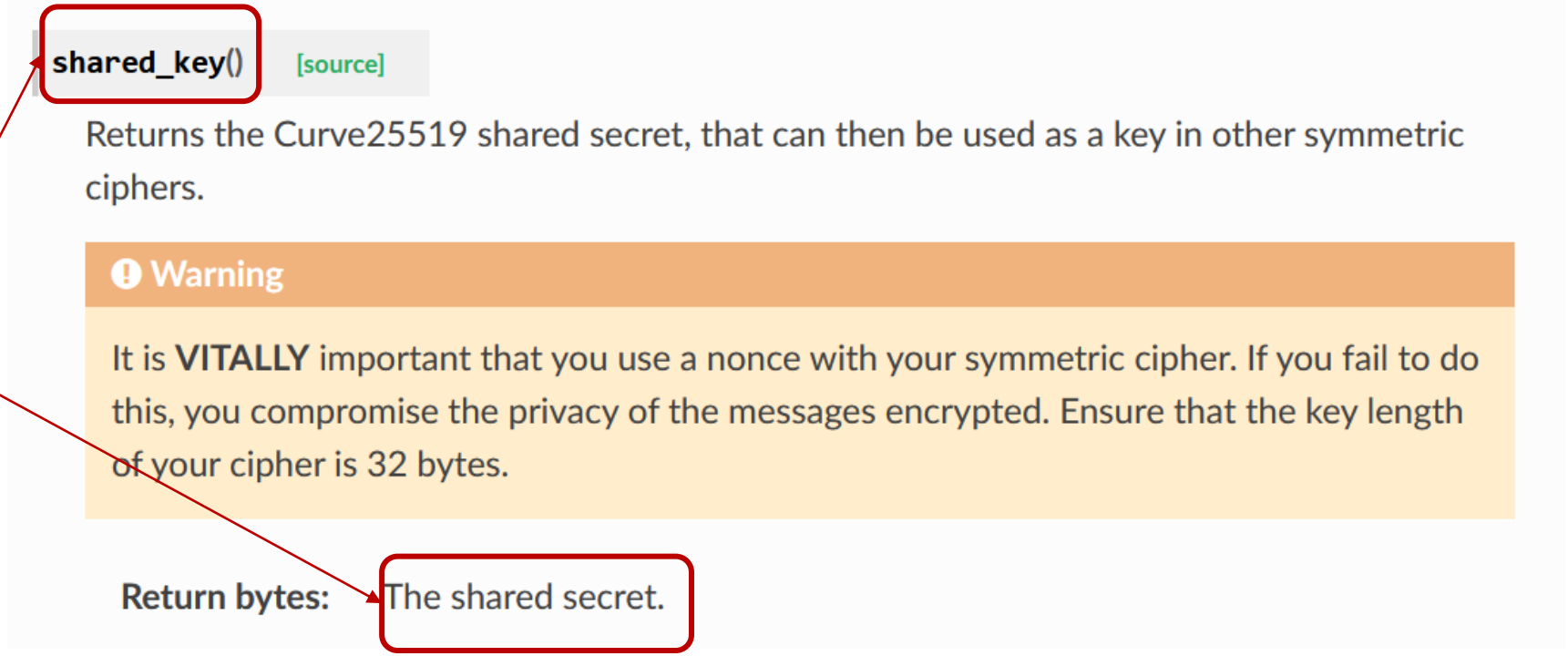


1Hz : 1 Hz generators of J1939 messages
MitM : Man-in-the-Middle
ECU : ECU added (& controlled) by student
TIC : Text Instrument Cluster
tool(s): One or more of can-utils (canplayer, candump, cansniffer, etc.)

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

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, Curve25519(b, 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

NaCl implement Curve25519

- 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

entropy

key derivation

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 and y will be entropy ('nonces' in this case)

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

# z is the conatenation of all nonces
zb = xb + yb
print(zb.hex(" ", 4))           # example: 61e015f7 920a8f0a 13a56ccd bed1a7c8

# Sn is the long-lived base key
Sn = bytes.fromhex("00000000 11111111 22222222 33333333")

# combine base with nonces (order matters! all participants mush use the same order)
keymaterial = Sn + zb
print(keymaterial.hex(" ", 4))  # example: 00000000 11111111 22222222 33333333 61e015f7 920a8f0a 13a56ccd bed1a7c8

# use the key material to create a new key, Sv
kdf = ConcatKDFHash(algorithm=hashes.SHA256(), length=16, otherinfo=b"00000000")
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.

- Order of entropy is **critical** to get common S_V.