SIGPwny

General FA2025 • 2025-10-16

Cryptography I

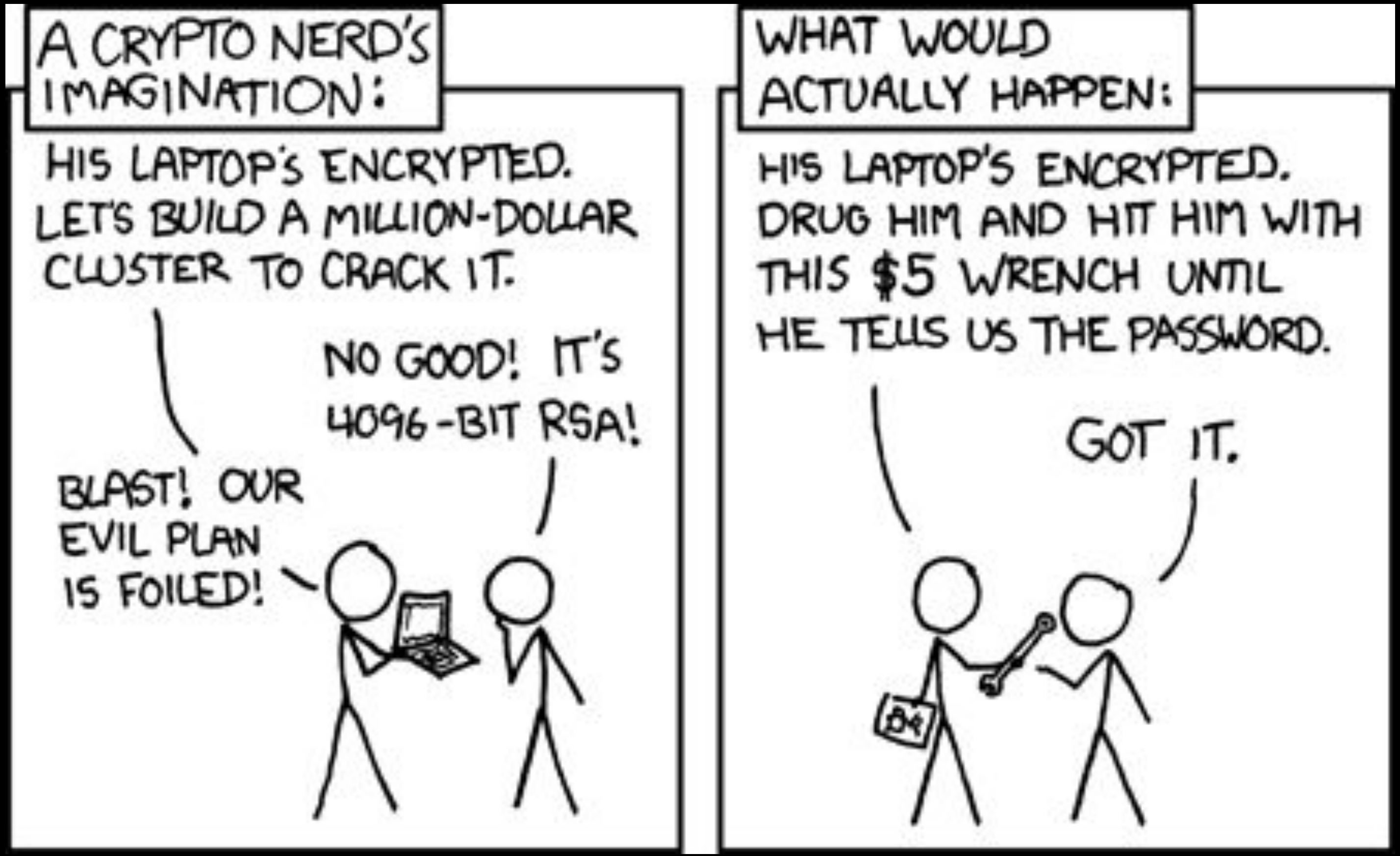
Ryan Yin, Nikhil Date

Announcements

- We will be playing Hack.lu CTF on **Friday** at **6:00 PM** in Siebel CS 2406 (room may change)!
  - There will be free food!

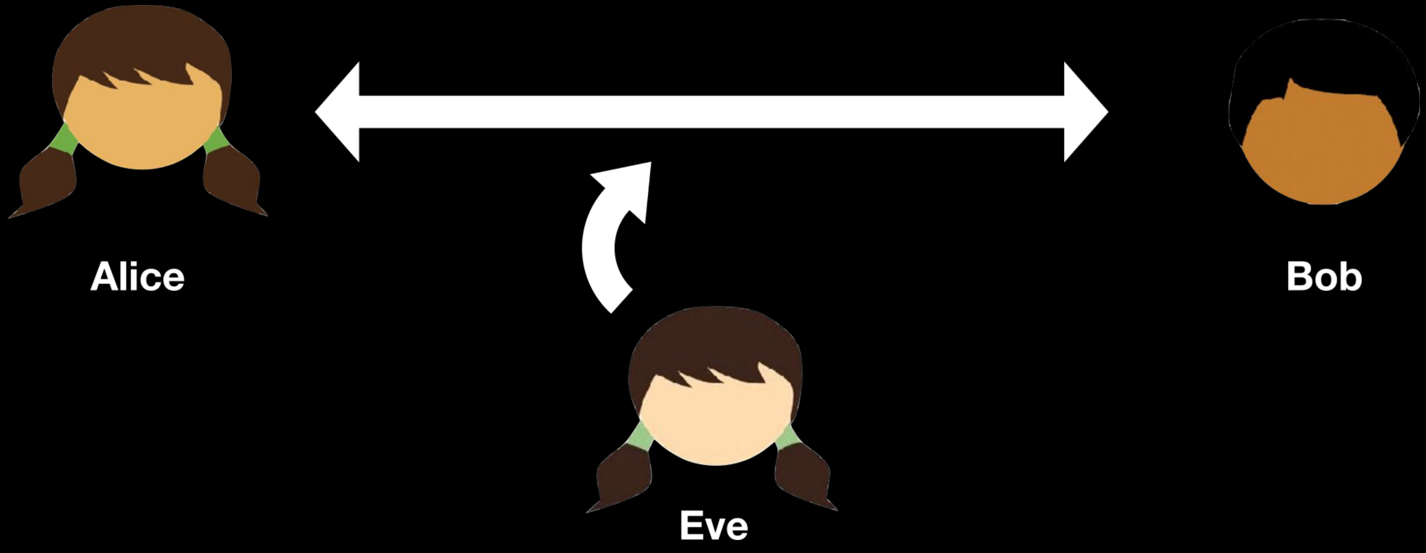
ctf.sigpwny.com

sigpwny{a1ic3_4nd_b0b}



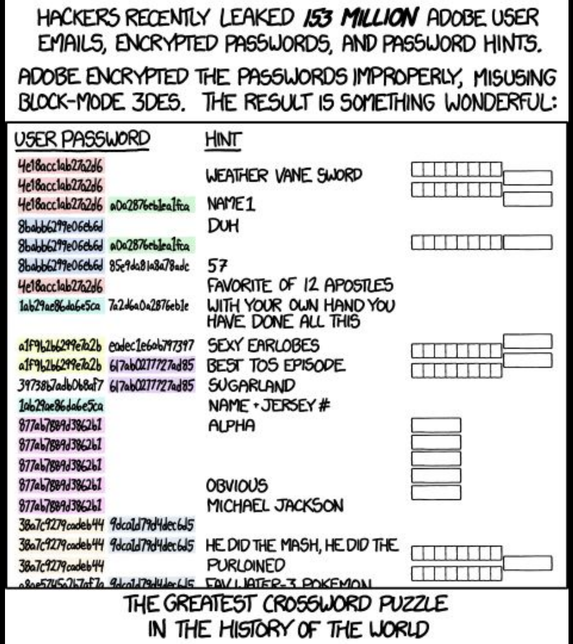
What is Cryptography all about?

- Secure communication between 2+ parties (Alice, Bob)
- Ensure that your "information" is safe from "threats".



Consequences of bad cryptography

- Mary Queen of Scots executed for conspiring to kill Queen Elizabeth I (Babington Plot)
- Vulnerabilities in OpenSSH (e.g. CVE-2008-0166) give an attacker a free shell on your system
- Adobe password breach (unsalted passwords exposed)
- PlayStation 3 Console ECDSA key recovery



Then vs. now

- Cryptanalysis done manually by spymasters, generally very targeted (e.g. military use)
    - Schemes were secure until they weren't
    - Security by obscurity "ok"
- Current day: your computer send millions of encrypted packets to tens of thousands of hosts
- We need schemes predicated on computational hardness assumptions (if these assumptions hold, this scheme is secure to these categories of attacks)
- To implement cryptographic protocols, we use primitives treated as unbreakable and problems that are considered "hard".

Substitution ciphers

- Caesar Cipher (a.k.a. `rot13`, hint for Vim users: `:h g?`)
    - Add 13 to every letter in the alphabet (with wraparound)
    - Ex. CAESAR -> PNRFNE
- Generally, any function that maps each letter to another letter
- **Insecure!!** Why?
- Cryptanalysis
    - Frequency analysis
    - Known plaintext (cribs): “Keine besonderen Ereignisse” (nothing to report)
    - Only 25 keys for Caesar Cipher, so we can try them all

Data Representation

```
>>> from Crypto.Util.number import long_to_bytes
>>> long_to_bytes(3735928559) # integer
b'\xde\xad\xbe\xef'
>>> base64.b64decode(b'3q2+7w==') # base64
b'\xde\xad\xbe\xef'
>>> bytes.fromhex("deadbeef") # hex string
b'\xde\xad\xbe\xef'
```

XOR

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

A.k.a. addition mod 2

Associative, commutative, self-inverse

The one-time pad

```
>>> plain = b"Test"
>>> cipher = bytes.fromhex("cafebabe")
>>> bytes([i ^ j for i, j in zip(cipher, plain)])
b'\x9e\x9b\xc9\xca'
```

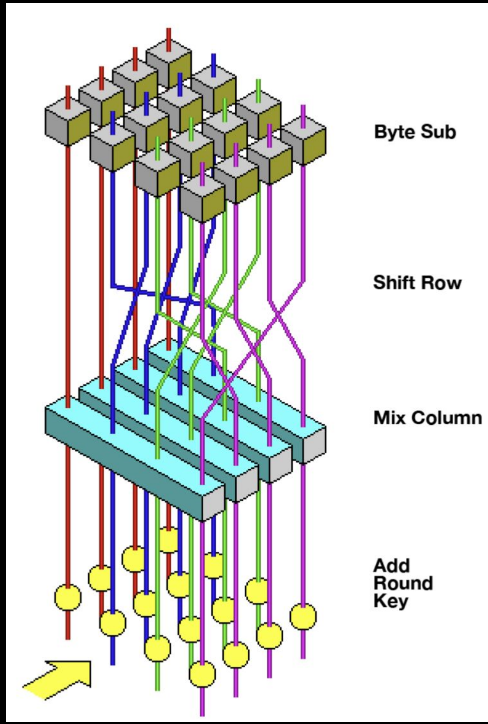
What are we doing here?

The one-time pad

- Achieves “perfect secrecy”! 🥳
    - …but at what cost?
- Requires a completely random bitstring the same length of your plaintext
    - Repeating your pad or having non-random pads defeats the purpose
    - Not only does this double the message size, but how do you agree on this shared secret?
    - Pseudorandom generators can “stretch” a little bit of randomness into a lot of randomness
    - Stay tuned for AES in crypto III…

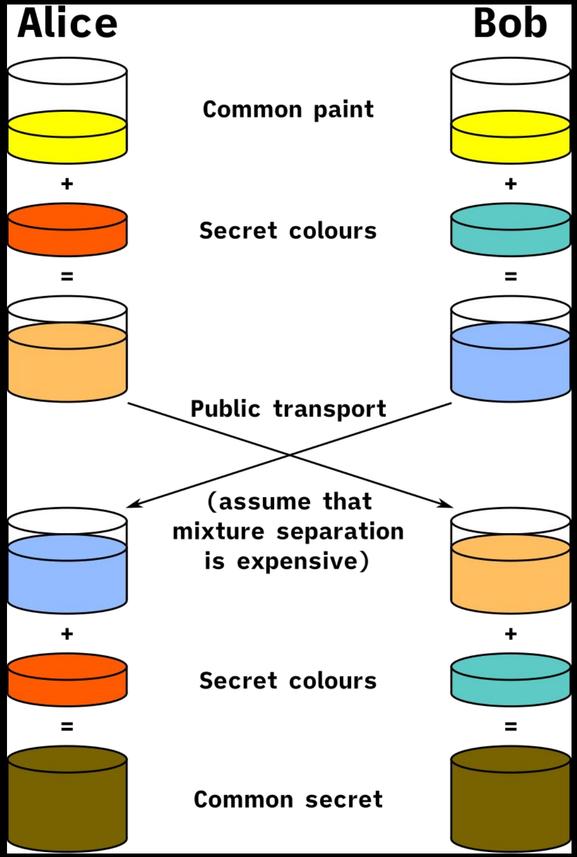
Symmetric Encryption

- Encryption where both parties know some shared key in advance.
- Encryption scrambles the input using some property of the key
- Decryption is simply encryption in reverse.
- Security property is chosen plaintext security
  - Even after the encryptions for many ciphertexts are revealed, the attacker still can't guess the encryption for a plaintext they haven't seen yet



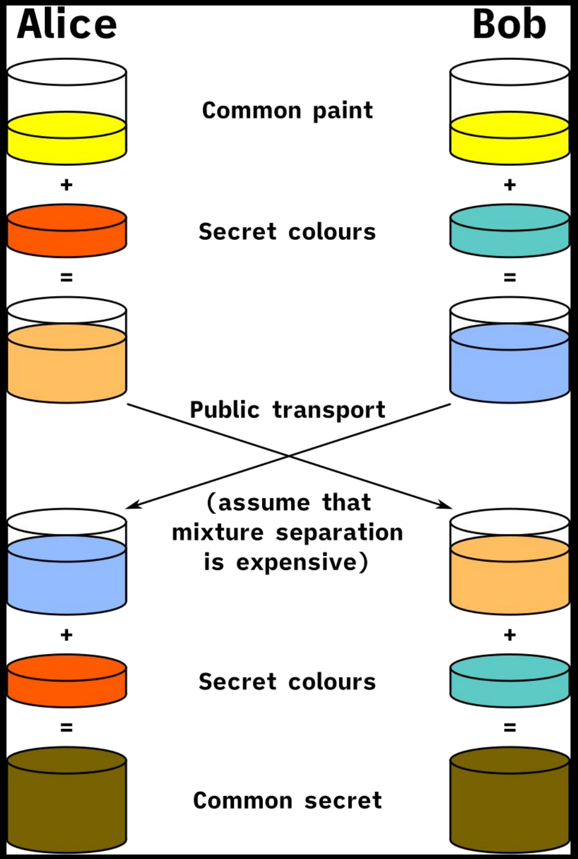
Diffie-Hellman

- Alice and Bob arrive at a shared secret using their private secrets
- All communication happens over a public channel
- Modern implementations perform computations over elliptic curves (ECDH)



Diffie-Hellman

- Finite-Field Diffie-Hellman relies on the fact that it's hard to compute a^bc mod p given a^b mod p and a^c mod p
- Each side generates a secret value x, then sends a^x mod p
- Then the value is exponentiated on each side using the secret value.
- Both end up with a^bc mod p
- Pick p such that it is a large prime number
- Pick a as a primitive root mod p



Computational hardness

- We cannot actually prove that these are hard, but they are strongly believed to be hard
  - This assumption turns out to be false for quantum computers, which is why people want to build quantum computers
- Discrete log/factoring problem
  - Exponentiation is easy, logarithms are hard
  - RSA relies on a similar premise, but the hard problem is factoring

$$a^b \equiv X \bmod p$$

Tools

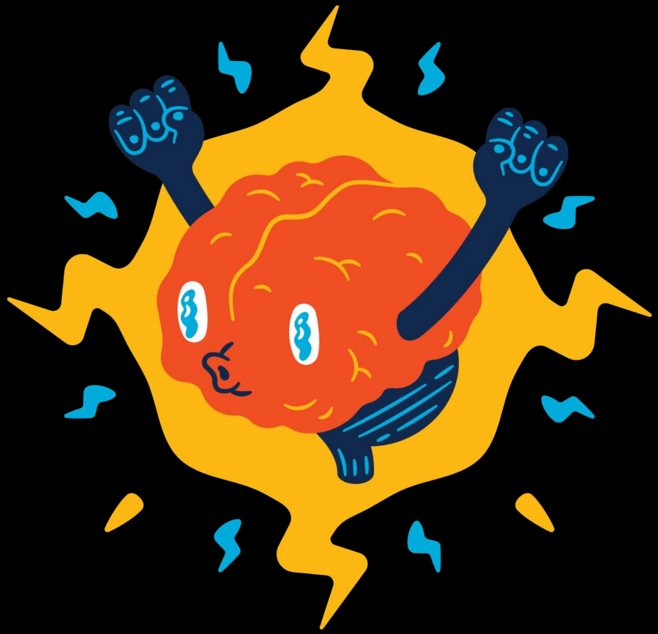
- Pen and paper
- Wikipedia
- Stack Exchange
- SageMath, PyCryptodome, pwntools

```
from sage.all import *
from pwn import *

conn = remote('localhost', 1337)

a = int(conn.recvline()[3:].decode('utf-8'))
b = int(conn.recvline()[3:].decode('utf-8'))
sol = a.powermod(b, p)

conn.recvuntil(b'c = ')
conn.sendline(str(int(sol)).encode('utf-8'))
print(conn.recvline())
```

Food for thought

- How can encryption be done asymmetrically? (RSA, Crypto II)
- How does Alice know she's really talking to Bob? (digital certificates, web of trust, public key infrastructure)
- If you take one thing away from this meeting: **never roll your own crypto!**

CryptoHack



Learn with fantastic lessons and challenges, and earn points on PwnyCTF while you're at it!

ctf.sigpwny.com/challenges#Meetings/CryptoHack

Challenges

- Start with First XOR, flag_format (both XOR-based) and Vigenère Visionary
- Diffie-Hellman god has you do the Diffie-Hellman shared secret computation (look at Wikipedia for implementation details)
- First AES and Add One are based on the “Advanced Encryption Standard (AES)” block cipher

Next Meetings

2025-10-17 • This Friday

- Hack.lu CTF
- Come to Siebel CS 2406 for Hack.lu CTF at 6:00 PM. There will be free food as always!

2025-10-19 • This Sunday

- Pwn II
- Learn about control flow hijacking and format string attacks!

2025-10-23 • Next Thursday

- Cryptography 2
- Topics include Chinese Remainder Theorem and RSA

ctf.sigpwny.com

sigpwny{a1ic3_4nd_b0b}

Meeting content can be found at sigpwny.com/meetings.

SIGPwny