Fun with Information Engineering and Security Summer 2024

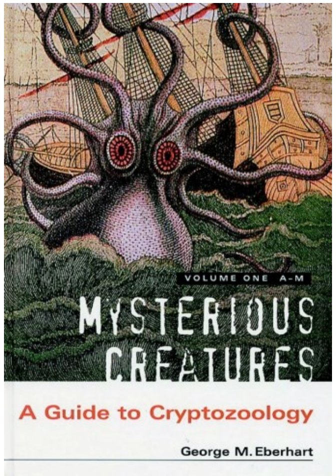
Day 2 (Aug-01, Thur)

Department of Information Engineering

香港中文大學
The Chinese University of Hong Kong

Basic Terminology

- "Crypto", from Greek, means "hidden" or "secret"
- "graph", also from Greek, means "write"
- So literally "cryptography" is to write secret
- Cryptocurrency vs 加密貨幣
  - Hidden currency? What does "加密" mean? Encryption? Or "With cryptography"?
- Cryptozoology
  - The study of legendary/hidden creatures
    - Existence often disputed/unsubstantiated
  - Not the study of encrypted animals



Basic Terminology

- Encryption – scramble data in a way that only authorized parties can understand the information
- A very common (but not the only) way of achieving **confidentiality**
- The encryption output usually looks “random” and unintelligible

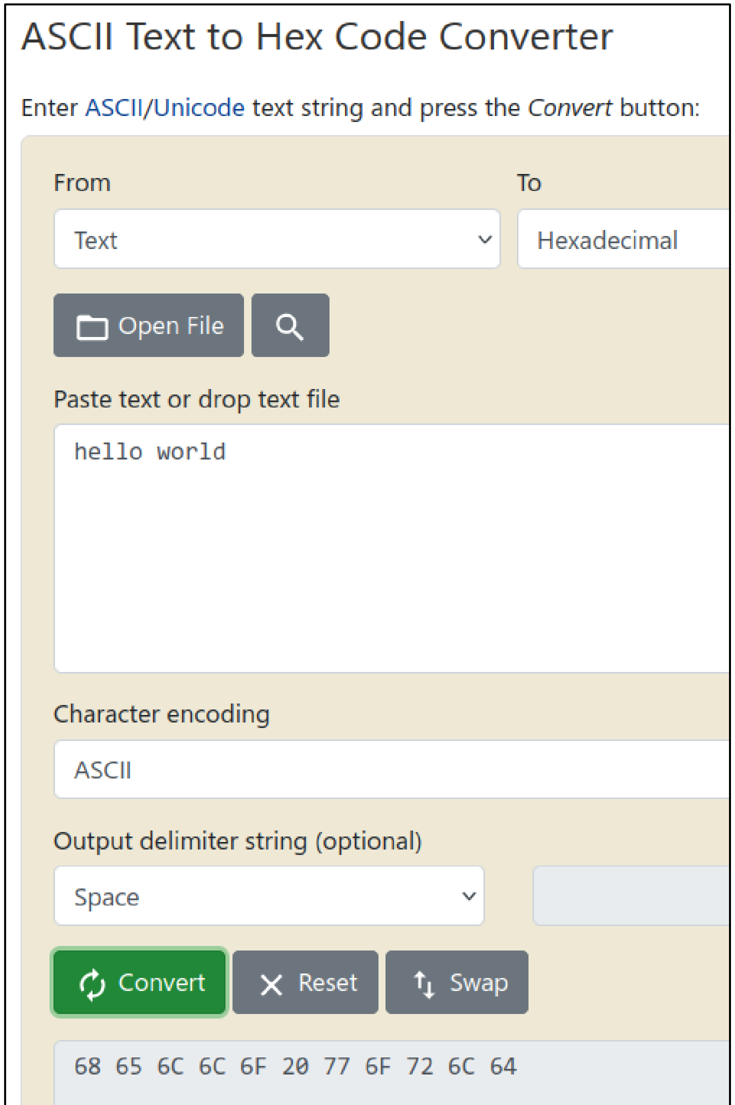
Basic Terminology

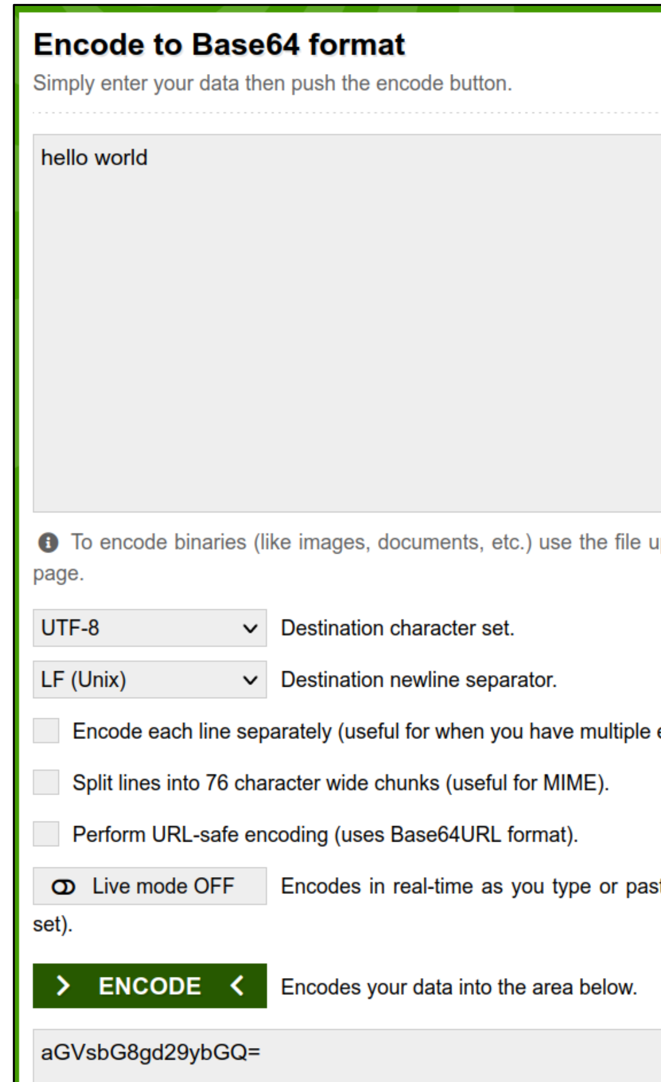
- Encryption in general can be classified based on the number of keys involved in a scheme

0. No keys (wrong!)
   - That's **not** proper encryption, more like encoding
1. "Secret key" crypto (a.k.a symmetric key crypto)
   - Communication parties all share the same key
2. "Public key" crypto (a.k.a asymmetric key crypto)
   - Each party have a pair of keys

These are not encryption

No keys, known (public) algorithms





Basic Terminology

- Cipher – Algorithms for Encryption & Decryption
- Plaintext – Original Message
- Ciphertext – Transformed (encrypted) Message
- (Secret) Key – Secret used in the transformations
- Correctness – Get the message back from ciphertext (encrypted under key k), using decryption under the same key k
  - $D_k(E_k(x)) = x$
  - We want to transform, not “destroy” the message

Shift Cipher

- Cryptanalysis
  - Can an attacker find K?
    - Yes, by a bruteforce attack (do an exhaustive key search)
      - Because key space is small (26 possible keys)
- Lessons learnt:
    - Key space needs to be large enough

Mono-alphabetic Substitution Cipher

- Key space: all permutations of {A, B, C, ..., Z}
  - each **key is** an invertible **mapping**
- For each letter x in plaintext P, $Enc_k(x)$:
  - replace x with k(x)
- For each letter y in ciphertext C, $Dec_k(y)$:
  - replace y with $k^{-1}(y)$

Example:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
k=	B	A	D	C	Z	H	W	Y	G	O	Q	X	S	V	T	R	N	M	L	K	J	I	P	F	E	U

BECAUSE → AZDBJLZ

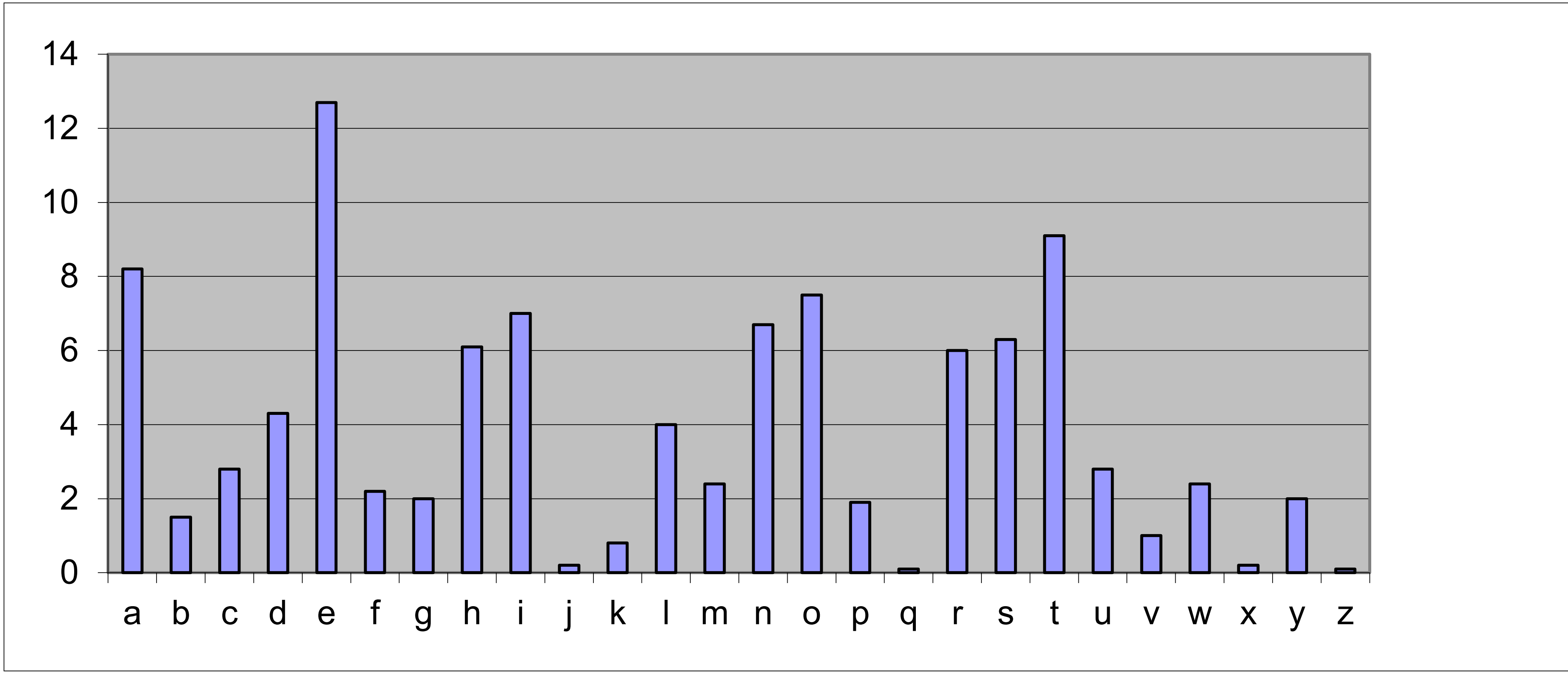
Mono-alphabetic Substitution Cipher

- Now exhaustive search is difficult
  - key space has a size of $26! \approx 2^{88}$
- Thought to be unbreakable for many years
- How to break it?
- Use features of the plaintext!

Frequency Analysis

- Each language has certain features:
  - Frequency of letters/groups of 2+ letters
- Substitution ciphers generally preserve such features
- Hence they’re susceptible to frequency analysis attacks

Frequency of Letters in English

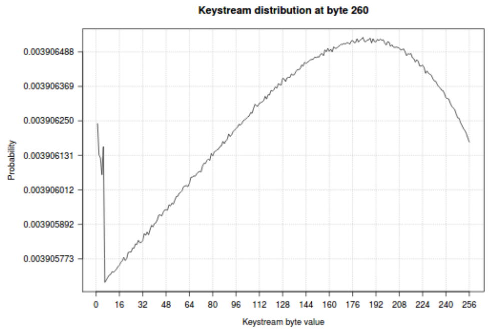


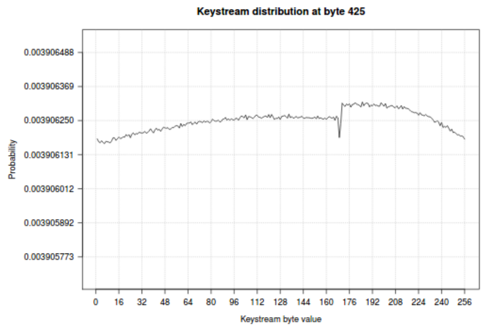
Stream Ciphers:

- Idea: stretch a short “random” key into a long enough “pseudorandom” key
- Use a PRNG: $\{0, 1\}^s \rightarrow \{0,1\}^n$ $n >> s$
  - Deterministic algorithm to expand a short (e.g., 128-bit) random seed into a long enough key that “looks random”
- Secret key is the seed
- $E_k(m) = m \oplus PRNG(k)$, $D_k(c) = c \oplus PRNG(k)$

PRNG and Stream Cipher

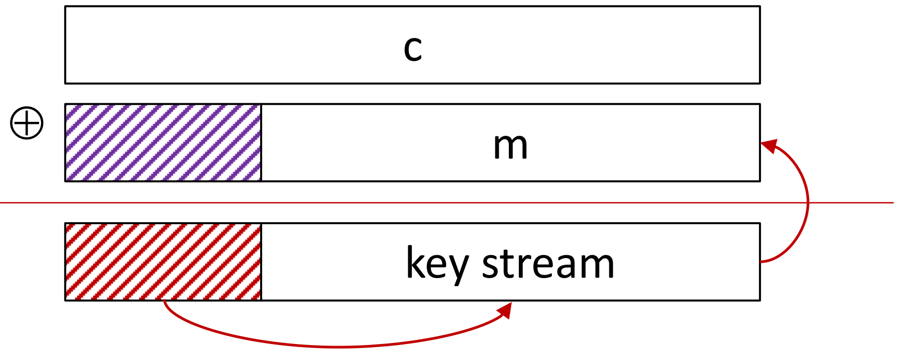
- Security of a stream cipher depends on its PRNG
  - Some PRNG are weak: knowing some amount of output bit sequence, can **recover seed** (key)
    - DO NOT use such PRNG to build stream ciphers, otherwise might lead to **key recovery attacks**
  - Some are thought to be cryptographically secure, but turns out to be biased
    - E.g., RC4



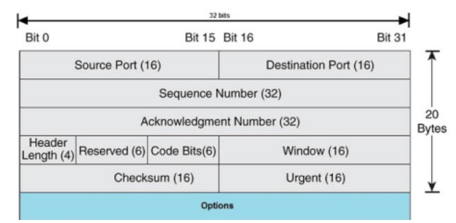


PRNG and Stream Cipher

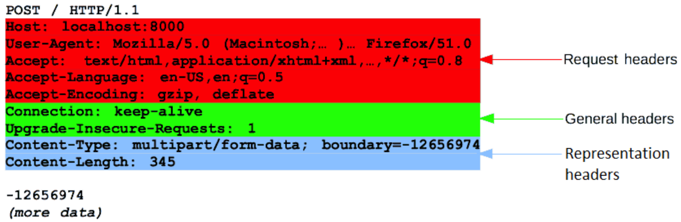
- Security of a stream cipher depends on its PRNG
  - Want PRNG to generate **unpredictable** sequences
    - Given consecutive sequence of output bits (but not the seed), the next bit must be hard to predict
  - Otherwise



$G(k)|_{1,..,i} \rightarrow G(k)|_{i+1}$

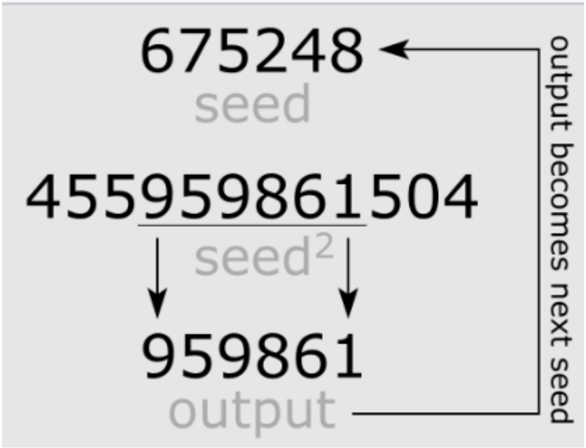


e.g., fields in message/packet headers might be easy to figure out (^ TCP, v HTTP)



Examples of weak PRNGs

- Do not use these for cryptographic needs



Middle-square method

Linear Congruential Generator (LCG) w/ parameters a, b, p

```
R[i] = a*R[i-1]+b mod p
```

R is the sequence of PRN

R[0] = seed

`random()` in glibc is a variant of LCG

$R[i] = R[i-3]+R[i-31] \bmod 2^{32}$

```
output R[i] >> 1
```

Examples of Stream Ciphers

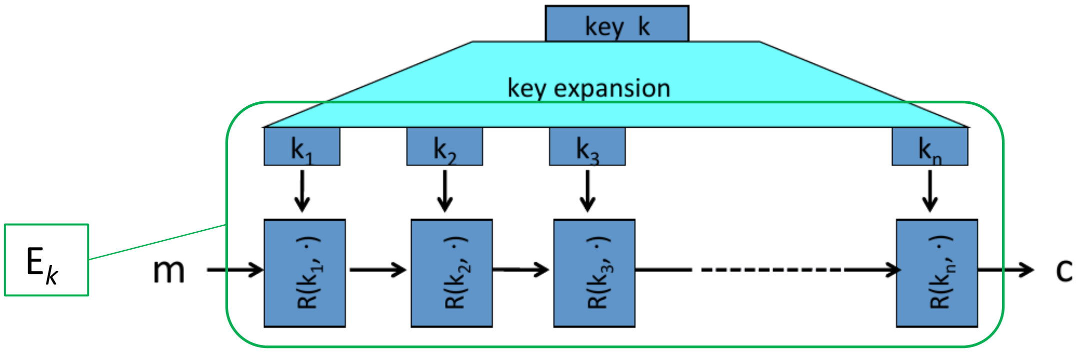
- RC4: broken, CSS (DVD): broken
- Salsa20 (and ChaCha) has shown good potential
  - Android's Google services sometimes use ChaCha

Why Block Ciphers?

- Remember how we got here?
- We were trying to **defeat frequency analysis**
  - Use different key value in different position
    - Example: stream ciphers
  - Another way: make the **unit of transformation** larger, rather than encrypting letter by letter, encrypting block by block
    - Example: **block** ciphers

Block Ciphers

- An n-bit plaintext (block) is encrypted to an n-bit ciphertext
  - $\mathcal{P}$: $\{0,1\}^n$
  - $\mathcal{C}$: $\{0,1\}^n$
  - $\mathcal{K}$: $\{0,1\}^s$
  - **E**: $\mathcal{K} \times \mathcal{P} \rightarrow \mathcal{C}$: E_k: a Pseudo Random Permutation on $\{0,1\}^n$
  - **D**: $\mathcal{K} \times \mathcal{C} \rightarrow \mathcal{P}$: D_k is E_k^{-1}
  - Block size: n
  - Key size: s



R(k,m) is called a round function

How to defeat frequency analysis – Block Cipher style

- Diffusion
  - Substitution is done in a way that changing 1 bit in the plaintext will propagate to as many ciphertext bits as possible
- Confusion
  - Each bit of the key will affect as many bits as possible of the output ciphertext block
- These are the 2 cornerstones of block cipher designs
  - Also referred to as the “avalanche effect”

Data Encryption Standard (DES)

- Designed by IBM, with modifications proposed by the National Security Agency, US national standard from 1977 to 2001
- Block size is 64 bits, Key size is 56 bits, Has 16 rounds
- Good diffusion: on average 1 bit change to the input block affects 34 bits of the output block
- Good confusion: on average 1 bit change to the key affects 35 bits of the output block
- Designed mostly for hardware implementations
  - Software implementation is somewhat slow
- Considered insecure now
  - Vulnerable to brute-force attacks
  - Mainly due to its short key size

DES challenge

msg = "The unknown messages is: XXXX … "

CT = c_1 c_2 c_3 c_4

Goal: find $k \in \{0,1\}^{56}$ s.t. $DES(k, m_i) = c_i$ for $i=1,2,3$

1997: Internet search -- **3 months**

1998: EFF machine (deep crack) -- **3 days** (250K $)

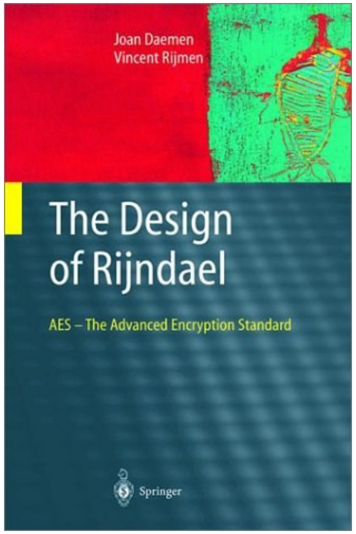
1999: combined search -- **22 hours**

2006: COPACOBANA (120 FPGAs) **-- 7 days** (10K $)

⇒ 56-bit ciphers should not be used !!

AES Features

- Designed to be efficient in both hardware and software across a variety of platforms.
- Block size: 128 bits
- Variable key size: **128, 192, or 256 bits.**
- No known design weaknesses to this date
  - But there are occasionally some implementation and deployment issues

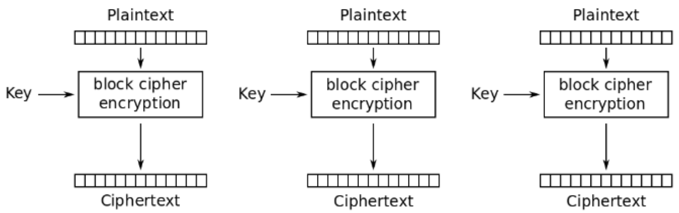


Putting block ciphers into good use

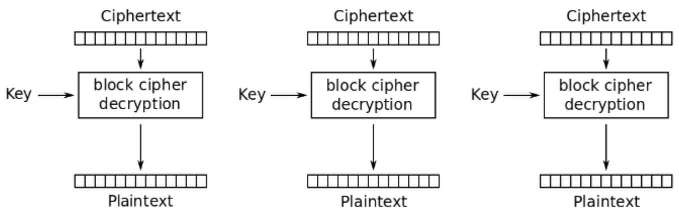
- A block cipher encrypts only **one block**
- Need a way to extend it to encrypt **arbitrarily long messages** (more useful)
  - Block cipher **modes of operation**
    - There are many modes in practice, but for simplicity we will talk about 2 here.

Block Cipher Operation Modes: ECB

- **Electronic Code Book (ECB)**: each block encrypted (and decrypted) separately
  - Message is broken into independent blocks
  - $X = x_0 \,||\, x_1 \,||\, x_2 \,||\, \ldots \,||\, x_n$
  - $C = c_0 \,||\, c_1 \,||\, c_2 \,||\, \ldots \,||\, c_n$
  - Encryption: $c_i = E_k(x_i)$
  - Decrytion: $x_i = D_k(c_i)$
- Both E & D are parallelizable
  - No data dependencies betwe blocks

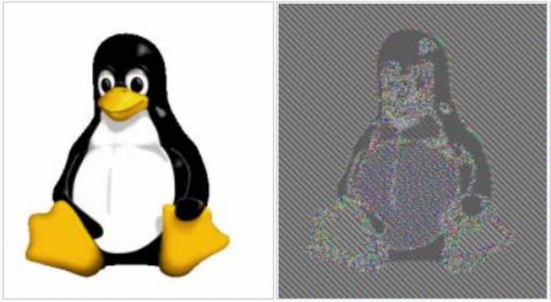


Electronic Codebook (ECB) mode encryption



Electronic Codebook (ECB) mode decryption

Properties of ECB



Original image   Encrypted using ECB mode

- Deterministic:
  - the same data block gets encrypted the same way
    - reveals patterns of data when a data block repeats
    - can think of this as "frequency feature" at the block level
  - when the same key is used, the same message is encrypted the same way
- Usage: not recommended to encrypt more than one block of data

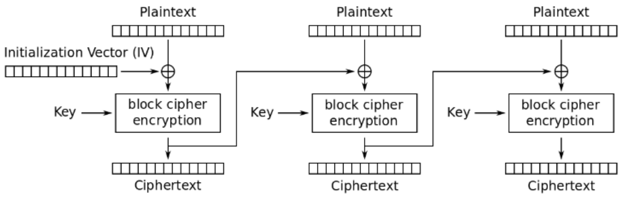
Block Cipher Operation Modes: CBC

- Cipher Block Chaining (CBC):
  - Uses a random Initial Vector (IV)
  - Next input depends upon previous output

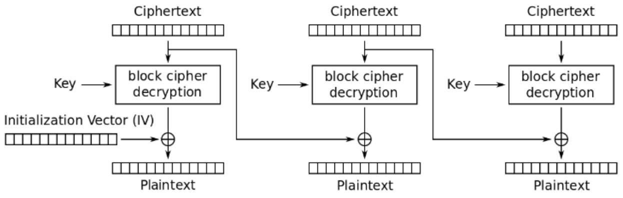
Encryption: $C_i = E_k(M_i \oplus C_{i-1})$, **with** $C_0 = IV$

Decryption: $M_i = C_{i-1} \oplus D_k(C_i)$, **with** $C_0 = IV$

not both E & D are parallelizable



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Properties of CBC

- Randomized encryption: repeated input blocks will be mapped to different ciphertext blocks

- Usage: chooses **random** IV
  - Note: the IV is not secret (it is part of ciphertext)
  - Thus the ciphertext will be at least 1 block longer than the plaintext

Some block cipher modes of operation need padding

- Recall that a block cipher on its own deals with transforming (en/decryption) 1 block of input

- What if size(msg) is **not a multiple** of size(block)?
  - Well, we can add some number of **padding** bytes to make size(msg) = k * size(block)

Introducing PKCS#7 **padding**

- Pad input with x bytes of hex value x to make the total size a multiple of size(block)
  - e.g. size(block) = 8

```
… … | DD DD DD DD DD DD DD DD | DD DD DD DD 04 04 04 04 |
```

- What if size(msg) is already a multiple of size(block)?
  - Can we use no padding in this case?
  - No, because that’d be **ambiguous**
    - how would the decrypting side know size(padding) = 0?
    - the last byte of payload might be misrecognized as padding

Introducing PKCS#7 **padding**

- If size(msg) is already a multiple of size(block), we add a whole block of padding, with x = size(block)
  - e.g. size(block) = 8

```
… … | DD DD DD DD DD DD DD DD | 08 08 08 08 08 08 08 08 |
```

- Mathematically
  - x = size(block) – (size(msg) mod size(block))
- Note: AES has a block size of 16 bytes (128 bits)
- And this is why a CBC ciphertext could be 2 blocks longer than the plaintext message (IV + padding)

AES CBC in practice

- Fortunately, we usually don’t (and shouldn’t) implement our own crypto stuff
  - Think of the development ecosystem as a supply chain; reuse off-the-shelf components
- AES, CBC, and PKCS7 padding are all implemented already as libraries (Python packages)
  - These are what you will use in the challenge
  - Remember read the documentation
    - This is the SOP of many programmers