
"Some speak of
an Armageddon;
A time when humans
will build machines
they neither understand
nor control.
To myself I whisper,
'We already have.'"
- Taylor Swift

Crypto:
Block Ciphers

Independence Under Chosen Plaintext Attack game: IND-CPA

- Eve is interacting with an encryption "Oracle"
  - Oracle has an unknown random key **k**
- She can provide two separate chosen plaintexts of the same length
  - Oracle will randomly select one to encrypt with the unknown key
  - The game can repeat, with the oracle using the same key...
- Goal of Eve is to have a better than random chance of guessing which plaintext the oracle selected
  - Variations involve the Oracle always selecting either the first or the second

Designing Ciphers

- Clearly, the whole trick is in the design of **E(M,K)** and **D(C,K)**
- One very simple approach:
  E(M,K) = **ROTK(M)**; **D(C,K)** = **ROT-K(C)**
  i.e., take each letter in **M** and "rotate" it **K** positions (with wrap-around) through the alphabet
- E.g., $\mathbf{M_i}$ = "DOG", **K** = 3
  $\mathbf{C_i}$ = $\mathbf{E(M_i,K)}$ = **ROT3**("DOG") = "GRJ"
  $\mathbf{D(C_i,K)}$ = **ROT-3**("GRJ") = "DOG"
- "Caesar cipher"
  - "This message has been encrypted twice by ROT-13 for your protection"

Attacks on Caesar Ciphers?

- Brute force: try every possible value of K
  - Work involved?
  - At most 26 “steps”

Attacks on Caesar Ciphers?

- Brute force: try every possible value of K
  - Work involved?
  - At most 26 “steps”
- Deduction:
  - Analyze letter frequencies (“ETAOIN SHRDLU”)
  - Known plaintext / guess possible words & confirm
    - E.g. “JCKN ECGUCT” =?

Attacks on Caesar Ciphers?

- Brute force: try every possible value of K
  - Work involved?
  - At most 26 “steps”
- Deduction:
  - Analyze letter frequencies (“ETAOIN SHRDLU”)
  - Known plaintext / guess possible words & confirm
    - E.g. “JCKN ECGUCT” =? “HAIL CAESAR”

Attacks on Caesar Ciphers?

- Brute force: try every possible value of K
  - Work involved?
  - At most 26 "steps"
- Deduction:
  - Analyze letter frequencies ("ETAOIN SHRDLU")
  - Known plaintext / guess possible words & confirm
    - E.g. "JCKN ECGUCT" =? "HAIL CAESAR" ⇒ K=2
  - Chosen plaintext
    - E.g. Have your spy ensure that the general will send "ALL QUIET", observe "YJJ OSGCR" ⇒ K=24
- Is this IND-CPA?

Kerckhoffs' Principle

- Cryptosystems should remain secure even when attacker knows all internal details
  - Don't rely on security-by-obscurity
- Key should be only thing that must stay secret
- It should be easy to change keys
  - Actually distributing these keys is hard, but we will talk about that particular problem later.
  - But key distribution is one of the real...

Better Versions of Rot-K ?

- Consider **$E(M,K)$ = Rot-{K_1, K_2, …, K_n}(M)**
  - i.e., rotate first character by **K_1**, second character by **K_2**, up through nth character. Then start over with **K_1**, ...
  - **K** = **{ K_1, K_2, ..., K_n }**
- How well do previous attacks work now?
  - Brute force: key space is factor of $26^{(n-1)}$ larger
    - E.g., n = 7 ⇒ 300 million times as much work
  - Letter frequencies: need more ciphertext to reason about
  - Known/chosen plaintext: works just fine
- Can change it so that it is a substitution
  - EG, A->C, B->Z, C->F…
  - Can layer substitutions…
- Can go further with "chaining", e.g., 2nd permutation depends on **K_2** and first character of ciphertext
  - We just described 2,000 years of cryptography

And Cryptanalysis: ULTRA

- During WWII, the Germans used ***enigma***:
  - System was a "rotor machine": A series of rotors, with each rotor permuting the alphabet and every keypress incrementing the settings
    - Key was the selection of rotors, initial settings, and a permutation plugboard
    - A great graphical demonstration: https://observablehq.com/@tmcw/enigma-machine
- The British built a system (the "Bombe") to brute-force Enigma
  - Required a known-plaintext (a "crib") to verify decryption: e.g. the weather report
  - Sometimes the brits would deliberately "seed" a naval minefield for a chosen-plaintext attack

One-Time Pad

- Idea #1: use a different key for each message **M**
  - Different = completely independent
  - So: known plaintext, chosen plaintext, etc., don't help attacker
- Idea #2: make the key as long as **M**
- **E(M,K)** = **M** ⊕ **K** (⊕ = XOR)

$$X \oplus 0 = X$$
$$X \oplus X = 0$$
$$X \oplus Y = Y \oplus X$$
$$X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z$$

⊕	**0**	**1**
0	0	1
1	1	0

One-Time Pad

- Idea #1: use a different key for each message M
  - Different = completely independent
  - So: known plaintext, chosen plaintext, etc., don't help attacker
- Idea #2: make the key as long as M
- $\mathbf{E(M,K) = M \oplus K}$ ($\oplus$ = XOR)

  $\mathbf{D(C,K) = C \oplus K}$

  $= \mathbf{M \oplus K \oplus K = M \oplus 0 = M}$

$\mathbf{X \oplus 0 = X}$

$\mathbf{X \oplus X = 0}$

$\mathbf{X \oplus Y = Y \oplus X}$

$\mathbf{X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z}$

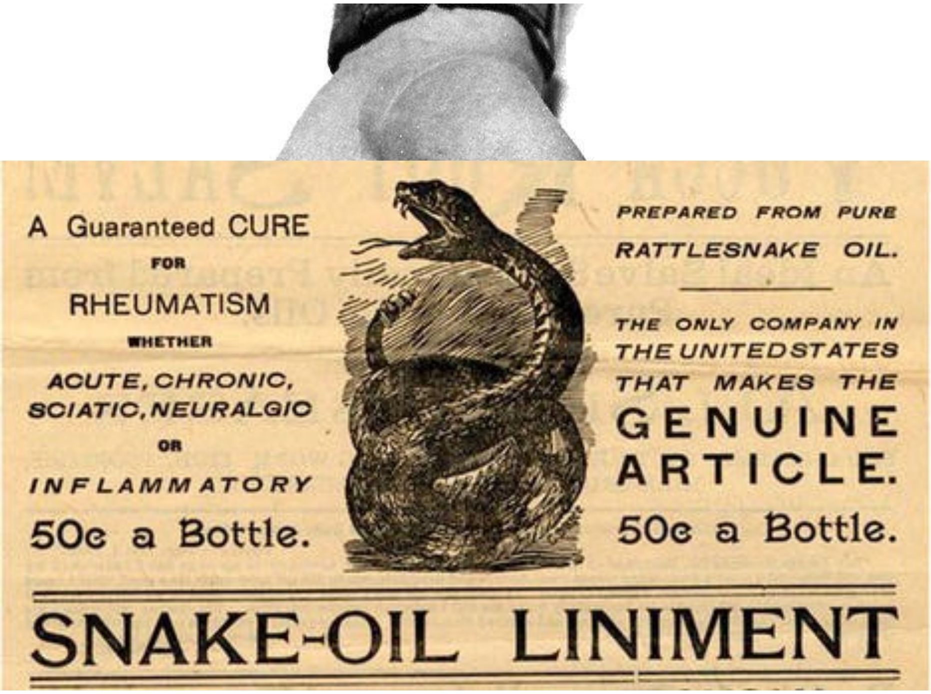
⊕	0	1
0	0	1
1	1	0

One-Time Pad: Provably Secure!

- Let’s assume Eve has partial information about **M**
- We want to show: from **C**, she does not gain any further information
- Formalization: supposed Alice sends either **M'** or **M''**
  - Eve doesn’t know which; tries to guess based on **C**
- Proof:
  - For random, independent **K**, all possible bit-patterns for **C** are equally likely
  - This holds regardless of whether Alice chose **M'** or **M''**, or even if Eve provides **M'** and **M''** to Alice and Alice selects which one (IND-CPA)
  - Thus, observing a given **C** does not help Eve narrow down the possibilities in any way:

One-Time Pad: Provably Impractical!

- Problem #1: key generation
  - Need truly random, independent keys
- Problem #2: key distribution
  - Need to share keys as long as all possible communication
  - If we have a secure way to establish such keys, just use that for communication in the first place!
    - Only advantage is you can communicate the key in advance: you may have the secure channel now but won't have it later



Two-Time Pad?

- What if we reuse a key **K** jeeeest once?
- Alice sends **C** = **E(M, K)** and **C'** = **E(M', K)**
- Eve observes **M** ⊕ **K** and **M'** ⊕ **K**
  - Can she learn anything about M and/or M' ?
- Eve computes **C** ⊕ **C' = (M** ⊕ **K)** ⊕ **(M'** ⊕ **K)**

Two-Time Pad?

- What if we reuse a key K jeeeest once?
- Alice sends **C = E(M, K)** and **C' = E(M', K)**
- Eve observes **M** ⊕ **K** and **M'** ⊕ **K**
  - Can she learn anything about M and/or M' ?
- Eve computes **C** ⊕ **C' = (M** ⊕ **K)** ⊕ **(M'** ⊕ **K)**

  = **(M** ⊕ **M')** ⊕ **(K** ⊕ **K)**

  = **(M** ⊕ **M')** ⊕ **0**

  = **M** ⊕ **M'**
- Now she knows which bits in **M** match bits in **M'**
- And if Eve already knew **M**, now she knows **M'**!
  - Even if not, Eve can guess **M** and ensure that **M'** is consistent

VENONA:
Pad Reuse in the Real World

- The Soviets used one-time pads for communication from their spies in the US
  - After all, it is provably secure!
- During WWII, the Soviets started reusing key material
  - Uncertain whether it was just the cost of generating pads or what...
- VENONA was a US cryptanalysis project designed to break these messages
  - Included confirming/identifying the spies targeting the US Manhattan project
  - Project continued until 1980!
- ***Not declassified until 1995!***
  - So secret ***even President Truman wasn't informed about it***.
  - But the Soviets found out about it in 1949, but their one-time pad reuse was fixed after 1948 anyway

Modern Encryption: Block cipher

- A function **E : {0, 1}^{b} ×{0, 1}^{k} → {0, 1}^{b}**. Once we fix the key **K** (of size **k** bits), we get:
- **EK : {0,1}^{b} → {0,1}^{b}** denoted by **E_K(M) = E(M,K)**.
  - (and also **D(C,K)**, **E(M,K)**'s inverse)
- Three properties:
  - Correctness:
    - **E_K(M)** is a permutation (bijective function) on b-bit strings
      - Bijective ⇒ invertible
  - Efficiency: computable in μsec's
  - Security:
    - For unknown **K**, "behaves" like a random permutation
- Provides a building block for more extensive encryption

DES (Data Encryption Standard)

- Designed in late 1970s
- Block size 64 bits, key size 56 bits
- NSA influenced two facets of its design
  - Altered some subtle internal workings in a mysterious way
  - Reduced key size 64 bits ⇒ 56 bits
    - Made brute-forcing feasible for attacker with massive (for the time) computational resources
- Remains essentially unbroken 40 years later!
  - The NSA's tweaking hardened it against an attack "invented" a decade later
- However, modern computer speeds make it completely unsafe due to small key size

Today's Go-To Block Cipher: AES (Advanced Encryption Standard)

- 20 years old, standardized 15 years ago...
- Block size 128 bits
- Key can be 128, 192, or 256 bits
  - 128 remains quite safe; sometimes termed "AES-128", paranoids use 256b
- As usual, includes encryptor and (closely-related) decryptor
  - How it works is beyond scope of this class…
    But if you are curious: http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html
- Not proven secure
  - But no known flaws
    - The NSA uses it for Top Secret communication with 256b keys:
      stuff they want to be secure ***for 40 years including possibly unknown breakthroughs***!
  - so we assume it is a secure block cipher

AES is also effectively free…

- Computational load is remarkably low
  - Partially why it won the competition:
    There were 3 really good finalists from a performance viewpoint:
    Rijndael (the winner), Twofish, Serpent
    One OK: RC6
    One ugggly: Mars
- On any given computing platform:
  Rinjdael was ***never*** the fastest
- But on every computing platform:
  Rinjdael was ***always*** the second fastest
- And now CPUs include dedicated AES support

How Hard Is It To Brute-Force 128-bit Key?

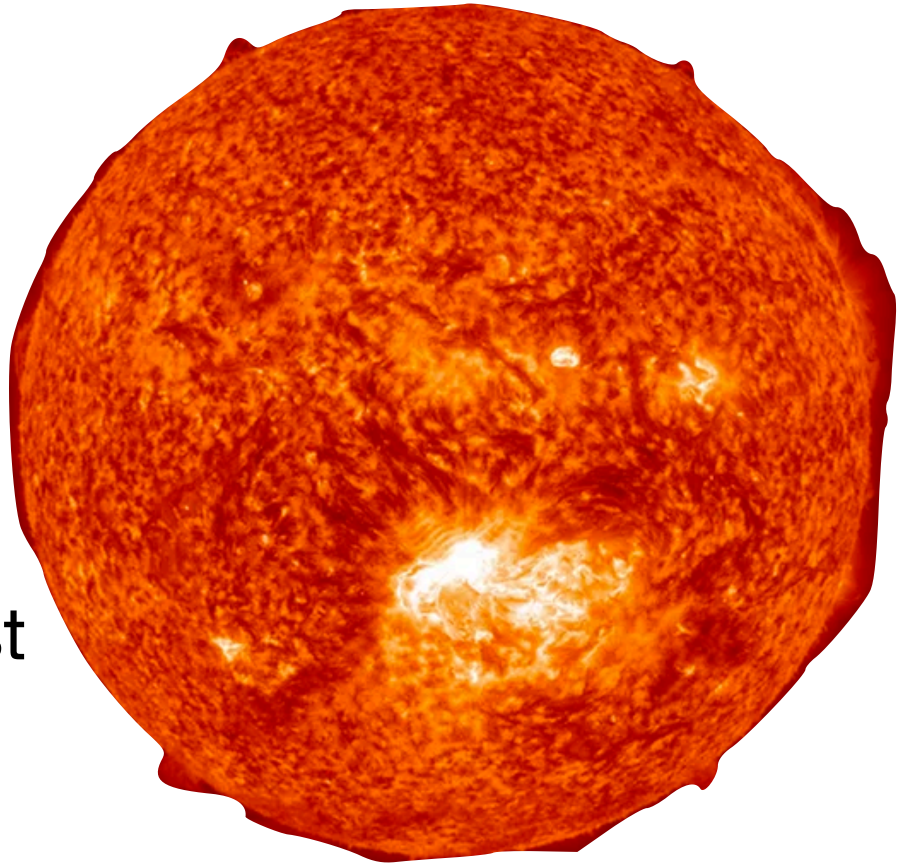
- 2^{128} possibilities – well, how many is that?
- Handy approximation: $2^{10} \approx 10^3$
- $2^{128} = 2^{10*12.8} \approx (10^3)^{12.8} \lesssim (10^3)^{13} \approx 10^{39}$

How Hard Is It To Brute-Force 128-bit Key?

- 2^{128} possibilities – well, how many is that?
- Handy approximation: $2^{10} \approx 10^3$
- $2^{128} = 2^{10*12.8} \approx (10^3)^{12.8} \lesssim (10^3)^{13} \approx 10^{39}$
- Say we build massive hardware that can try 10^9 (1 billion) keys in 1 nanosecond (a billionth of a second)
  - So 10^{18} keys/sec
  - Thus, we'll need $\approx 10^{21}$ sec
- How long is that?
  - One year $\approx 3 \times 10^7$ sec
  - So need $\approx 3 \times 10^{13}$ years $\approx$ 30 trillion years

What about a 256b key in a year?

- Time to start thinking in ***astronomical*** numbers:
  - If each brute force device is $1mm^3$...
  - We will need 10^{52} of these things...
- 10^{43} cubic meters...
- Or the volume of ***$7x10^{15}$ suns***!
- Brute force is ***not a factor*** against modern block ciphers...
  IF the key is actually random!



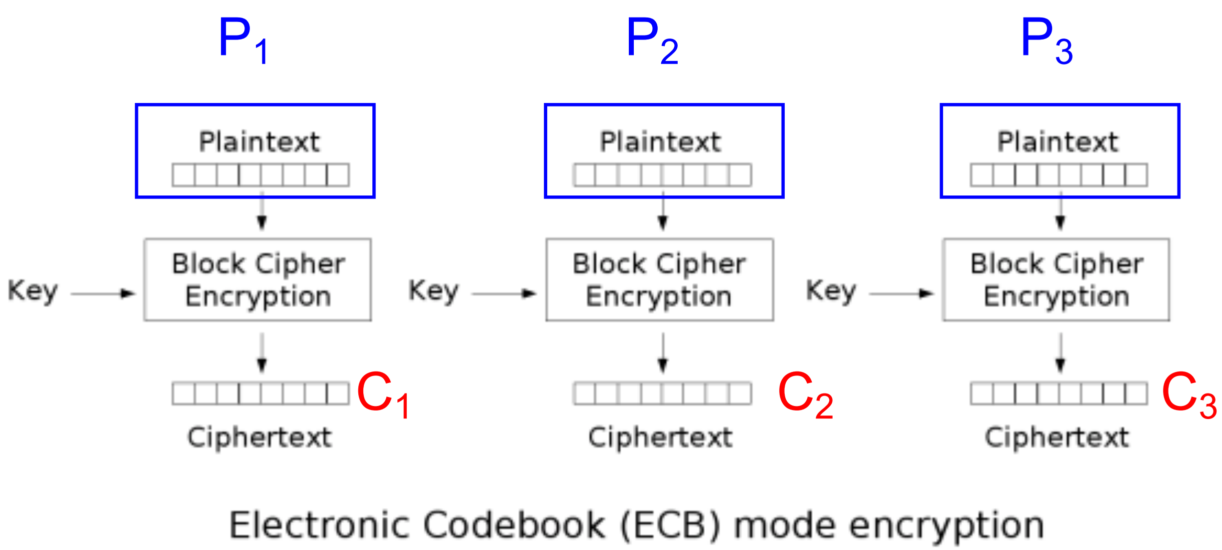
Issues When Using the Building Block

- Block ciphers can only encrypt messages of a certain size
  - If **M** is smaller, easy, just pad it (more later)
  - If **M** is larger, can repeatedly apply block cipher
    - Particular method = a “block cipher mode”
    - Tricky to get this right!
- If same data is encrypted twice, attacker knows it is the same
  - Solution: incorporate a varying, known quantity (IV = “initialization vector”)

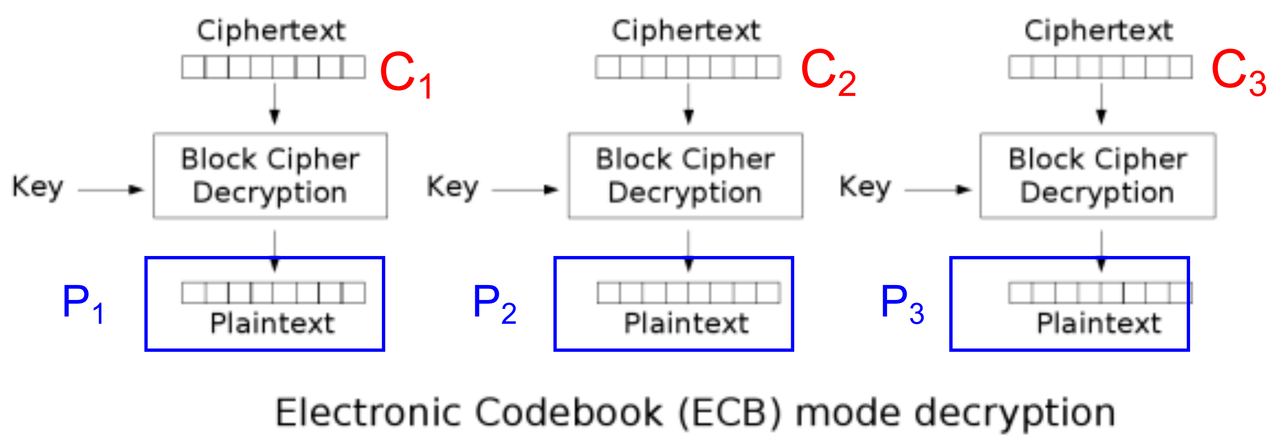
Electronic Code Book (ECB) mode

- Simplest block cipher mode
- Split message into b-bit blocks $\mathbf{P_1}$, $\mathbf{P_2}$, …
- Each block is enciphered independently, separate from the other blocks
  $\mathbf{C_i = E(P_i, K)}$
- Since key **K** is fixed, each block is subject to the same permutation
  - (As though we had a “code book” to map each possible input value to its designated output)

ECB Encryption



ECB Decryption

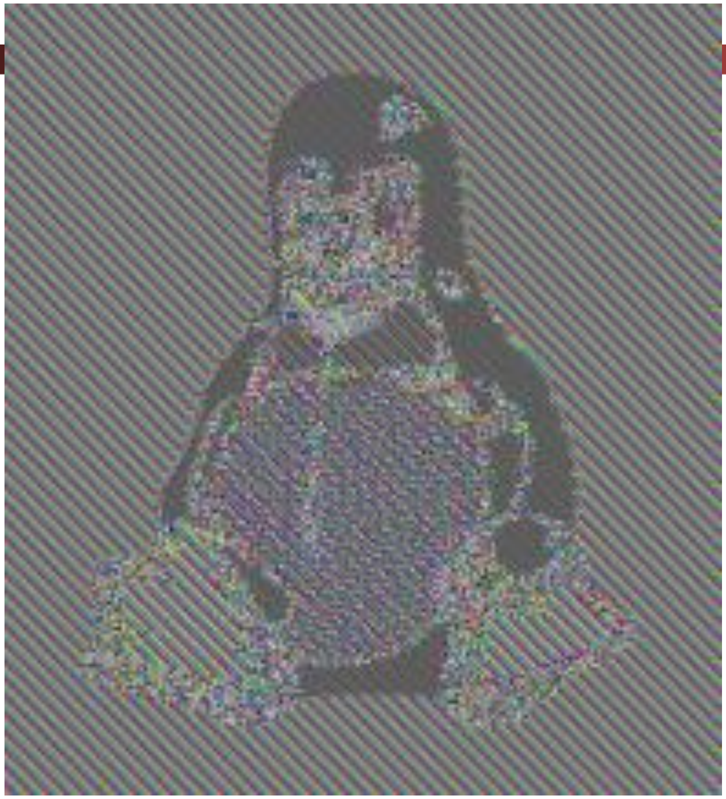


Problem: Relationships between P_i's reflected in C_i's

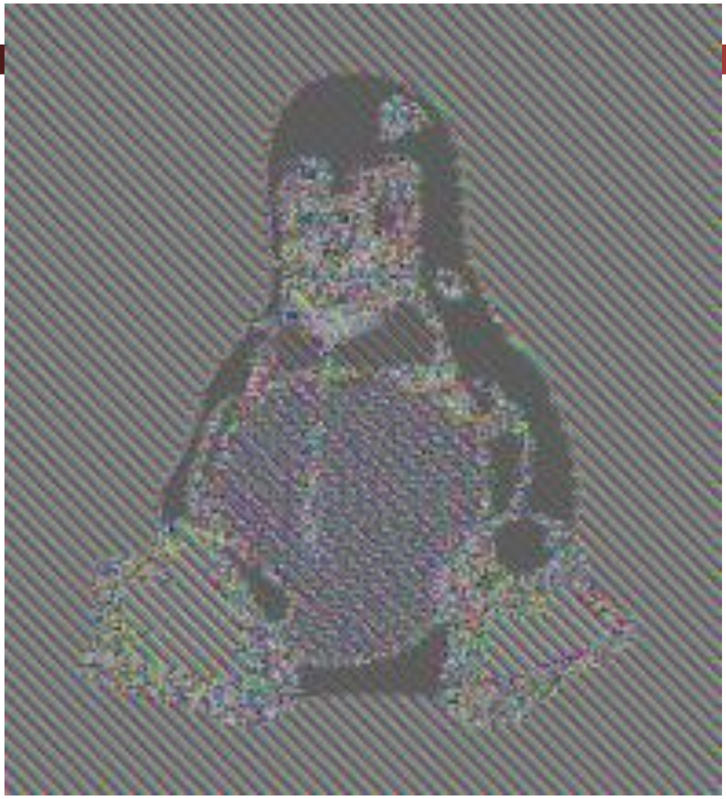
IND-CPA and ECB?

- Of course not!
- **M,M'** is 2x the block length...
  - **M** = all 0s
  - **M'** = 0s for 1 block, 1s for the 2nd block
- This has catastrophic implications in the real world...

Original image, RGB values split into a bunch of b-bit blocks



Encrypted with ECB and interpreting ciphertext directly as RGB



Later (identical) message again encrypted with ECB

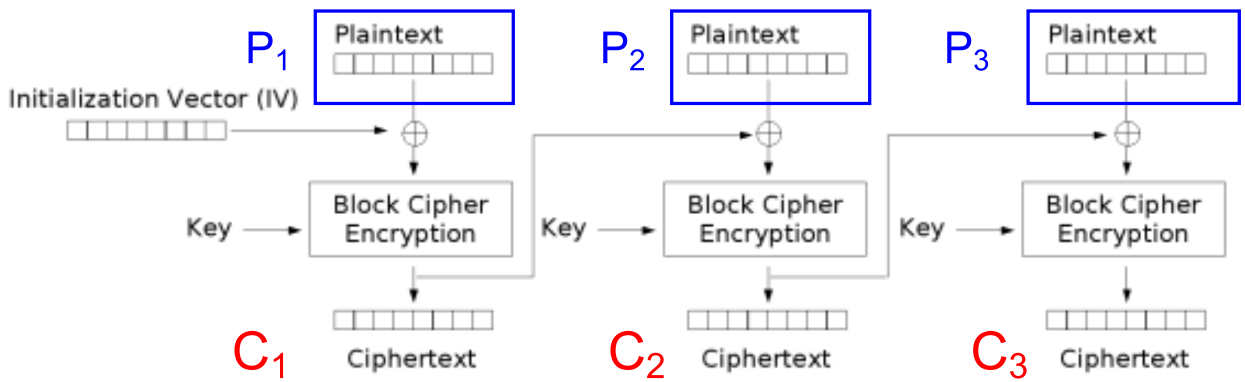
Building a Better Cipher Block Mode

- Ensure blocks incorporate more than just the plaintext to mask relationships between blocks. Done carefully, either of these works:
  - Idea #1: include elements of prior computation
  - Idea #2: include positional information
- Plus: need some initial randomness
  - Prevent encryption scheme from determinism revealing relationships between messages
  - Introduce initialization vector (IV):
    - IV is a public ***nonce***, a use-once unique value: Easiest way to get one is generate it randomly
- Example: Cipher Block Chaining (CBC)

CBC Encryption

E(Plaintext, K):

- If b is the block size of the block cipher, split the plaintext in blocks of size b: P_1, P_2, P_3,..
- Choose a random IV (do not reuse for other **messages**)
- Now compute:



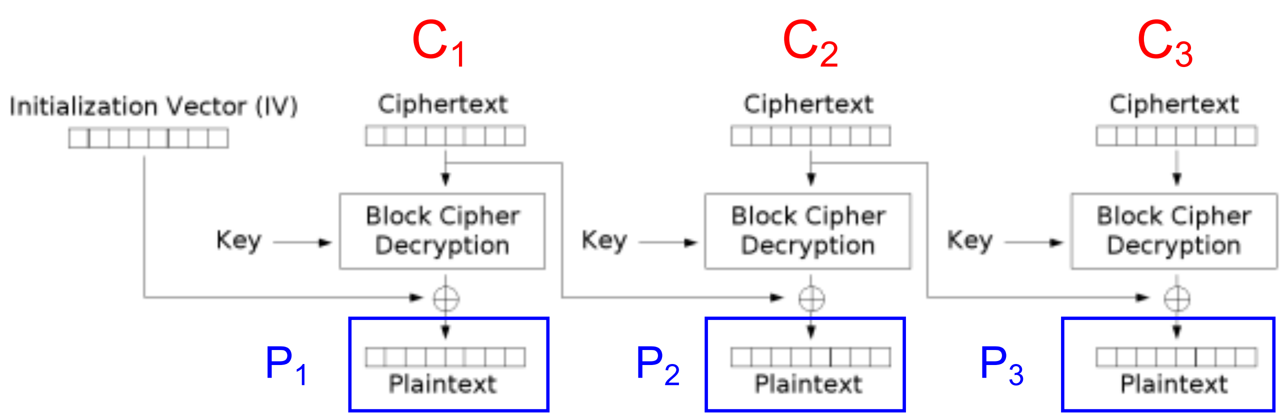
Cipher Block Chaining (CBC) mode encryption

- Final ciphertext is (IV, C_1, C_2, C_3). This is what Eve sees.

CBC Decryption

D(Ciphertext, K):

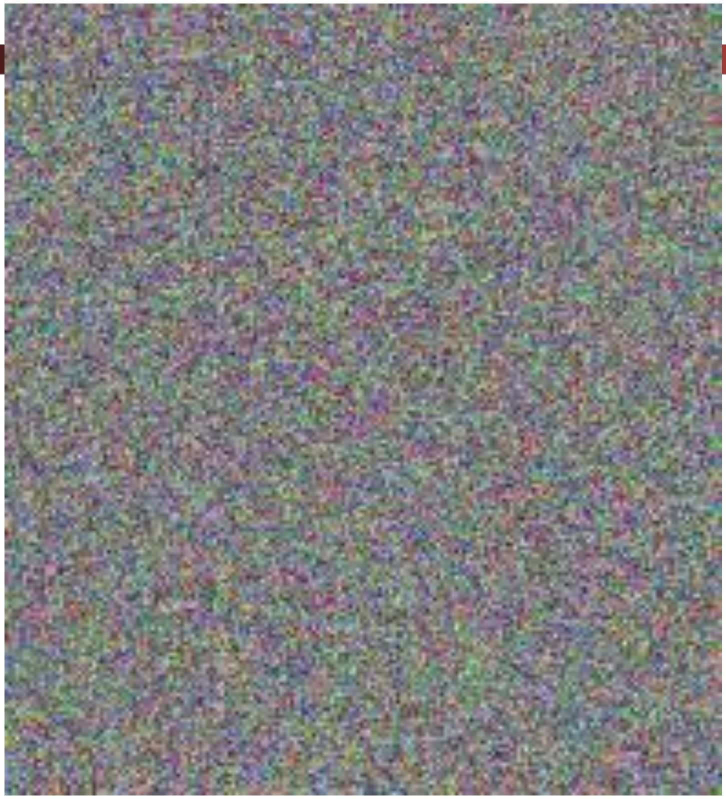
- Take IV out of the ciphertext
- If b is the block size of the block cipher, split the ciphertext in blocks of size b: C_1, C_2, C_3, …
- Now compute this:



Cipher Block Chaining (CBC) mode decryption

- Output the plaintext as the concatenation of P_1, P_2, P_3, ...

Original image, RGB values split into a bunch of b-bit blocks



Encrypted with CBC: Should be indistinguishable from random noise

CBC Mode...

- Widely used
- Issue: sequential encryption, can't parallelize encryption
  - ***Must*** finish encrypting block b before starting b+1
  - But you can parallelize decryption
- Parallelizable alternative: CTR (Counter) mode
- Security: If no reuse of nonce, both are provably secure (IND-CPA) assuming the underlying block cipher is secure

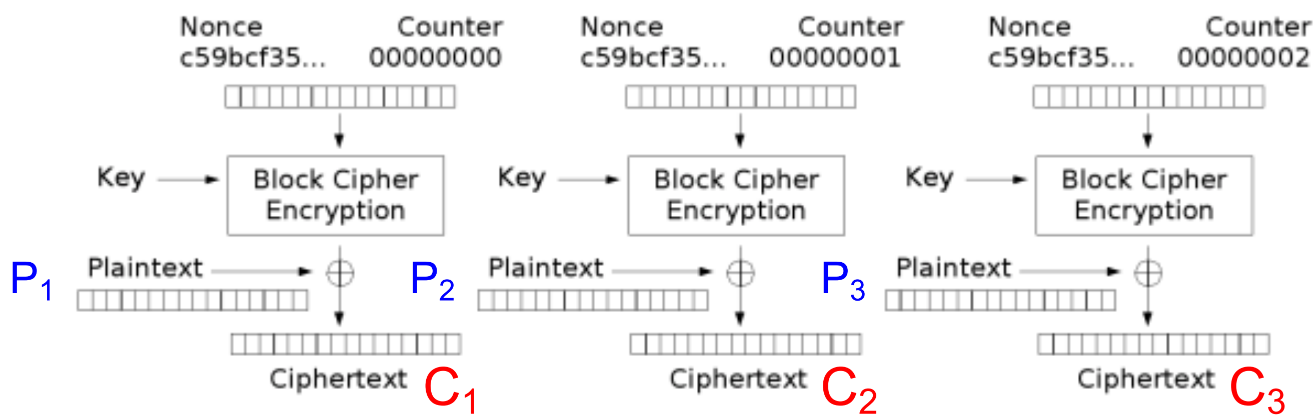
And padding…

- What happens if length(**M**) % BlockSize != 0?
  - Need to "Pad" to add bits
- Two main options:
  - Send the length at the start of the message…
    - And then who cares what you add on at the end
  - Use a padding scheme that you can add on to the end…
- EG, PKCS#7:
  - If M % BlockSize == Blocksize - 1: Pad with 0x01
  - If M % BlockSize == Blocksize - 2: Pad with 0x02 0x02

    ….
  - If M % BlockSize == 0: Pad a ***full block*** with the block size (so for AES 0x20 0x20…)

CTR Mode Encryption

(Nonce = Same as IV)

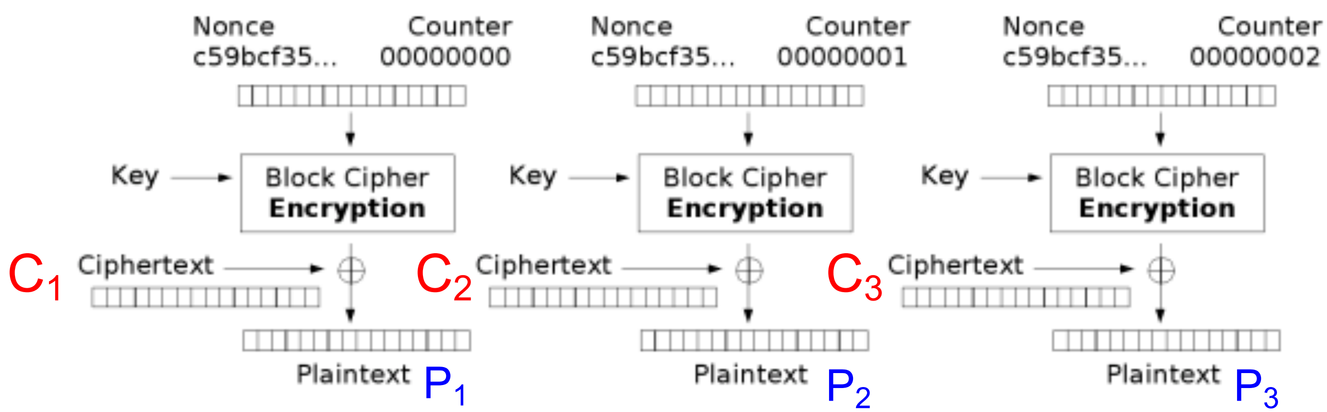


Counter (CTR) mode encryption

Important that nonce/IV does not repeat across different encryptions.

Choose at random!

Counter Mode Decryption



Counter (CTR) mode decryption

Note, CTR decryption uses block cipher's *encryption*, **not** decryption

Thoughts on CTR mode...

- CTR mode is actually a stream cipher (more on those later):
  - You no longer need to worry about padding which is nice
- CTR mode is fully parallelizeable for encryption as well as decryption

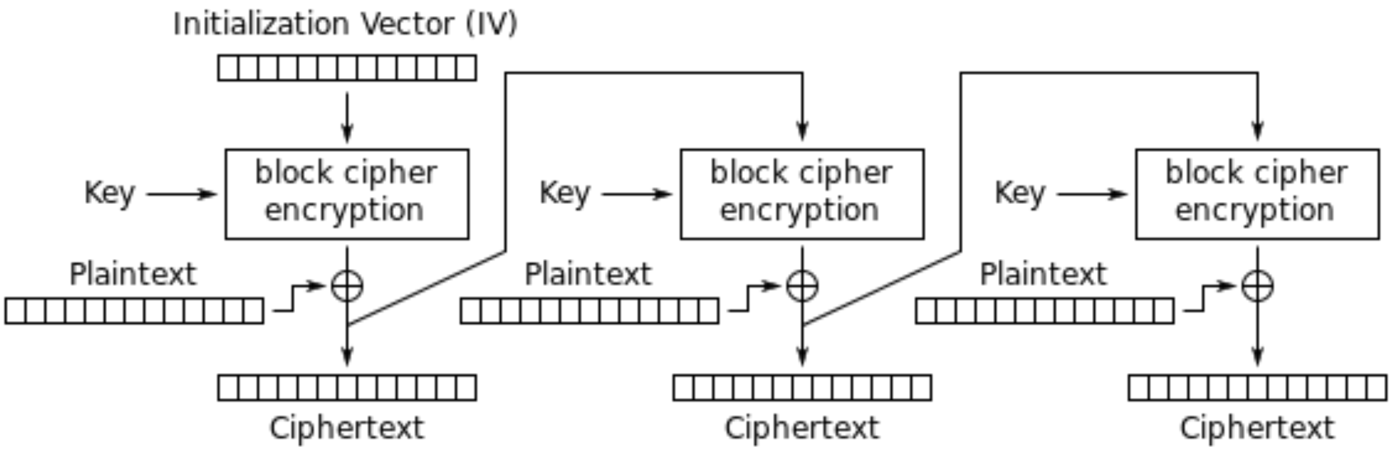
NEVER EVER EVER use CTR Mode! (Well, if you are paranoid…)

- What happens if you reuse the IV in CBC...
  - Its bad but not catastrophic:
    you fail IND-CPA but the damage may be tolerable:
    - **M = {A,A,B}**
      M' = {A,B,B}
      Adversary can see that the first part of M and M' are the same, but not the later part
- What happens if you reuse the IV in CTR mode?
  - It is ***exactly*** like reusing a one-time pad!
- An example of a system which fails badly...
  - CTR mode is ***theoretically*** as secure as CBC when used properly...
  - But when it is misused it fails catastrophically:
    Personal bias: I believe we need systems that are still robust ***when implemented incorrectly***

What To Use Then?

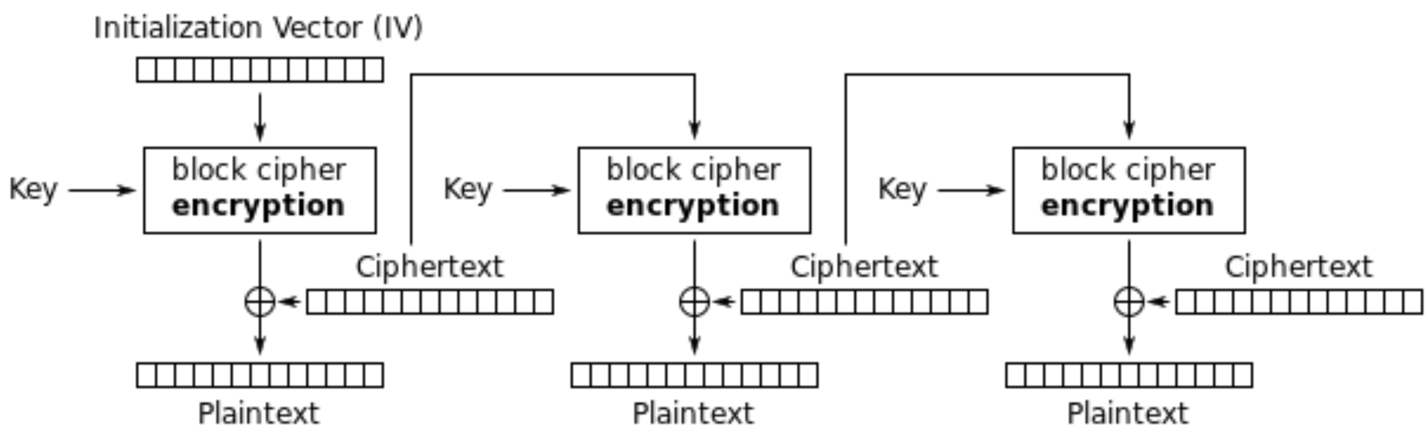
- What if you want a cipher mode where you don't need to pad (like CTR mode)?
  - But you want the robust to screwup properties of CBC mode?
- Idea: lets do it CTR-like (xor plaintext with block cipher output), but...
- Instead of the next block input being an incremented counter...
  have the next block be the previous ciphertext
- Still lacks integrity however, we'll fix that next time...

CFB Encryption



Cipher Feedback (CFB) mode encryption

CFB Decryption



Cipher Feedback (CFB) mode decryption

CFB doesn't need to pad...

- Since the encryption is XORed with the plaintext...
  - You can end on a "short" block without a problem
  - So more convenient than CBC mode
- But similar security properties as CBC mode
  - Sequential encryption, parallel decryption
  - Same error propagation effects
  - Effectively the same for IND-CPA
- But a bit worse if you reuse the IV