
Cryptography is nightmare magic
math that cares what kind of pen
you use -@swiftonsecurity

Finish Up Software Security & Crypto 1

Announcements & Reminders

- Reminders on the chat....
  - Use Q&A to ask questions
  - Use chat to talk amongst yourselves:
    be sure to send to ALL not just "send to panelists"
  - Just say SOMETHING, SOMETIME in chat for attendance tracking
- During the short break...
  - Those who still want partners, lets use the chat to find them
- Homework 1 due tomorrow at 11:59 PM PDT
  - No late homework accepted, but you do get one homework drop

Why does software have vulnerabilities?

- Programmers are humans.
  And humans make mistakes.
  - Use tools
- Programmers often aren't security-aware.
  - Learn about common types of security flaws.
- Programming languages aren't designed well for security.
  - Use better languages (Java, Python, …).

Testing for Software Security Issues

- What makes testing a program for security problems difficult?
  - We need to test for the absence of something
    - Security is a ***negative*** property!
      - "nothing bad happens, even in really unusual circumstances"
  - Normal inputs rarely stress security-vulnerable code
- How can we test more thoroughly?
  - Random inputs (fuzz testing)
  - Mutation
  - Spec-driven
  - Use tools like Valgrind
  - Test ***corner cases***
- How do we tell when we've found a problem?
  - Crash or other deviant behavior
- How do we tell that we've tested enough?
  - Hard: but code-coverage tools can help

Laura (X-23): Disney's BEST Princess!

Working Towards Secure Systems

- Along with securing individual components, we need to keep them up to date …
- What's hard about patching?
  - Can require restarting production systems
  - Can break crucial functionality

Threat Level: GREEN YELLOW ORANGE RED

Storm Center Tools

ISC Diary

Refresh Latest Diaries

previous next

Oracle quitely releases Java 7u13 early

Published: 2013-02-01,
Last Updated: 2013-02-01 21:59:59 UTC
by Jim Clausing (Version: 2)

Recommend Tweet +1

2 comment(s)

First off, a huge thank you to readers Ken and Paul for pointing out that Oracle has released Java 7u13. As the CPU (Critical Patch Update) bulletin points out, the release was originally scheduled for 19 Feb, but was moved up due to the active exploitation of one of the critical vulnerabilities in the wild. Their Risk Matrix lists 50 CVEs, 49 of which can be remotely exploitable without authentication. As Rob discussed in his diary 2 weeks ago, now is a great opportunity to determine if you really need Java installed (if not, remove it) and, if you do, take additional steps to protect the systems that do still require it. I haven't seen jusched pull this one down on my personal laptop yet, but if you have Java installed you might want to do this one manually right away. On a side note, we've had reports of folks who installed Java 7u11 and had it silently (and unexpectedly) remove Java 6 from the system thus breaking some legacy applications, so that is something else you might want to be on the lookout for if you do apply this update.

Working Towards Secure Systems

- Along with securing individual components, we need to keep them up to date …
- What's hard about patching?
  - Can require restarting production systems
  - Can break crucial functionality
  - Management burden:
    - It never stops (the "patch treadmill") …

News

infosecurity
STRATEGY /// INSIGHT /// TECHNIQUE

IT administrators give thanks for light Patch Tuesday

07 November 2011

Microsoft is giving IT administrators a break for Thanksgiving, with only four security bulletins for this month's Patch Tuesday.

Only one of the bulletins is rated critical by Microsoft, which addresses a flaw that could result in remote code execution attacks for the newer operating systems – Windows Vista, Windows 7, and Windows 2008 Server R2.

The critical bulletin has an exploitability rating of 3, suggesting

Working Towards Secure Systems

- Along with securing individual components, we need to keep them up to date …
- What's hard about patching?
  - Can require restarting production systems
  - Can break crucial functionality
  - Management burden:
    - It never stops (the "patch treadmill") …
    - … and can be difficult to track just what's needed where
- Other (complementary) approaches?
  - Vulnerability scanning: probe your systems/networks for known flaws
  - Penetration testing ("pen-testing"): pay someone to break into your systems …
    - … provided they take excellent notes about how they did it!

RISK ASSESSMENT / SECURITY & HACKTIVISM

ars technica

Extremely critical Ruby on Rails bug threatens more than 200,000 sites

Servers that run the framework are by default vulnerable to remote code attacks.

by **Dan Goodin** - Jan 8 2013, 4:35pm PST

HARDENING 38

Hundreds of thousands of websites are potentially at risk following the discovery of an extremely critical vulnerability in the Ruby on Rails framework that gives remote attackers the ability to execute malicious code on the underlying servers.

The bug is present in Rails versions spanning the past six years and in default configurations gives hackers a simple and reliable way to pilfer database contents, run system commands, and cause websites to crash, according to Ben Murphy, one of the developers who has confirmed the vulnerability. As of last week, the framework was used by more than 240,000 websites, including Github, Hulu, and Basecamp, underscoring the seriousness of the threat.

"It is quite bad," Murphy told Ars. "An attack can send a request to any Ruby on Rails sever and then execute arbitrary commands. Even though it's complex, it's reliable, so it will work 100 percent of the time."

Murphy said the bug leaves open the possibility of attacks that cause one site running rails to seek out and infect others, creating a worm that infects large swaths of the Internet. Developers with the Metasploit framework for hackers and penetration testers are in the process of creating a module that can scan the Internet for vulnerable sites and exploit the bug, said HD Moore, the CSO of Rapid7 and chief architect of Metasploit.

Maintainers of the Rails framework are urging users to update their systems as soon as possible to

Some Approaches for Building Secure Software/Systems: Use Mitigations!

- Run-time checks
  - Automatic bounds-checking (overhead)
  - What do you do if check fails? Probably controlled crash...
- Address randomization
  - Make it hard for attacker to determine layout
  - But they might get lucky / sneaky
- Non-executable stack, heap
  - May break legacy code
  - See also Return-Oriented Programming (ROP)
- Monitor code for run-time misbehavior
  - E.g., illegal calling sequences
  - But again: what do you if detected?

Approaches for Secure Software, con't

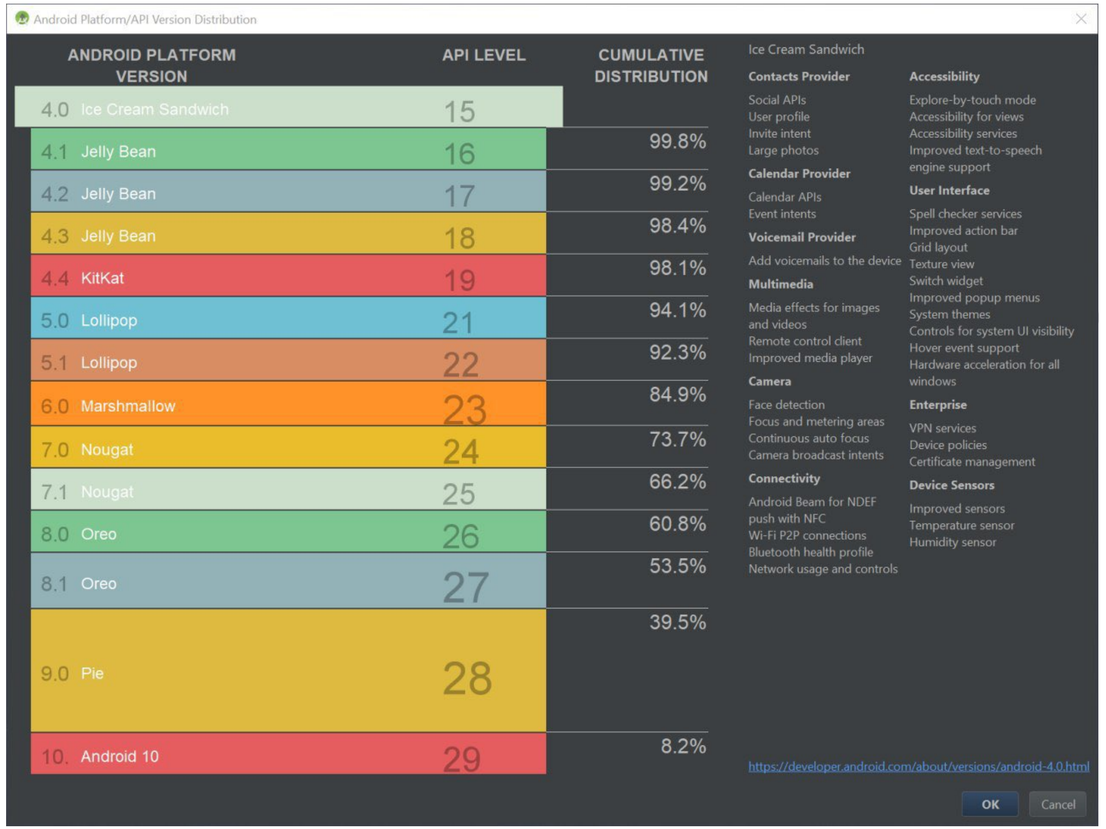
- Program in checks / "defensive programming"
  - E.g., check for null pointer even though sure pointer will be valid
  - Relies on programmer discipline
- Use safe libraries
  - E.g. **`strlcpy`**, not **`strcpy`**; **`snprintf`**, not **`sprintf`**
  - Relies on discipline or tools …
- Bug-finding tools
  - Excellent resource as long as not many false positives
- Code review
  - Can be very effective … but expensive

Approaches for Secure Software, con't

- Use a safe language
  - E.g., Java, Python, C#, Go, Rust
  - Safe = memory safety, strong typing, hardened libraries
  - Installed base? Programmer base? Legacy? ~~Performance~~?
- Structure user input
  - Constrain how untrusted sources can interact with the system
    - Really key later when we get to SQL injection...
  - Perhaps by implementing a reference monitor
- Contain potential damage
  - E.g., run system components in jails or VMs
  - Think about privilege separation

Real World Security: Securing your cellphone... Look on the back:

- Does it say "iPhone"?
  - Keep it up to date and be happy
- Does it say "Pixel"?
  - Keep it up to date and be happy
- Does it say anything else?
  - Toss it in the trash and buy an iPhone SE or a Pixel 4a
- Why? The Android Patch Model...
  - "Imagine if your Windows update needed to be approved by Intel, Dell, and Comcast...
    And none of them cared or had a reason to care"
  - (Note: Google stopped updating the public dashboard that provided the great pie-chart, which is why this is a 2019 version in the pie chart)



A Real World Exploitation Example: Exploit on Safari

- This was discovered about 2 year ago: https://blog.ret2.io/2018/07/11/pwn2own-2018-jsc-exploit/
  - But the theme of how this work is quite common:
    Any use-after-free in Javascript where you can do this to an arbitrary object will usually target arrays
- Basic idea: a race condition can enable use-after-free in the JavaScript interpreter
  - And the attacker's code is in JavaScript running on the target browser
- Use to create limited read/write primitive...
- To create arbitrary read/write primitive...
- To create binary code execution!

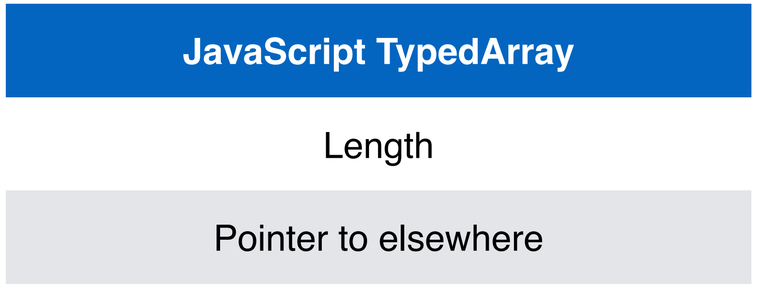
Key Insight #1: JavaScript Arrays...

- JavaScript arrays are length-checked
  - So you can't read past the end of the array
- But if we allocate a bunch of arrays...
  - And they are all still being used...
  - But the runtime has a use-after-free error that allows us to reuse the memory as something else...
  - We can overwrite the length field in the array specifier!
  - Try this a whole bunch of times until successful (since it is probably a race condition to create the use-after-free condition)
- This gets a relative read/write primitive
  - We have a JavaScript array that can read/write a long hunk of memory including other JavaScript objects

JavaScript Array	JavaScript Integer
Length	Value
Data	
Data	

Key Insight #2: JavaScript TypedArray objects

- A way for JavaScript to have high performance access to other parts of memory
  - Used for video, audio, etc...
    Runtime can say "Here, JavaScript, this is a fixed blob of memory for you to play with"
  - Consists of a pointer to non-JavaScript memory and a length field



- So once we can read/write to a hunk of memory containing other JavaScript objects...
  - Lets take one of those objects and rewrite it so the runtime thinks it is a TypedArray object:
    Now we can both adjust the pointer and the length
- Now we can arbitrarily write & read memory using JavaScript!

Key Insight #3:
Why is JavaScript not glacially slow?

- Because JavaScript code is dynamically compiled into machine code!
  - So there exist pages in memory for this code...
  - That are set as both writeable and executable!
- So just find one of them...
  - Create & run JavaScript functions, follow the pointers, and there you go!
- Now we can write assembly... Start it running...
  And we've won! Full arbitrary code execution!

Of course...
Now you have to exit the sandbox!

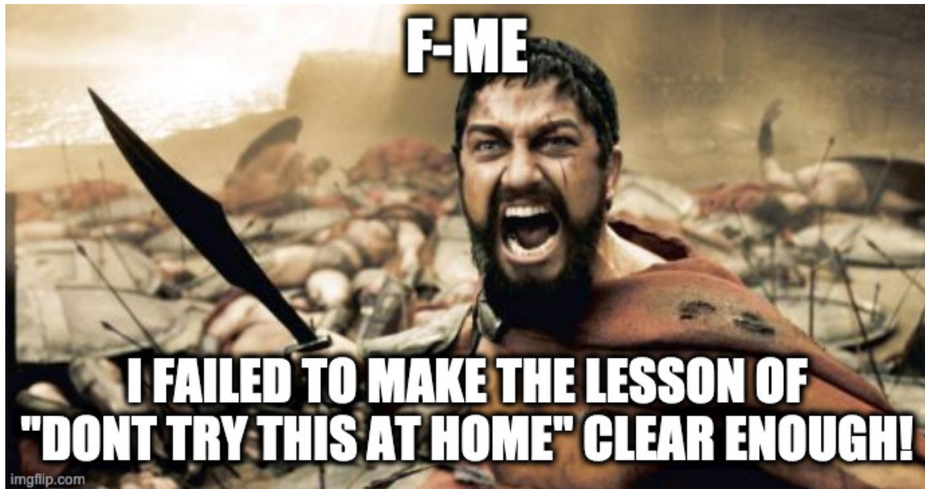
- https://blog.ret2.io/2018/07/25/pwn2own-2018-safari-sandbox/
- The Safari renderer is running in its own limited-authority process
  - With a white-list of external resources its now allowed to access
- So find a vulnerability in one of those external resources
  - Which in the Mac case is the "window server" that handles the drawing and has a wide attack surface

Cryptography: Philosophy...

- This part of the class is really ***don't try this at home***
  - It is ***incredibly easy*** to screw this stuff up
- It isn't just a matter of making encryption algorithms...
  - Unless your name is David Wagner or Ralcua Popa, ***your crypto is broken***!
- It isn't just a matter of coding good algorithms...
  - Although just writing 100% correct code normally is hard enough!
- There is all sorts of deep voodoo that, ***when*** you screw up your security breaks
  - EG, bad random number generators, side channel attacks, reusing one-use-only items, replay attacks, downgrade attacks, you name it...

LET ME REITERATE!!!
DON'T DO THIS AT HOME!!!!

- This summer, 61A did a custom exam tool
  - It would encrypt several python files for each student
    - Every student got a different exam
  - Written by a student who took CS161 already!!! With Me! WITH THESE WARNINGS!!!!
    - Yet I failed...
- Each exam question was supposed to take 20 minutes
  - So they would release the key for "question 1" to everyone...
    Then question 2...
  - Everyone had the same key but a different question
- They used a "secure" algorithm...
  In an insecure way!
  - Breaking this will be a late-semester lab2
- And we don't ***know*** if it got broken
  - One detected student claimed there was a leakage of the exam before the start



Three main goals

- ***Confidentiality***: preventing adversaries from ***reading*** our private data
  - Data = message or document
- ***Integrity***: preventing attackers from ***altering*** our data
  - Data itself might or might not be private
- ***Authentication***: proving who ***created*** a given message or document
  - Generally implies/requires integrity

Special guests

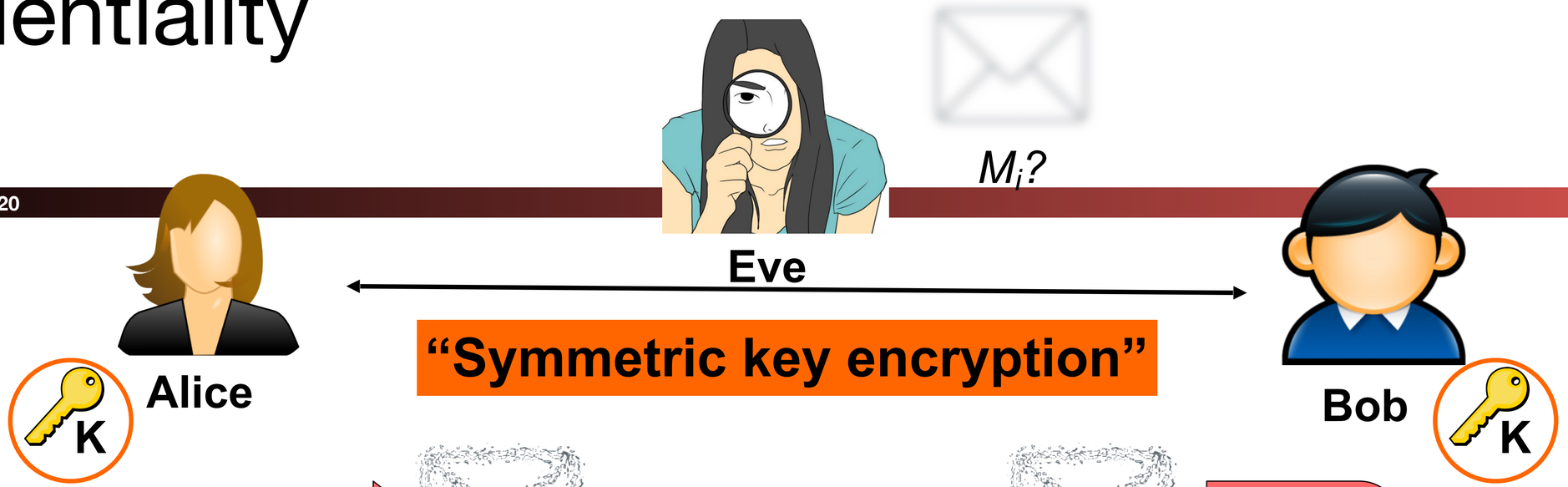
- Alice (sender of messages)

- Bob (receiver of messages)

- The attackers
  - Eve: “eavesdropper”
  - Mallory: “manipulator”

Eve

Confidentiality



$E(M_i, K)$ and $D(C_i, K)$ are *inverses* for the same K

The Ideal Contest

- Attacker's goal: any knowledge of M_i beyond an upper bound on its length
  - Slightly better than 50% probability at guessing a single bit: attacker wins!
  - Any notion of how M_i relates to M_j: attacker wins!
- Defender's goal: ensure attacker has no reason to think any $M' \in \{0,1\}^n$ is more likely than any other
  - (for M_i of length n)

Eve's Capabilities/Foreknowledge

- No knowledge of **K**
  - We assume **K** is selected by a ***truly random process***
  - For **b**-bit key, any **K** $\in$ $\mathbf{\{0,1\}^b}$ is equally likely
- Recognition of success: Eve can generally tell if she has correctly and fully recovered $\mathbf{M_i}$
  - But: Eve cannot recognize anything about ***partial solutions***, such as whether she has correctly identified a particular bit in $\mathbf{M_i}$
    - There are some attacks where Eve can guess and verify:
      Often a side-channel using a "decryption oracle": fooling the server into trying & failing
      with the nature of different failures telling Eve whether she guessed right
  - Does not apply to scenarios where Eve exhaustively examines every possible $\mathbf{M_i'} \in \mathbf{\{0,1\}^n}$

Eve's Available Information

1. Ciphertext-only attack:
   - Eve gets to see every instance of $\mathbf{C_i}$
   - Variant: Eve may also have partial information about $\mathbf{M_i}$
     - "It's probably English text"
     - Bob is Alice's stockbroker, so it's either "Buy!" or "Sell"
2. Known plaintext:
   - Eve knows part of $\mathbf{M_i}$ and/or entire other $\mathbf{M_j}$s
   - How could this happen?
     - Encrypted HTTP request: starts with "GET"
     - Eve sees earlier message she knows Alice will send to Bob
     - Alice transmits in the clear and then resends encrypted
     - Alex the Nazi always transmits the weather report at the same time of day, with the word "Wetter" in a known position

Eve's Available Information, con't

3. Chosen plaintext

- Eve gets Alice to send $\mathbf{M_j}$'s of Eve's choosing
- How can this happen?
  - E.g. Eve sends Alice an email spoofed from Alice's boss saying "Please securely forward this to Bob"
  - E.g. Eve has some JavaScript running in Alice's web browser that is contacting Bob's TLS web server

4. Chosen ciphertext:

- Eve tricks Bob into decrypting some $\mathbf{C_j}$' of her choice and he reveals something about the result
- How could this happen?
  - E.g. repeatedly send ciphertext to a web server that will send back different-sized messages depending on whether ciphertext decrypts into something well-formatted
  - Or: measure how long it takes Bob to decrypt & validate

Eve's Available Information, con't

5. Combinations of the above

- Ideally, we'd like to defend against this last, the most powerful attacker
- And: we can!, so we'll mainly focus on this attacker when discussing different considerations

Independence Under Chosen Plaintext Attack game: IND-CPA

- Eve is interacting with an encryption "Oracle"
  - Oracle has an unknown random key **k**
- Eve 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 record

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
  $\mathbf{C_i}$ = **E(**$\mathbf{M_i}$**,K)** = **ROT3**("DOG") = "GRJ"
  D($\mathbf{C_i}$**,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?

Break Time!

- Hey, lets use chat to find partners!

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 go further with "chaining", e.g., 2nd rotation 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 ⊕ 0 = X

X ⊕ X = 0

X ⊕ Y = Y ⊕ X

X ⊕ (Y ⊕ Z) = (X ⊕ Y) ⊕ 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
- **E(M,K) = M ⊕ K** (⊕ = XOR)
  D(C,K) = C ⊕ K
  = M ⊕ K ⊕ K = M ⊕ 0 = M

X ⊕ 0 = X
X ⊕ X = 0
X ⊕ Y = Y ⊕ X
X ⊕ (Y ⊕ Z) = (X ⊕ Y) ⊕ 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

Number Stations:
One Time Pads in the Real World

- There are shortwave and terrestrial radio "number stations"
  - At a regular time, a voice gets on the air, reads a series of seemingly random numbers
- For those without a corresponding one-time pad...
  - They are simply a sequence of random numbers
- But if you do have the one-time pad...
  - You can decrypt the super-secret spy message being sent to you
- But what if you don't want to send anything to any spies?
  - Just read out a list of random numbers ***anyway***

"Traffic Analysis" & "Sidechannels"

- Traffic analysis: Simply knowing who is talking to whom or when
  - Can often reveal umm, interesting secrets
- Sidechannels: Something outside the cryptography itself that reveals something interesting
  - How modern crypto systems are usually broken

A Sidechannel/Traffic Analysis Spy Example

- In the 90s, there were some Russian spies in the US
  - "The Americans" was based on this incident
- A cuban-broadcast numbers station had a bug...
  - Some nights it would never say "9"
- It turned out this corresponded when the Russian spies were on vacation!
  - And the FBI used that as part of their investigation!
- (Revealed inadvertently by Struck and analyzed by Matt Blaze)

Modern Encryption: Block cipher

- A function $\mathbf{E : \{0, 1\}^b \times \{0, 1\}^k \rightarrow \{0, 1\}^b}$. Once we fix the key **K** (of size **k** bits), we get:
- $\mathbf{EK : \{0,1\}^b \rightarrow \{0,1\}^b}$ denoted by $\mathbf{E_K(M) = E(M,K)}$.
  - (and also **D(C,K)**, **E(M,K)**'s inverse)
- Three properties:
  - Correctness:
    - $\mathbf{E_K(M)}$ 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
  - The other two good ones always had a "this is the compute platform they are bad at"
- And now CPUs include dedicated AES support

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

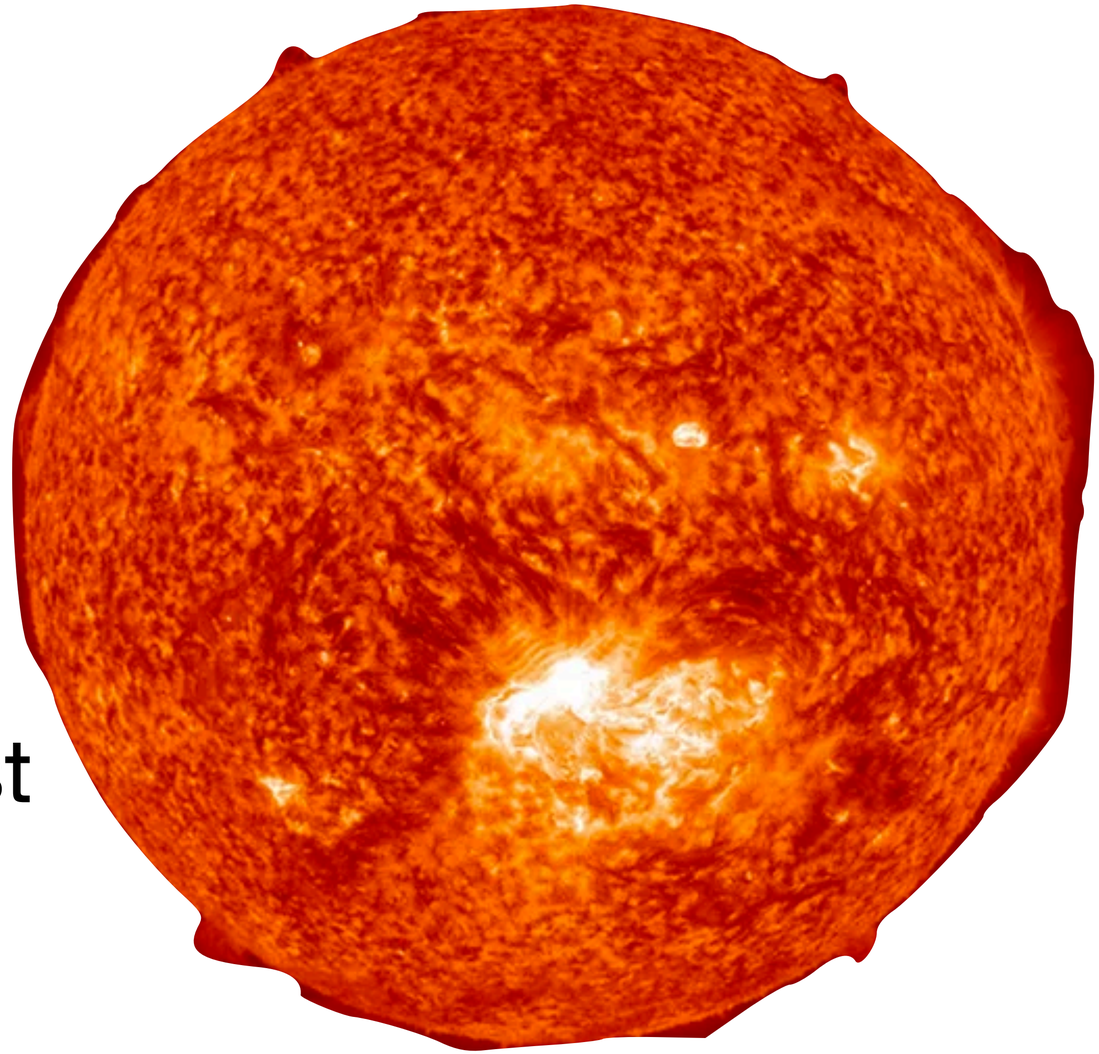
- 2^{128} possibilities – well, how many is that?
- Handy approximation: $2^{10} \approx 10^3$
- $2^{128} = 2^{10*12.8} \approx (10^3)^{12.8} \lesssim (10^3)^{13} \approx 10^{39}$

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 ***7×10^{15} suns***!
- Brute force is ***not a factor*** against modern block ciphers...
  IF the key is actually random!



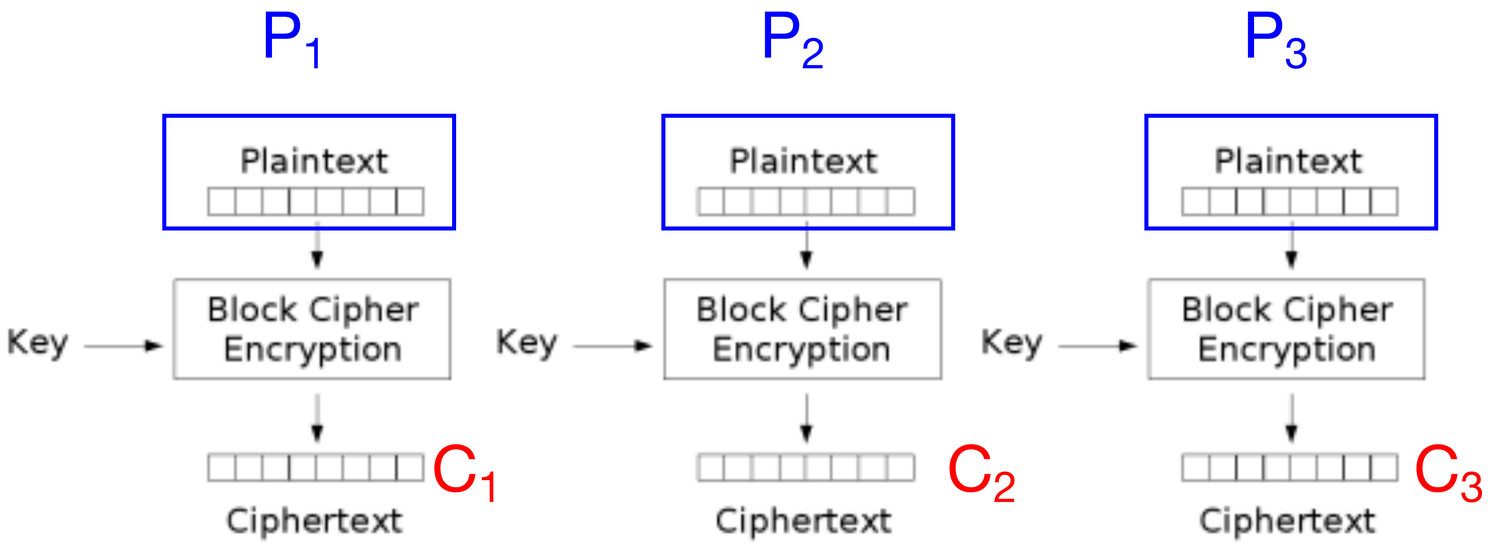
Issues When Using the Building Block

- 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

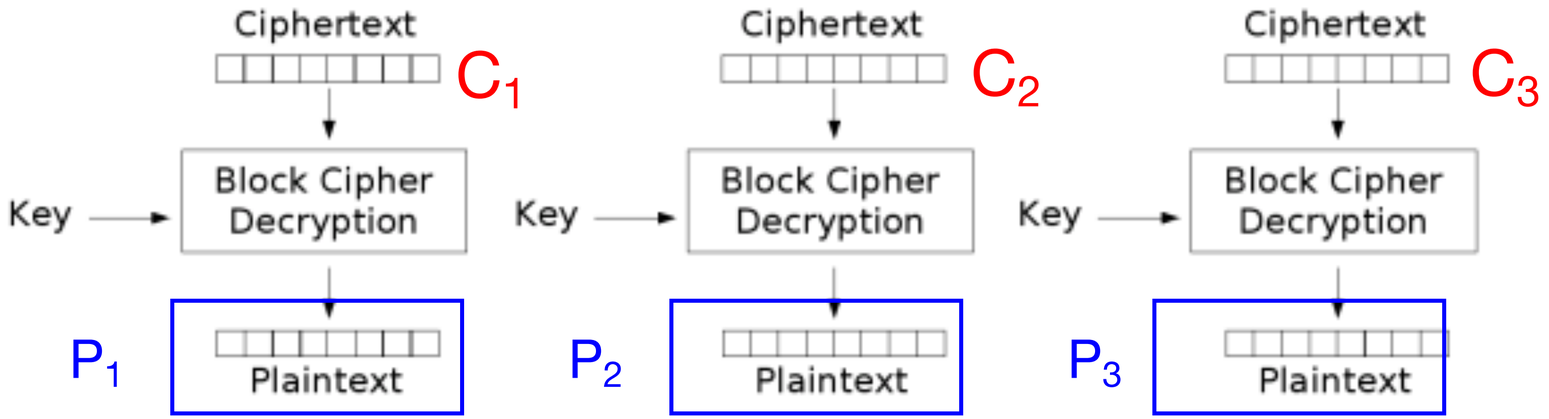
- Simplest block cipher mode
- Split message into b-bit blocks $\mathbf{P_1}$, $\mathbf{P_2}$, …
- Each block is enciphered independently, separate from the other blocks
  $\mathbf{C_i = E(P_i, K)}$
- Since key **K** is fixed, each block is subject to the same permutation
  - (As though we had a “code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

ECB Encryption



Electronic Codebook (ECB) mode encryption

ECB Decryption



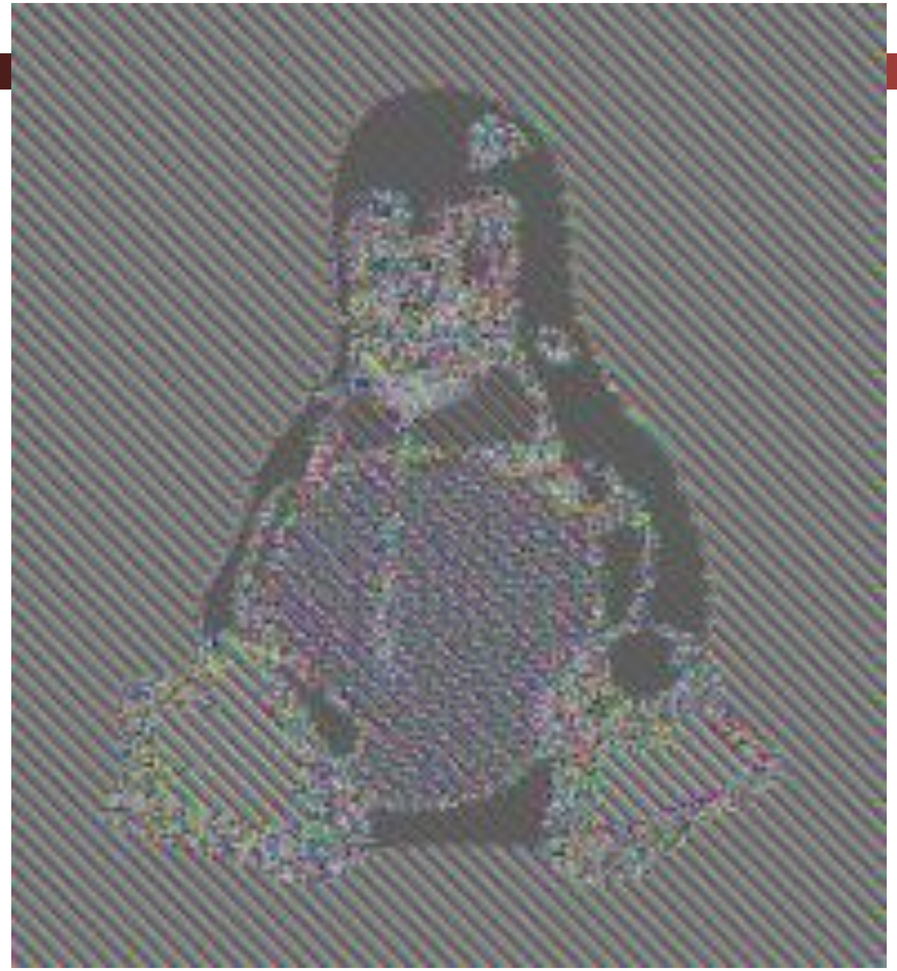
Electronic Codebook (ECB) mode decryption

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

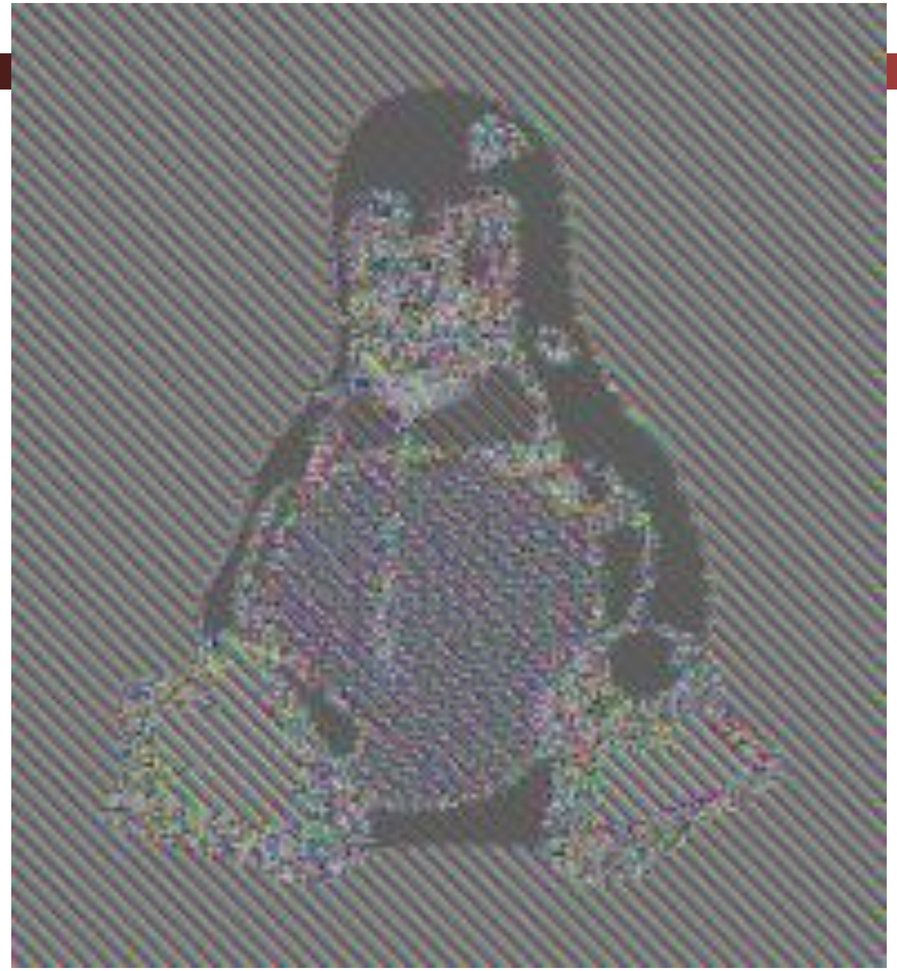
IND-CPA and ECB?

- 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) which we will discuss next time