Cryptography 1

Introduction to Offensive Security

What is Cryptography

- Transmitting a message between 2 parties without having a listener be able to tell what's being said
    - Alice wants to talk to Bob
    - But Eve is eavesdropping!
    - How can Alice talk with Bob privately?

Classical Ciphers

- Substitution ciphers
  - Bijective map from each character to another
    - Why bijective?
  - Monoalphabetic
  - Let a->z, b->y, c->x, …
    - hello
    - svool

Classical Ciphers

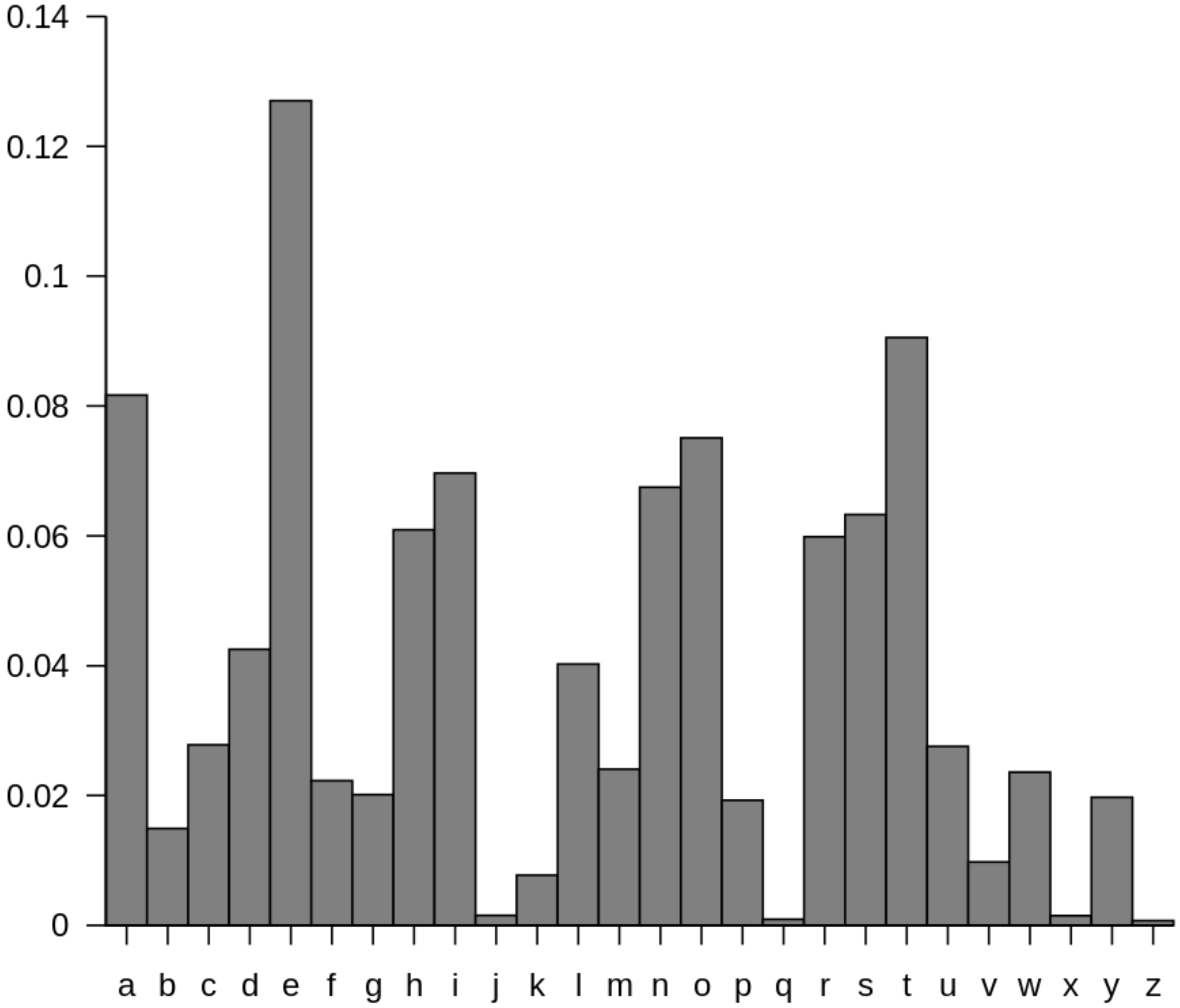
- Caesar cipher
  - a.k.a. shift cipher
  - A simple family of substitution ciphers
  - Each character maps to the character some number of positions down the alphabet
  - If shift is 1: a->b, b->c, c->d, etc.
    - Hello!
    - Ifmmp!
  - ROT13

Classical Ciphers

- Breaking a Caesar cipher
  - There are only 26 options…
  - Try them all!
- But we can do better!
- Frequency analysis

Frequency Analysis

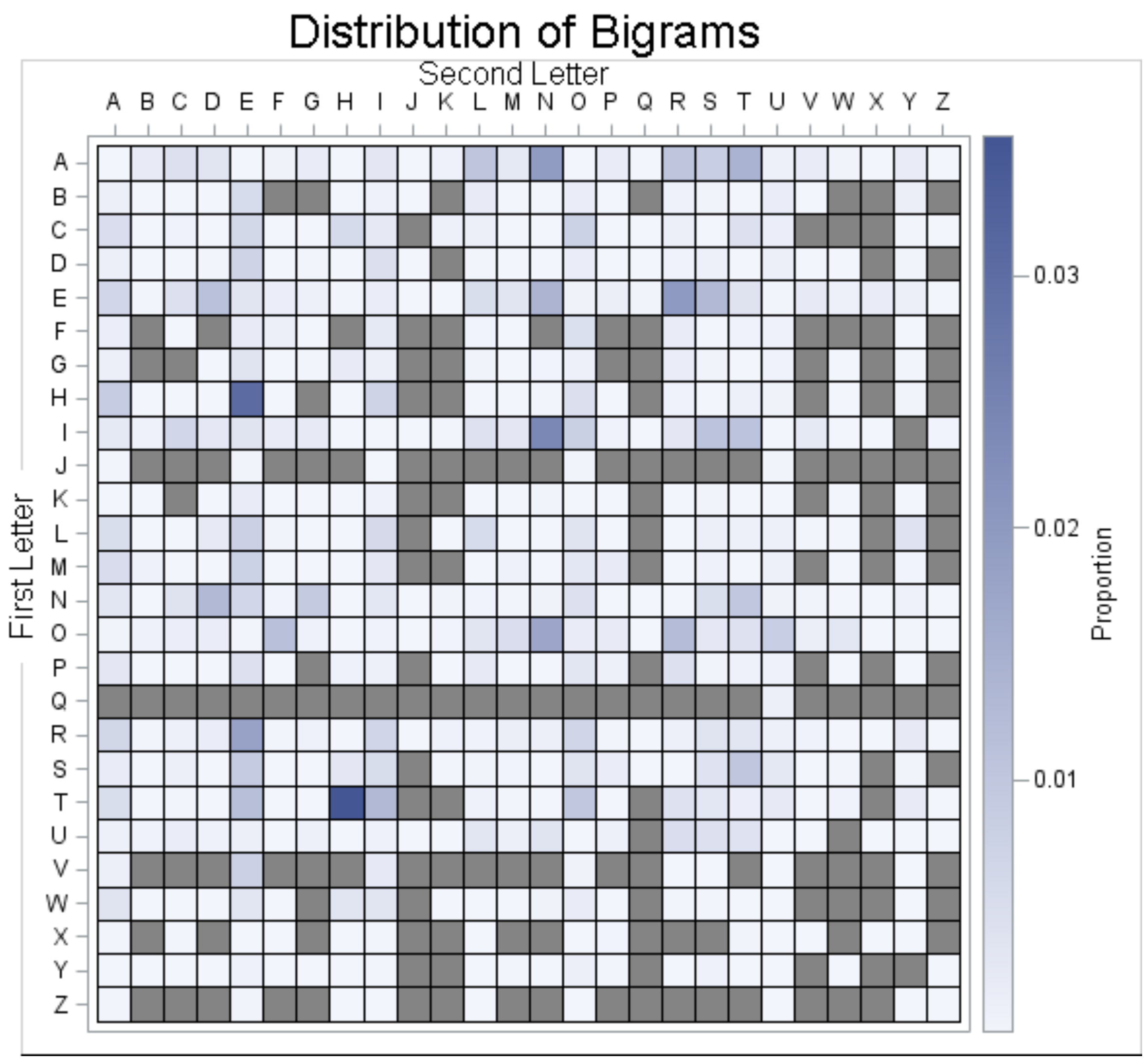
- Some characters are used more than others
- For English, " " (space) and "e" are super common characters while "j", "q", "x", and "z" are almost never used
- For a large enough ciphertext, the frequency distribution should roughly match the general English distribution
- Then, we just have to match the letters together!


0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
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

Frequency Analysis

- This can also be extended to more than one character: n-gram analysis
  - Bigram analysis: matching the distribution of character pairs
  - https://en.wikipedia.org/wiki/Bigram#Bigram_frequency_in_the_English_language

English bigram frequencies



Polyalphabetic Ciphers

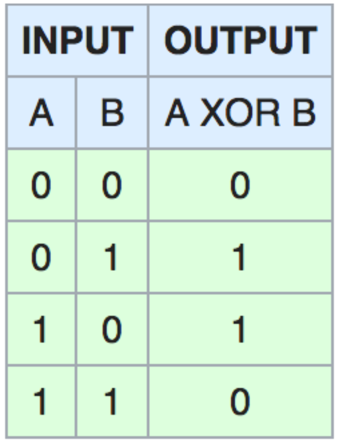
- One character no longer always encrypts to the same result!
    - The result somehow combines the message and a key
- Vigenère cipher
    - Shift cipher where the offset is based on the current character in the key
    - Message is "ABC", key is "ABC"
        - A->A
        - B->C
        - C->E
        - Ciphertext is "ACE"

Vigenère square

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

XOR

- eXclusive OR
  - You probably remember this from one of *many* prior classes
  - Output is 1 if A *or* B is 1, but not if both are
  - ⊕ is the symbol to represent the operation
    - Programming languages usually use ^
  - Has some nice properties:
    - A⊕B⊕A = B
      - It is its own inverse
    - (A⊕B)⊕C = A⊕(B⊕C)
      - Commutativity



INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

XOR

- $A \oplus B \oplus A = B$
  - It is its own inverse
- If A is a key and can be transmitted securely, we can use this to encrypt!
- Example
  - Alice wants to send m=13 to Bob
  - They've agreed on k=60
  - Alice computes 13 ^ 60 = 49, and sends it to Bob
  - Bob computes 49 ^ 60 = 13.
  - Eve meanwhile only sees 49!

XOR

- Example 2
  - Alice wants to send m="Hello!" ([72, 101, 108, 108, 111, 33]) to Bob
  - They've agreed on k="foobar" ([102, 111, 111, 98, 97, 114])
  - Alice computes 72^102=46, 101^111=10, etc. and sends it to Bob
  - Bob computes 46^102=72, 10^111=101, etc.
  - Eve only sees [46, 10, 3, 14, 14, 83]
- This is a One Time Pad (OTP) which is perfectly secure
  - As long as you don’t reuse the one time pad
  - As long as the one time pad is truly random
- However, this relies on k being transmitted securely out of band
  - So why not transfer m this way?

XOR

- So let's use a single byte!
- Alice wants to send m="Hello!" ([72, 101, 108, 108, 111, 33]) to Bob
- They've agreed on k=60
- Alice computes 72^60=116, 101^60=89, etc. and sends it to Bob
- Bob computes 116^60=72, 89^60=101, etc.
- Eve only sees [116, 89, 80, 80, 83, 29]
- But…

XOR

- Eve tries to decrypt [116, 89, 80, 80, 83, 29] with all 1 byte values
- One of them works! 60
- Could also use frequency analysis
  - Sometimes NULL byte (or another constant) is common as well…
  - 0^k = k, so a large chunk of the same values may be the key…
- This breaks the cryptosystem entirely as the security hinges on a single byte which can be quickly brute forced

XOR

- Ok, what about a repeating multi-byte key?
- Alice and Bob agree on k="SecretKey" ([83, 101, 99, 114, 101, 116, 75, 101, 121])
- For the sake of analysis, lets say m={contents of Alice in Wonderland}
- How can we break this?

XOR

- k="SecretKey" ([83, 101, 99, 114, 101, 116, 75, 101, 121])
- If len(k) is known, create a new ciphertext with every len(k)'th character
    - i.e. take char 0, 9, 18, 27, etc. in one group, 1, 10, 19, 28, etc. in another, …
- Then we can just perform our single-byte attack on each group

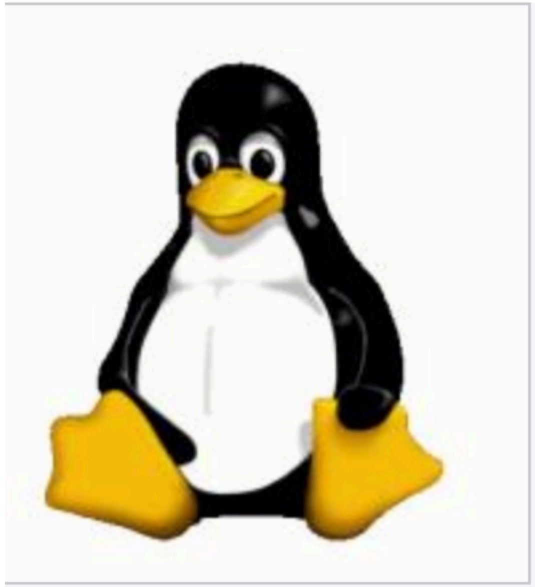
Symmetric Crypto

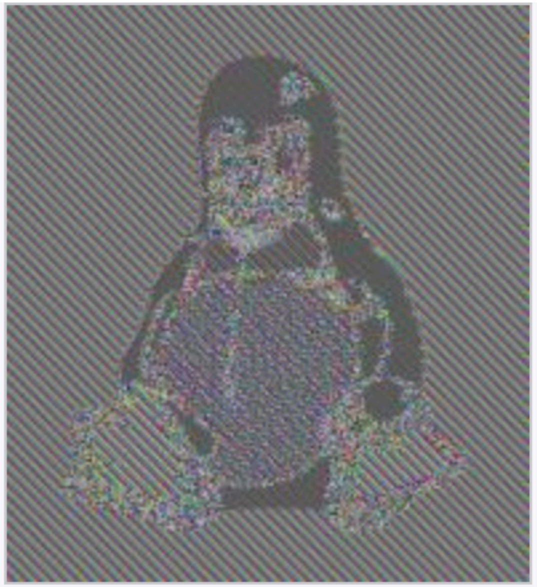
- Basic XOR is symmetric – both Alice and Bob use the same key k
- Other common symmetric crypto schemes:
    - RC4
    - DES
    - 3DES
    - AES

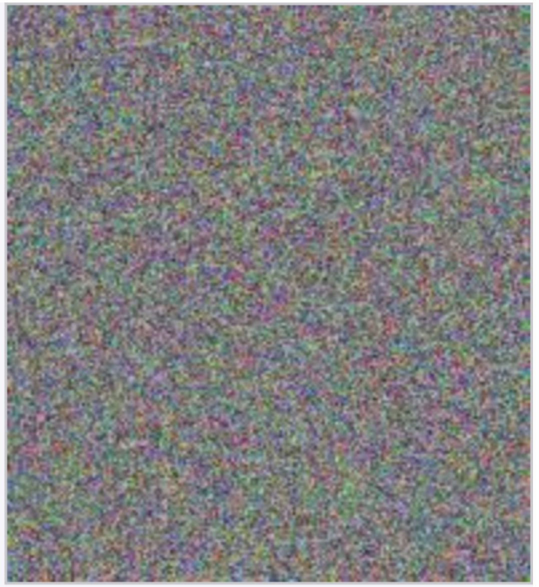
AES

- Block based symmetric encryption
- Input is chunked into blocks of 16, 24, or 32 bytes (128, 192, 256 bits)
- Multiple modes
  - ECB
    - Flaw: same input block results in same output block
  - CBC
    - Flaw: malleable. XOR on the ciphertext XORs the plaintext when decrypted
  - CTR
  - GCM
  - … many more

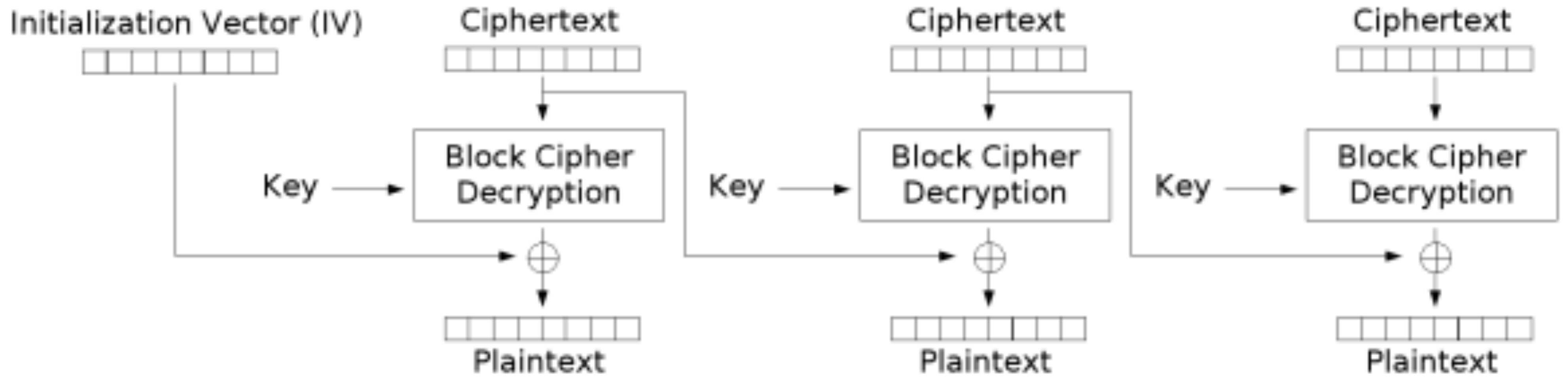
ECB Penguin


Original image


Encrypted using ECB mode


Modes other than ECB result in pseudo-randomness

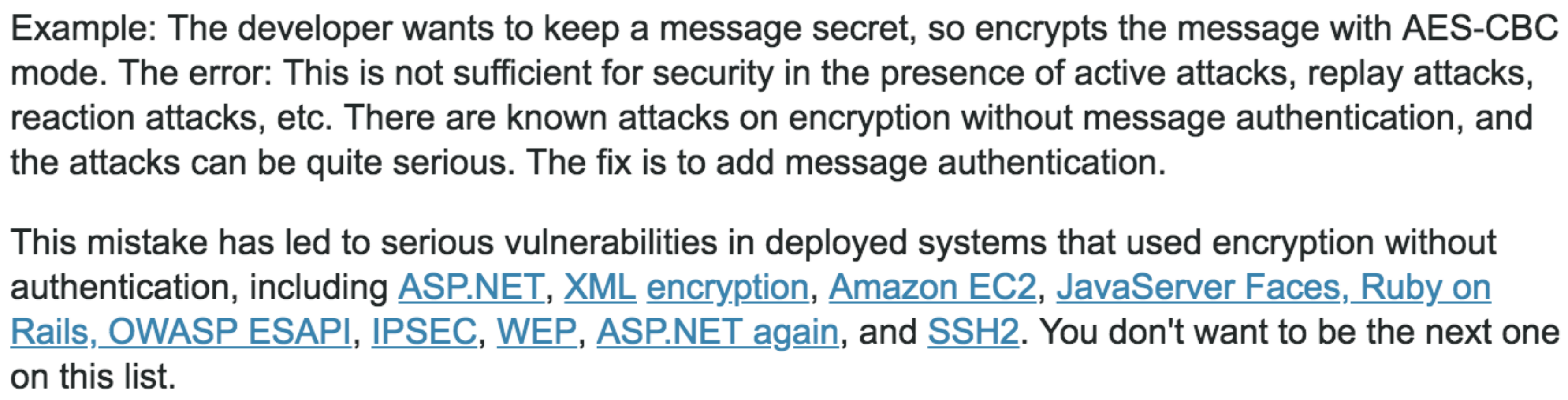
CBC Decryption Overview



Cipher Block Chaining (CBC) mode decryption

CBC Malleability Demo

Real-World CBC Malleability Attacks



Example: The developer wants to keep a message secret, so encrypts the message with AES-CBC mode. The error: This is not sufficient for security in the presence of active attacks, replay attacks, reaction attacks, etc. There are known attacks on encryption without message authentication, and the attacks can be quite serious. The fix is to add message authentication.

This mistake has led to serious vulnerabilities in deployed systems that used encryption without authentication, including ASP.NET, XML encryption, Amazon EC2, JavaServer Faces, Ruby on Rails, OWASP ESAPI, IPSEC, WEP, ASP.NET again, and SSH2. You don't want to be the next one on this list.

- https://security.stackexchange.com/questions/2202/lessons-learned-and-misconceptions-regarding-encryption-and-cryptology/2206#2206

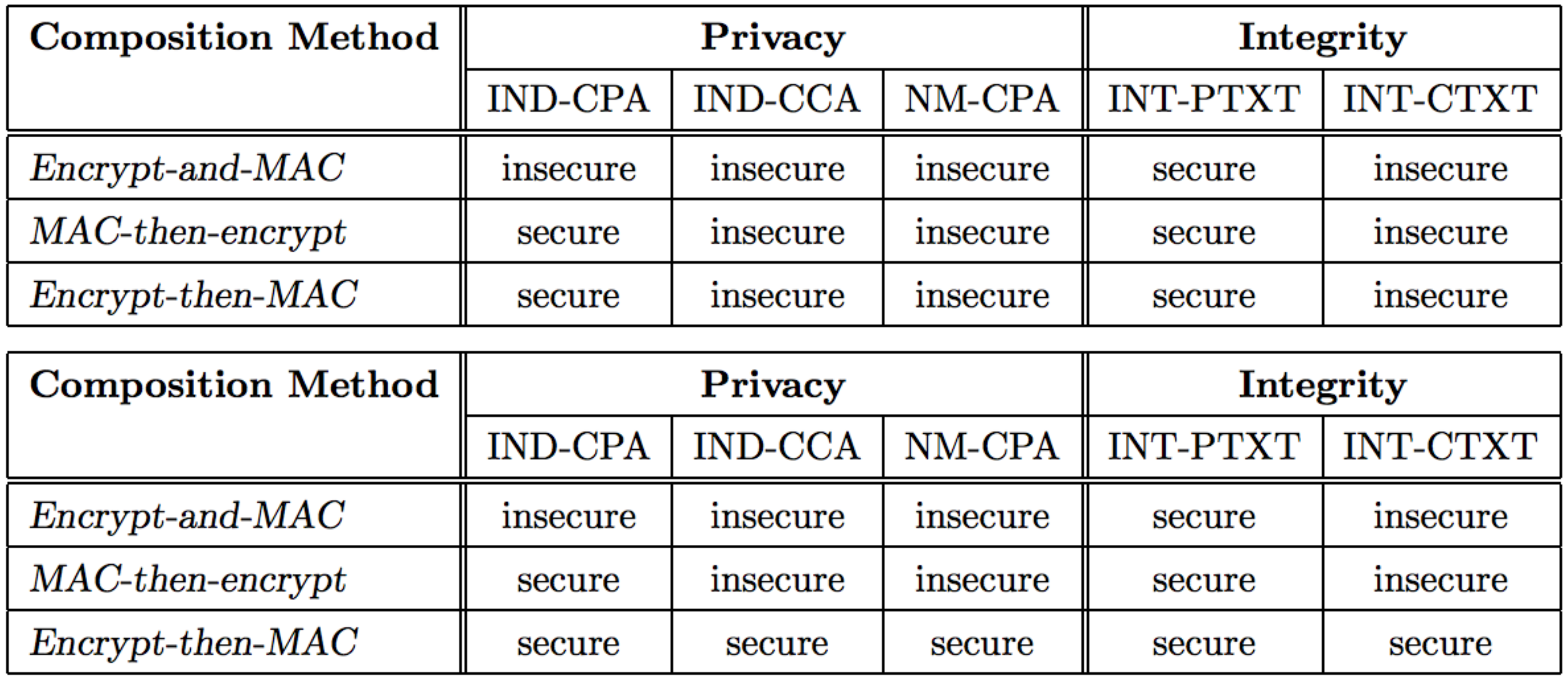
Solution to Malleability?

- Use *authenticated encryption*!
  - Many out-of-the-box cipher modes support authentication natively
  - GCM, EAX, CCM, OCB, …
- Do-it-yourself (less recommended):
  - We can use a *message authentication code* (MAC) to compute a hash of the message, and then use it to verify that the message was decrypted correctly
  - Encrypt-then-MAC:
    - Encrypt your data => C. Then compute MAC(C), and send C, MAC(C)
  - Encrypt-then-MAC works fine, but requires some care in implementing

Why not MAC-then-encrypt?

- We could also imagine instead taking a MAC of the *plaintext*, and then checking it after decryption
- Theoretical reason: Bellare and Namprempre, 2000
- "Cryptographic Doom Principle": if you have to perform *any* cryptographic operation before validating the MAC on a message, it will *somehow* likely lead to doom (Moxie Marlinspike, 2011)
    - https://moxie.org/blog/the-cryptographic-doom-principle/
- Examples from the article:
    - Vaudenay Attack
    - Attack on SSH

Bellare and Namprempre



Composition Method	**Privacy**			**Integrity**	
	IND-CPA	IND-CCA	NM-CPA	INT-PTXT	INT-CTXT
Encrypt-and-MAC	insecure	insecure	insecure	secure	insecure
MAC-then-encrypt	secure	insecure	insecure	secure	insecure
Encrypt-then-MAC	secure	insecure	insecure	secure	insecure

Composition Method	**Privacy**			**Integrity**	
	IND-CPA	IND-CCA	NM-CPA	INT-PTXT	INT-CTXT
Encrypt-and-MAC	insecure	insecure	insecure	secure	insecure
MAC-then-encrypt	secure	insecure	insecure	secure	insecure
Encrypt-then-MAC	secure	secure	secure	secure	secure

Figure 2: Summary of security results for the composite authenticated encryption schemes. The given encryption scheme is assumed to be IND-CPA for both tables while the given MAC is assumed to be weakly unforgeable for the top table and strongly unforgeable for the bottom table.

Asymmetric Crypto

- a.k.a. public key cryptography
- Involves 2 distinct keys: a "public" key and a "private" key
  - Should be obvious which one you need to keep secret…
- When you encrypt with one, it becomes readable only by the other
  - Works both ways
  - Encrypting with public key makes it decipherable only to those with the private key
  - "Encrypting" with private key can be used to sign messages
- Commonly used to bootstrap symmetric crypto algorithms as they are faster

RSA

- The most well known asymmetric cryptosystem
- Let's walk through a sample encrypt/decrypt cycle

RSA Example (“textbook” RSA)

- Alice generates a public/private key pair
  - Generates 2 (large) primes: p and q
  - Computes n=pq. This is the <u>public modulus</u>
  - Computes the totient of n: phi = (p-1)(q-1)
  - Pick a number e coprime to phi (usually 3, 5, 17, 257, 65537)
  - Compute d as the **multiplicative inverse** of e mod phi

  - (n,e) is the public key
  - (n,d) is the private key

RSA Example ("textbook" RSA)

- Bob now wants to talk with Alice
- Alice sends Bob the public key
- Bob encrypts his message m by raising it to the public exponent (e) mod n
  - $c = m^e \pmod{n}$
- Bob then sends the ciphertext c to Alice
- Alice decrypts it by raising c to the private exponent (d)
  - $m = c^d \pmod{n}$
- This turns into:
  - $(m^e)^d \pmod{n} \Rightarrow m^{ed} \pmod{n} \Rightarrow m^1 = m$

Simple RSA Attacks

- There are many, many ways to mess up implementing RSA
- A great overview of attacks:
  - Dan Boneh: “Twenty years of attacks on the RSA Cryptosystem”
  - For RSA-based CTF challenges, maybe 90% of them will be based on an attack from this paper
- We’ll only cover a few of the simpler ones in class

Factoring

- Factoring the product of two large prime numbers is generally considered to be very hard
  - Theory note: although it is *not* known to be NP-Hard
    Probably lies in some complexity class between P and NP, but we don't know for sure.
  - Future note: quantum computing, if realized, will change this. Shor's algorithm is a polynomial time algorithm for factoring – as long as you've got a quantum computer!
- If n is sufficiently small, we can just factor it!
- Current best algorithm: General Number Field Sieve (GNFS)

Small Modulus Attack

- Recall that to encrypt we do
    - m^e (mod N)
- Q: does the attacker know e?

Small Modulus Attack

- Recall that to encrypt we do
  - m^e (mod N)
- Q: does the attacker know e? **YES**
- (N,e) is the *public key* – everyone knows it
- What happens if $m^e < N$?

Small Modulus Attack

- Recall that to encrypt we do
  - $m^e \pmod N$
- Q: does the attacker know e? **YES**
- (N,e) is the *public key* – everyone knows it
- What happens if $m^e < N$?
  - We can just take the e-th root to recover m!