EECS 388

Introduction to Computer Security

Lecture 4:

Confidentiality

September 5, 2024

Prof. Halderman

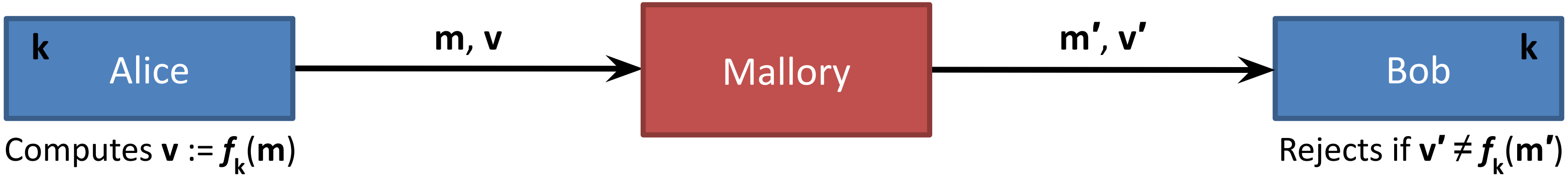
Review: Message Integrity

Problem: Integrity of message from Alice to Bob over an ***untrusted channel***

Approach: Alice must append bits to message that only Alice (or Bob) can make

Ideal solution: **Random functions**

Practical solution:



Pseudorandom functions (PRFs)

f_k() is indistinguishable in practice from random, unless you know **k**

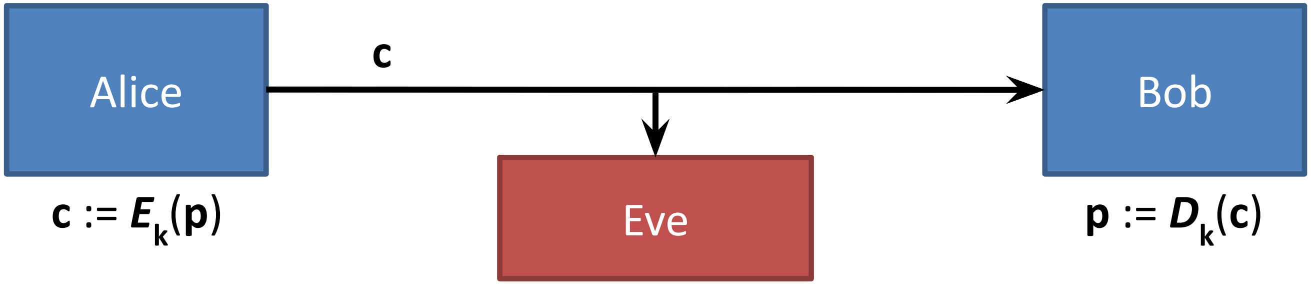
The **HMAC construction** turns a **cryptographic hash function** (e.g., **SHA-256**) into a **Message authentication code (MAC)**

For most practical purposes, we believe we can use **HMAC-SHA-256** as a PRF.

New Goal: Confidentiality

Confidentiality: Keep the content of message **p** secret from an *eavesdropper*

Approach and threat model:



Passive Eavesdropper
(a kind of **passive attacker**)

sees everything on the channel
but can't change anything

Terminology:

- **p** plaintext
- **c** ciphertext
- **k** secret key
- $E_k()$ encryption function
- $D_k()$ decryption function

Digression: Historical Cryptography

Caesar cipher

First recorded use: Julius Caesar (100-44 BC)

Replace each plaintext *letter* with one a fixed number of places down the alphabet:

Encryption: $\mathbf{c}_i := (\mathbf{p}_i + \mathbf{k}) \mod 26$

Decryption: $\mathbf{p}_i := (\mathbf{c}_i - \mathbf{k}) \mod 26$

Examples using **k**=3: (that's the key Caesar used!)

```
 p: ABCDEFGHIJKLMNOPQRSTUVWXYZ
+k: 33333333333333333333333333
=c: DEFGHIJKLMNOPQRSTUVWXYZABC

 p: fox   go wolverines
+k: 333   33 3333333333
=c: ira   jr zroyhulqhv
```

[How would you break the Caesar cipher?]

Cryptanalysis of the Caesar cipher

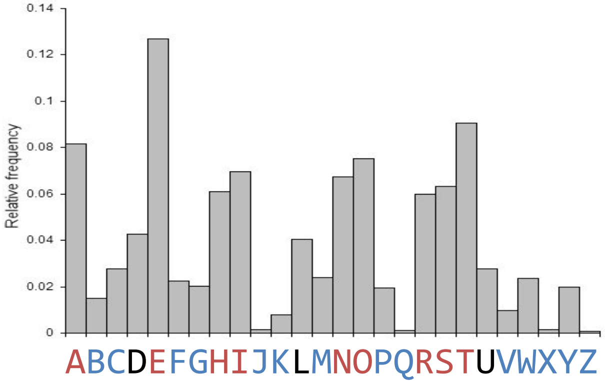
Only 26 possible keys:

Try every possible **k** by "brute force"

Practical to do by hand!

How can a computer recognize the right one?

One solution: **Frequency analysis**. English text has characteristic letter frequency distribution:



Recognize with, e.g., chi-squared (χ^2) test

Digression: Historical Cryptography

Later advance: Vigenère cipher c. 1553

«*le chiffre indéchiffrable*» (“the indecipherable cipher”)

Encrypts successive letters using a sequence of Caesar ciphers keyed by the letters of a keyword.

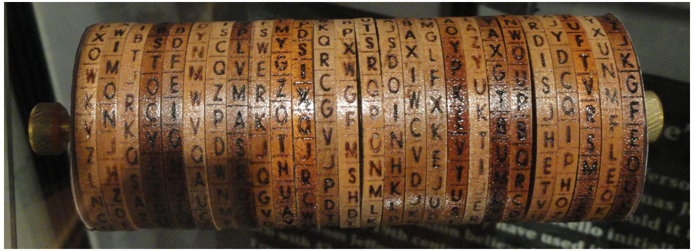
For an **n**-letter “key word” **k**,

Encryption: $\mathbf{c_i} := (\mathbf{p_i} + \mathbf{k_{i \bmod n}}) \bmod 26$

Decryption: $\mathbf{p_i} := (\mathbf{c_i} - \mathbf{k_{i \bmod n}}) \bmod 26$

Example: **k**=ABC (i.e., $\mathbf{k_0}=0$, $\mathbf{k_1}=1$, $\mathbf{k_2}=2$)

```
 p:  bbbbbb   amazon
+k:  012012   012012
=c:  bcdbcd   anczpp
```



[Break *le chiffre indéchiffrable*?]

Cryptanalysis of the Vigenère cipher

Easy, if we know the keyword length, **n**:

1. Divide ciphertext into **n** slices
2. Solve each slice as a Caesar cipher

How to find **n**? One way: **Kasiski method**

Published 1863 by Kasiski (earlier known to Babbage?)

Repeated strings in long plaintext will sometimes, by coincidence, be encrypted with same key letters:

```
 p:  CRYPTOISSHORTFORCRYPTOGRAPHY
+k:  ABCDABCDABCDABCDABCDABCDABCD
=c:  CSASTPKVSIQUTGQUCSASTPIUAQJB
```

Distance between repeats is (likely) a multiple of key size.

(ex: 16⇒16,8,**4**,2,1). Use multiple repeats to narrow down

Another way: Iterate over **n** to find best match

(e.g., minimize sum of χ^2 for the individual Caesar ciphers)

One-time Pad (OTP)

Back to the present…

How can we achieve confidentiality securely?

One-time pad (OTP)

Alice and Bob share a secret, very long string of random bits (a "one-time pad") **k**

Encryption: $\mathbf{c_i} := \mathbf{p_i} \oplus \mathbf{k_i}$

Decryption: $\mathbf{p_i} := \mathbf{c_i} \oplus \mathbf{k_i}$

XOR Facts

a	b	a ⊕ b
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

$a \oplus b \oplus b = a$

$a \oplus b \oplus a = b$

Pro: ***Provably secure***

Information-theoretically secure

(no computational complexity assumptions)

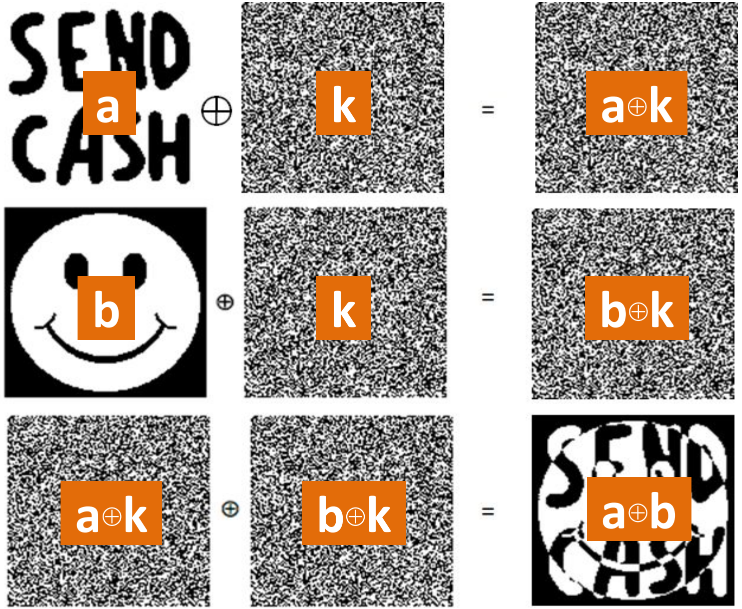
First proof published by Claude Shannon in 1949

[Show Mallory can't do better than guessing]

Con: ***Usually impractical*** [Why?] [Exceptions?]

Caution "one-time" means you MUST NEVER reuse any part of the pad

If you do: Let $\mathbf{k_i}$ be pad bit. From ciphertexts ($\mathbf{a} \oplus \mathbf{k_i}$) and ($\mathbf{b} \oplus \mathbf{k_i}$), attacker learns $\mathbf{a} \oplus \mathbf{b}$. [How's that useful?]



Stream Cipher: Use a PRG for Confidentiality

More practical approach to confidentiality:

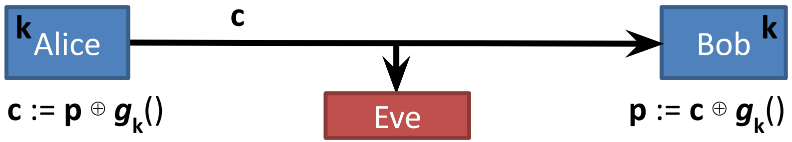
Use a **pseudorandom generator (PRG)** instead of a truly random pad

Recall: A PRG $g_k()$ is practically indistinguishable from a random stream of bits, unless you know **k**.

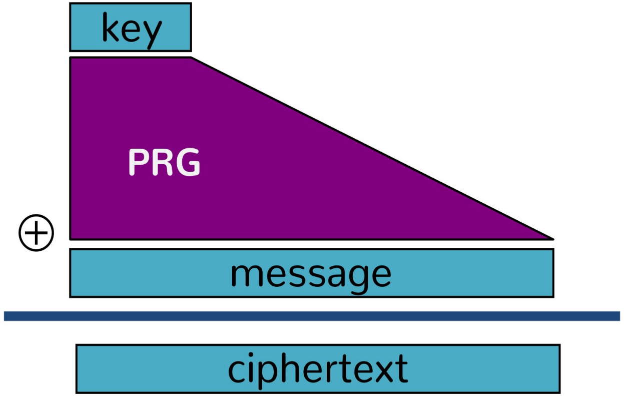
Called a **stream cipher**

Alice and Bob choose PRG $g()$, share secret key **k**

Encryption: $\mathbf{s} := g_k();\ \mathbf{c}_i := \mathbf{p}_i \oplus \mathbf{s}_i$
Decryption: $\mathbf{s} := g_k();\ \mathbf{p}_i := \mathbf{c}_i \oplus \mathbf{s}_i$



Caution! **NEVER reuse keys**
NEVER reuse PRG output bits



Provably secure if $g()$ is a secure PRG, under complexity assumptions

However… we don't know how to prove that secure PRGs even exist!

Best we can do (again): Use well studied functions where we haven't spotted a problem yet

Examples: ~~RC4~~ **ChaCha20**

Block Ciphers

Another approach:

Block ciphers

consist of a function that **encrypts** fixed-size (**n**-bit) blocks with a reusable key **k**:

$$E_k(p) : \{0,1\}^{|k|} \times \{0,1\}^n \rightarrow \{0,1\}^n$$

and an inverse function that **decrypts** the blocks when used with same key:

$$D_k(c) = E_k^{-1}(c) : \{0,1\}^{|k|} \times \{0,1\}^n \rightarrow \{0,1\}^n$$

such that $\forall k : D_k(E_k(p)) = p$.

In effect, **k** selects one *permutation* from the set of $2^n!$ possible permutations of E's domain.

A block cipher is ***different*** from a PRF. [Why?]

What do we want, if not a PRF?

Pseudorandom permutation (PRP)

A secure PRP is a function that cannot practically be distinguished from a truly random permutation unless you know **k**. (Similar to the PRF game)

Annoying question again:

Do PRPs *actually exist*?

Same annoying answer:

We don't know. :(

Best we can do:

Design a complex function that's invertible if and (hopefully) only if you know **k**

Examples: ~~DES~~ **AES**

AES Block Cipher

Today's most common block cipher:

AES (Advanced Encryption Standard)

aka **Rijndael**, for its designers, Rijmen and Daemen

- Standardized by **NIST** in 2001 after winning a long, public international design competition
- Efficient in both software and hardware. Hardware-accelerated in many modern CPUs
- Widely believed to be a secure PRP (but we don't know how to prove it)

Fixed block size: 128 bits
Variable key size: 128, 192, or 256 bits

10, 12, or 14 **rounds** (based on key size)

Generates **r := #rounds** ***subkeys*** from **k**.
Performs same set of operations **r** times, each with a different subkey

Each AES round

128-bits input
128-bit subkey
128-bit output

picture as operations on a 4×4 grid of 8-bit values

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

Four steps:

1. Non-linear substitution
   Run each byte thru a nonlinear function (lookup table)
2. Shift rows
   Circular-shift each row: **i**th row shifted by **i** (0–3)
3. Linear-mix columns
   Treat each column as a 4-vector;
   multiply by a constant invertible matrix
4. Key-addition
   XOR each byte with corresponding byte of round subkey

To decrypt, just undo the steps, in reverse order

Padding and Block Cipher Modes

Challenge for block ciphers:

How to encrypt **arbitrary-sized messages**?

Padding: Add bytes to end of message to make it a multiple of block size

Flawed approach: add zeros [What's the issue?]

| MM MM MM MM MM 00 00 00 |

Don't know what to remove after decryption!

Better approach (**PKCS7**): Add **n** bytes of value **n**

| MM MM MM MM MM 03 03 03 |

Edge case: Message that ends at block boundary?

| MM MM MM MM MM MM MM MM | 08 08 08 08 08 08 08 08 |

Add an **entire block** of padding

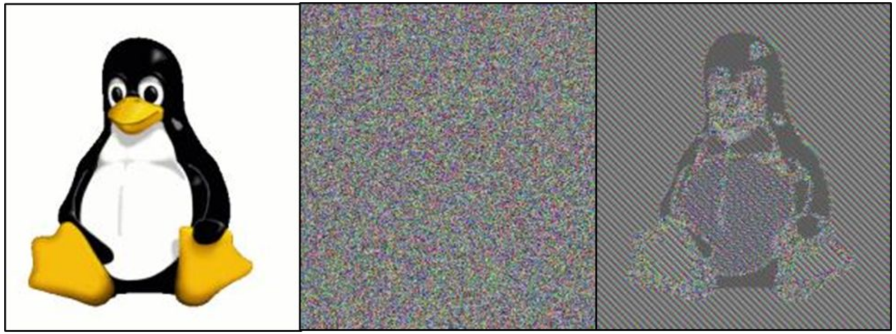
Ensures receiver can ***unambiguously*** distinguish the padding from the message after decrypting

Cipher modes: Algorithms for applying block ciphers to more than one block

Flawed approach: [What's the issue?]

Encrypted codebook (ECB) mode

Simply encrypt each block independently: $\mathbf{c_i} := E_k(\mathbf{p_i})$



Plaintext Pseudorandom ECB mode

More Cipher Modes

Cipher-block chaining (CBC) mode

"Chains" ciphertexts to obscure later ones

Choose a **random** **initialization vector** **IV**

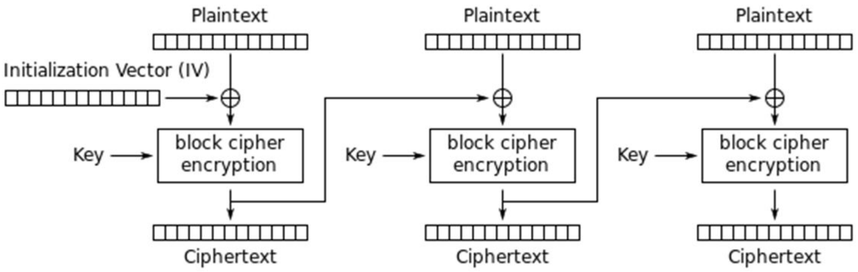
Encrypt: $\mathbf{c_0} := \mathbf{IV}$; $\mathbf{c_i} := E_k(\mathbf{p_i} \oplus \mathbf{c_{i-1}})$

Decrypt: $\mathbf{p_i} := D_k(\mathbf{c_i}) \oplus \mathbf{c_{i-1}}$,

[Why do we need the IV?]

Have to send IV with ciphertext

Can't encrypt blocks in parallel or out of order



Counter (CTR) mode

Turns a block cipher into a stream cipher

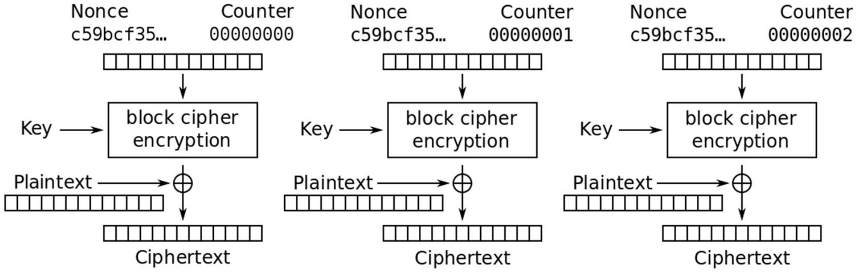
Generate **keystream** **s** for **k** and **unique** **nonce**:

$\mathbf{s} := E_k(\mathbf{nonce}\|0) \,\|\, E_k(\mathbf{nonce}\|1) \,\|\, E_k(\mathbf{nonce}\|2) \,\|\, \ldots$

Encrypt: $\mathbf{c} := \mathbf{p} \oplus \mathbf{s}$ Decrypt: $\mathbf{p} := \mathbf{c} \oplus \mathbf{s}$

Benefits: Doesn't require padding
Efficient parallelism/random access

Caution: Never reuse **nonce** for same **k**!



Getting both confidentiality and integrity?

Integrity (~~tampering~~)

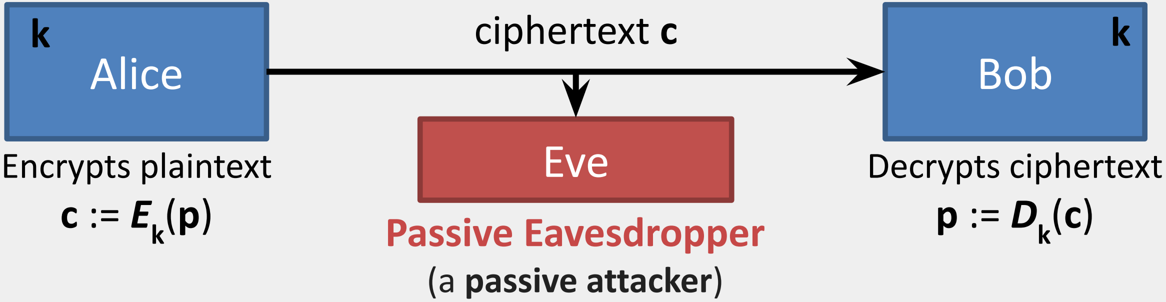
Let $f()$ be a secure PRF.

In practice: e.g., **HMAC-SHA-256**

Confidentiality (~~eavesdropping~~)

Construct $E()$ and $D()$ from secure PRG (a stream cipher) ***or*** secure PRP (a block cipher) with appropriate padding/cipher mode.

In practice: e.g., **AES-128 in CTR mode**



What if we want integrity and confidentiality *at the same time*?

(Next lecture!)

Coming Up

Reminders:

Lab Assignment 1 due TODAY at 6 p.m.

Quiz on Canvas after every lecture

Project 1, Part 1 due next Thursday at 6 p.m.

Tuesday

Combining Confidentiality and Integrity

Confidentiality attacks,
authenticated encryption

Thursday

Public Key Cryptography

Diffie-Hellman key exchange,
RSA encryption,
digital signatures