Stream Ciphers

January 25, 2022

Today

1. **perfect security** *(but don't get your hopes up, it's not practical)*
2. **practical security**:
   commonly used **stream ciphers** (RC4, Salsa20 / ChaCha20)

Thank you for filling out the background survey online!

Feedback: https://forms.gle/JGbNCmCsU69iWaTv8

Motivating Example: Web Security

`www.example.com`

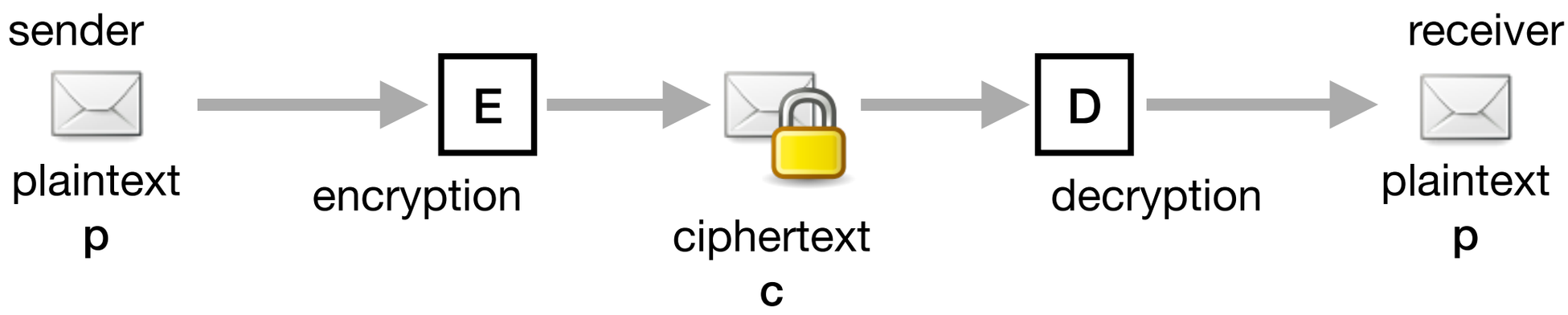
web browser

non-secure channel

web server

Reminder: Encryption

- Model
  - plaintext **p**
  - encryption algorithm (cipher) **E**
  - ciphertext **c**
  - decryption algorithm (cipher) **D**



Kerckhoffs's Principle

- Design requirement for ciphers (1883):

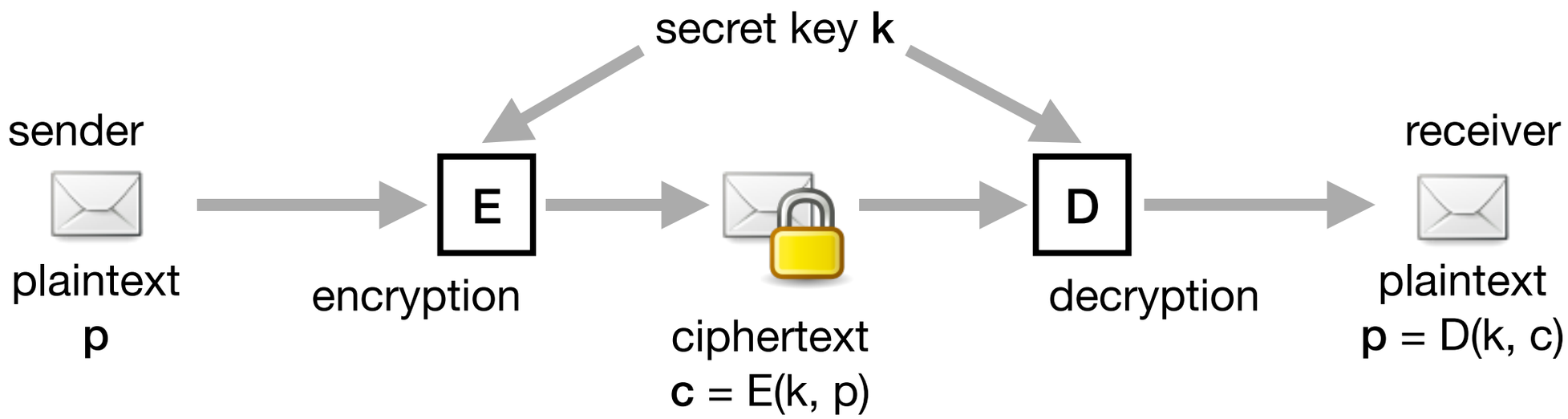
  "*A cryptographic system should be secure even if all of its details, except for the key, are publicly known.*"

- Rejection of security by obscurity for cryptography

Auguste Kerckhoff

Reminder: Encryption

- Model
  - plaintext **p**
  - secret key **k**
  - encryption algorithm (cipher) **E**
  - ciphertext **c** = E(k, p)
  - decryption algorithm (cipher) **D**



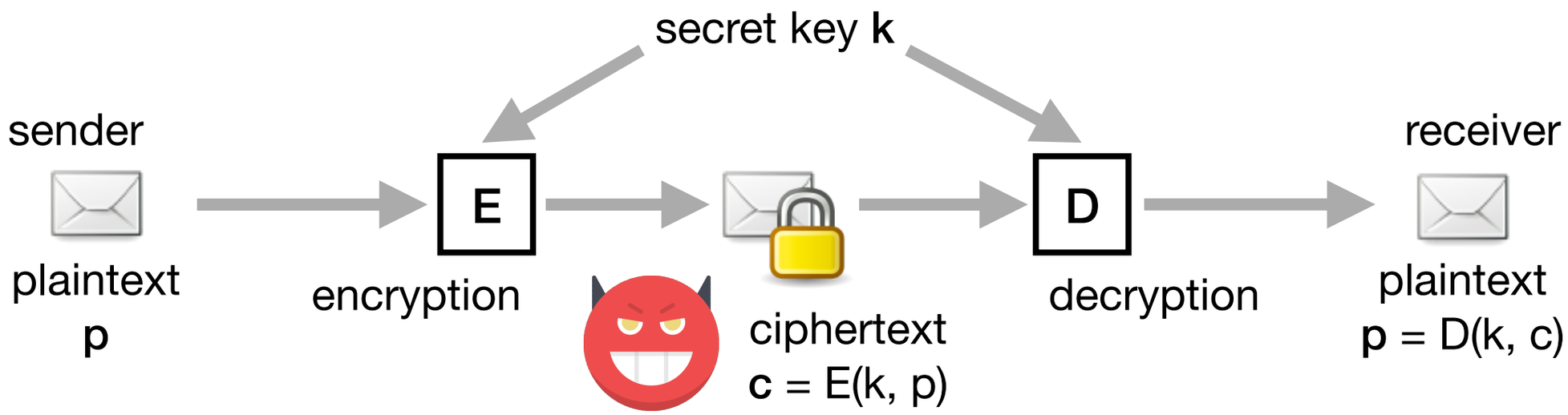
Example Sources of True Randomness

How to generate truly random keys on a computer?

- Clock drift
  - modern computers often have more than one crystal oscillator for timing (*e.g.*, for real-time clock, for CPU)
  - clock crystals are not precise, and there can be **random variations in their speeds**
- Disk drives
  - hard disk drives have small **random fluctuations in their rotational speeds** due to chaotic air turbulence
  - measuring disk seek-time captures this randomness
- Linux: `/dev/random`
  - multiple sources (*e.g.*, mouse and keyboard activity, disk I/O operations, specific interrupts)
  - similar sources on other operating systems
- …

Reminder: Encryption

- Model
  - plaintext **p**
  - secret key **k**
  - encryption algorithm (cipher) **E**
  - ciphertext **c** = E(k, p)
  - decryption algorithm (cipher) **D**



- Attacker's goal: recover the secret key or the plaintext
- None of the classic cryptographic algorithms are secure

How to define security?

How to define security?

- First idea:
  "*Attacker cannot recover the secret key*"
  - **E(k, p) = p** would be considered secure under this definition
- Second idea:
  "*Attacker cannot recover the complete plaintext*"
  - suppose that **E(k, p)** is secure
  - then, **E'(k, p_1 | p_2) = p_1 | E(k, p_2)** would also be secure
- Third idea:
  "*Attacker cannot recover any part of the plaintext*"
  - what if some part of the plaintext is deterministic (*e.g.*, "`GET / HTTP/1.1 …`")?

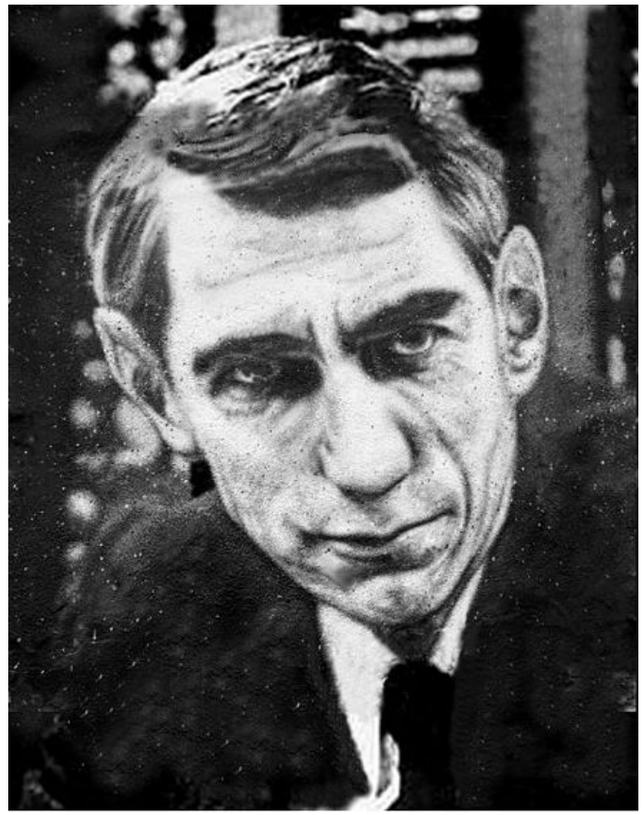
None of these ideas define security correctly 

Perfect Security

- Published by Claude Shannon in 1949, based on his classified report from 1945

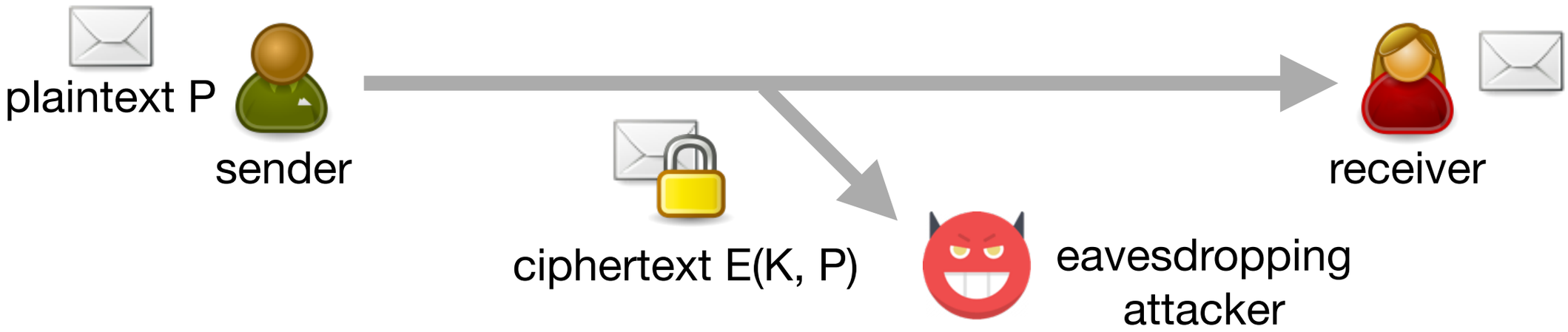
- **Perfect security:**
  "*after a cryptogram is intercepted by the enemy, the a posteriori probabilities of this cryptogram representing various messages be identically the same as the a priori probabilities of the same messages before the interception*"

  ≈ attacker gains no information about the plaintext from observing the ciphertext



Claude Shannon

Perfect Security Formally



- Perfect security

$$\Pr[P = p] = \Pr[P = p \mid E(K, P) = c]$$

probability that plaintext P is a particular message p (without observing the ciphertext)

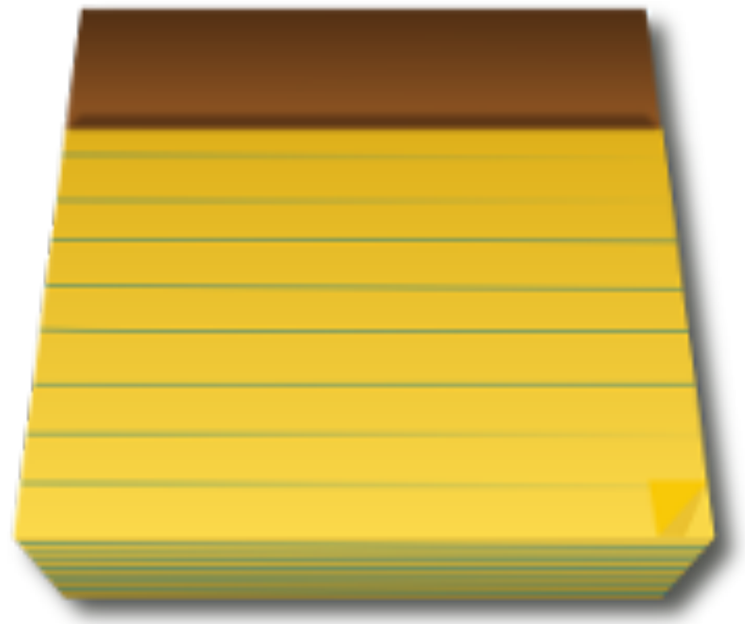
probability that plaintext P is a particular message p given that the ciphertext E(K, P) is c

- Equivalent to saying that

plaintext P and **ciphertext E(K, P)** are **independent**

One-Time Pad

- Works for binary data as well as alphabetic text
  - plaintext: sequence of bits or letters
- Key: **sequence of bits or letters**
  1. at least as long as the plaintext
  2. chosen uniformly at random
  3. used to encrypt only one message



- Encryption / decryption: modulo add / subtract each bit (or letter) of the key to the corresponding bit (or letter) of the plaintext / ciphertext
  - binary: $\mathbf{c_i = p_i \oplus k_i}$ (XOR operation, i.e., modulo 2 addition)
  - alphabetic: $\mathbf{c_i = p_i + k_i \bmod 26}$ and $\mathbf{p_i = c_i - k_i \bmod 26}$ (for 26 letters)

One-Time Pad
Binary Example

- XOR operation:

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

  - same as **x + y mod 2** addition (or subtraction since -1 = 1 mod 2)

- Encryption
  - *plaintext:* 0 1 0 1
  - *key:* 0 1 1 0
  - *ciphertext:* 0 0 1 1
- Decryption
  - *ciphertext:* 0 0 1 1
  - *key:* 0 1 1 0
  - *plaintext:* 0 1 0 1

One-Time Pad
Alphabet Example

- Encode space as the 27th letter and perform addition **modulo 27**

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	**SPACE**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	**26**

- plaintext: `mr mustard with the candlestick in the hall`
  key: `pxlmvmsydofuyrvzwc tnlebnecvgdupahfzzlmnyih`
  ciphertext: `ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS`

- plaintext: `miss scarlet with the knife in the library`
  key: `pftgpmiydgaxgoufhkllmhsqdqogtewbqfgyovuhwt`
  ciphertext: `ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS`

- Both keys are equally likely
  → attacker **cannot learn** which key and plaintext are the correct ones from observing the ciphertext

Security of One-Time Pad

- One-time pad is **perfectly secure**
- Proving perfect security:
  we have to show that for any plaintext **p** and ciphertext **c**,

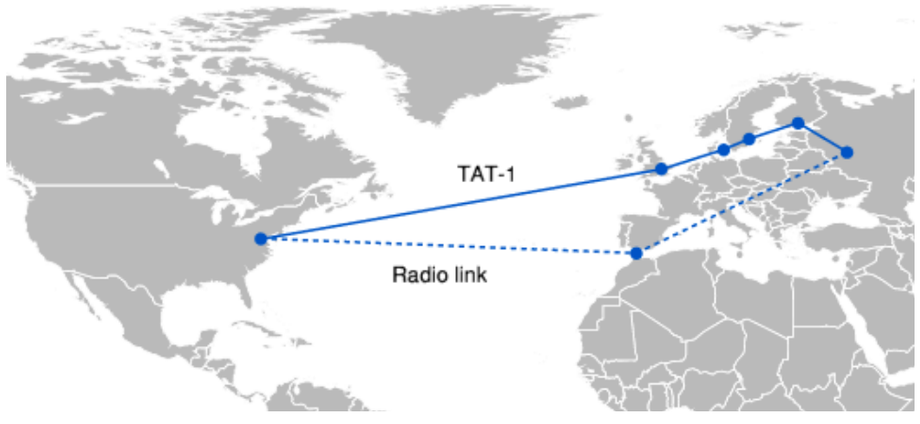
$$\Pr[P = p \mid E(K, P) = c] = \Pr[P = p]$$

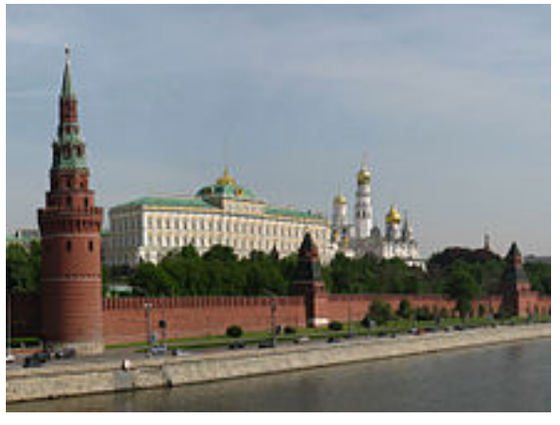
- *Proof for special case*: plaintext is 1-bit long (either 0 or 1)
  - key and ciphertext are also 1-bit long
  - given that the **ciphertext** is **0**, the probability that the **plaintext** is **0**:
    $\Pr[P = 0 \mid E(K, 0) = 0] = \Pr[P = 0 \mid K \oplus 0 = 0] = \Pr[P = 0 \mid K = 0] = \Pr[P = 0]$
  - given that the **ciphertext** is **0**, the probability that the **plaintext** is **1**:
    $\Pr[P = 1 \mid E(K, 1) = 0] = \Pr[P = 1 \mid K \oplus 1 = 0] = \Pr[P = 1 \mid K = 1] = \Pr[P = 1]$
  - …
  - similar argument works for the general case

One-Time Pad in Practice

- One-time pad has been used in special applications since early 20th century
  - early systems used pencil, paper, and mental arithmetic
  - reportedly used by various intelligence agencies and diplomatic services
- During World War II and the Cold War, Soviet security agencies made heavy use of one-time pads
  - keys were often printed on miniaturized pads made of highly-flammable paper

- Washington-Moscow hotline
  - established in 1963 between the Pentagon and the Kremlin





  - each country delivered keys on tapes via its embassy abroad

Limitations of Perfect Security

Why is one-time pad not the "end of cryptography?"

- For perfect security, we always need that

  length of key ≥ length of plaintext

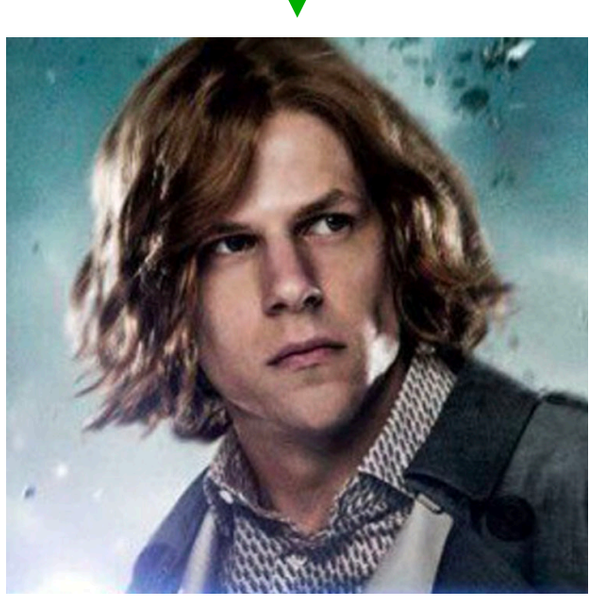
- *Practical problems*
  - distributing and storing long keys
  - generating long truly random keys

- Practical protocols, such as SSL (HTTPS), IPSec, or SSH, do **not use one-time pad**

Perfect security
(undefeatable)

“Perfect attacker”
(unbounded capabilities,
except for knowing the key)

Practical security

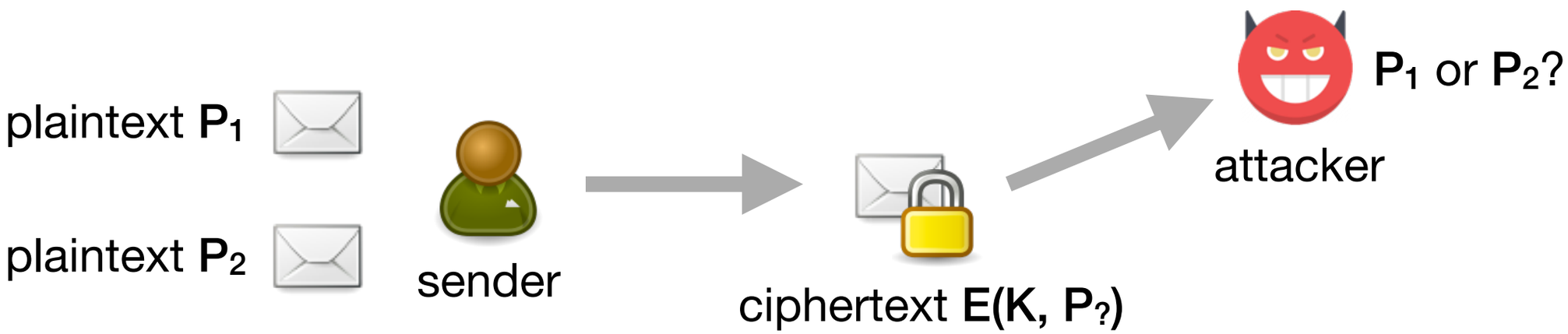


Practical attacker
(bounded computational power)

Semantic Security

- Proposed by Goldwasser and Micali in 1982
- Definition for chosen plaintext attacks (simplified):
    1. attacker chooses two plaintexts
    2. sender encrypts one of the two plaintexts (chosen at random)
    3. attacker observes ciphertext and must guess which plaintext was encrypted

encryption is **semantically secure** if the attacker's advantage for any **efficiently computable** guess is **negligible** over random guessing



Bad Idea:
"Many-Time Pad"

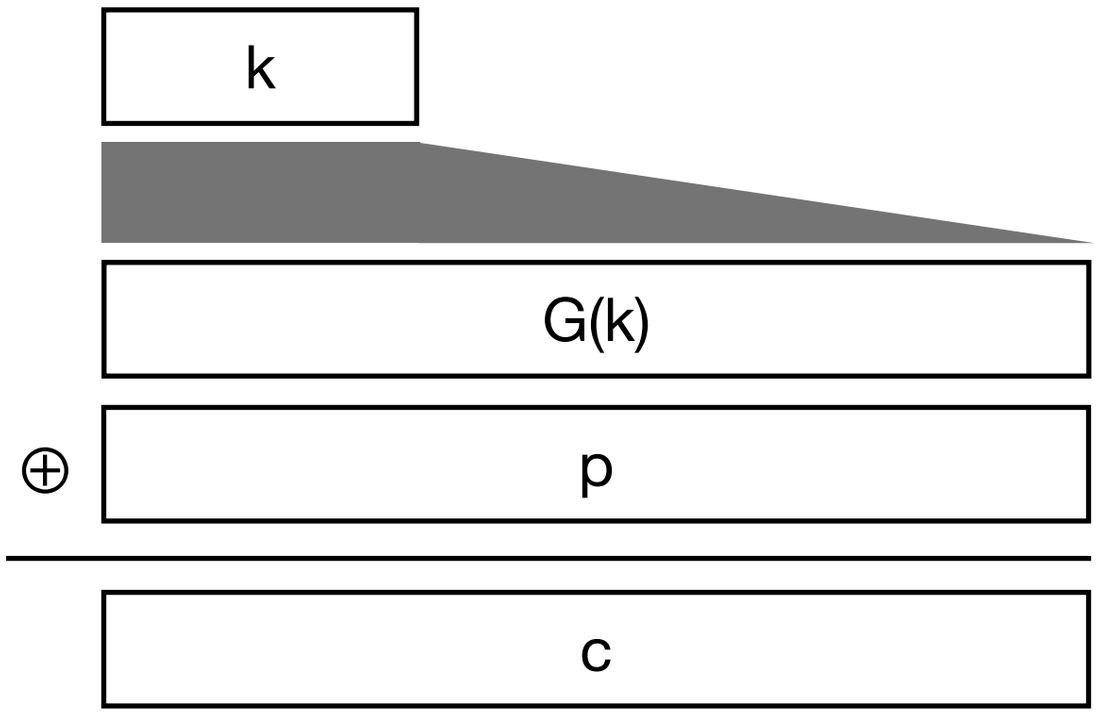
- *Idea*: simply **reuse the key** for multiple plaintexts
- *Example*
  - key: $k = 01010011$ $k = 01010011$
    plaintexts: $p_1 = 00111101$ $p_2 = 10110000$
    ciphertexts: $c_1 = 01101110$ $c_2 = 11100011$
  - attacker can get:
    $c_1 \oplus c_2 = 10001101$
    $c_1 \oplus c_2 = (p_1 \oplus k) \oplus (p_2 \oplus k) = p_1 \oplus p_2 \oplus k \oplus k = p_1 \oplus p_2$

- Problem: if the attacker knows some parts of a plaintext, it can recover the same parts of other plaintexts
  - *example*: attacker knows $p_1 = 00111101$ (*e.g.*, standard protocol format)
  - then, attacker can compute $p_1 \oplus c_1 \oplus c_2 = p_1 \oplus p_1 \oplus p_2 = 0 \oplus p_2 = p_2$

Stream Ciphers

Stream Ciphers

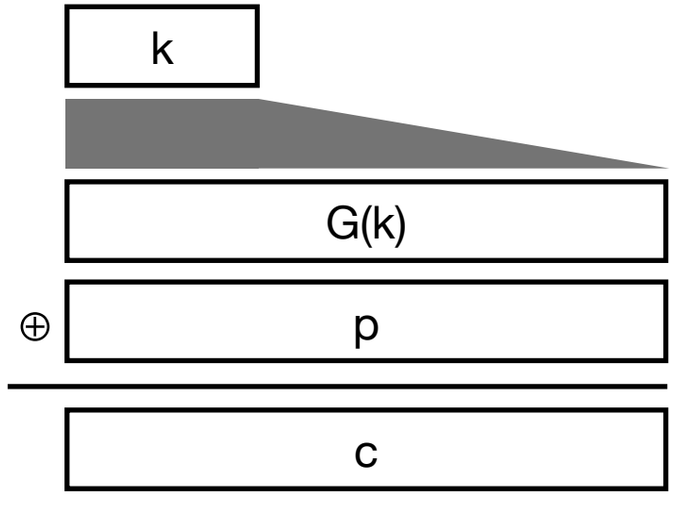
- *Idea*: make one-time pad practical by **extending the key securely**



- Deterministic pseudorandom number generator G(k)
  - takes a fixed length seed **k** (i.e., the key) and produces a sequence of bits
- **Decryption:** generate the same sequence and XOR to the ciphertext

Pseudorandom Number Generator (PRNG)

PRNG: takes fixed length seed, and generates a sequence of bits using a deterministic algorithm



- *Performance requirement*
  - PRNG must generate sequences that are as long as the plaintexts, so it must be **computationally efficient**
    - many algorithms are tailored for hardware architectures or implementations
- *Security requirement:* for an attacker who does not know the key, **bit sequence must be indistinguishable from true randomness**
- Statistical randomness tests (*e.g.,* NIST SP 800-22 “Statistical Test Suite”)
  - frequency of 0s and 1s must be roughly equal
  - distribution of the lengths of uninterrupted “runs” of 0s and 1s (*e.g.*, 1**0001** or 0**1111**0) must be roughly the same as for a true source of randomness
  - …

passing these tests does not guarantee security!

Bad Example: Linear Congruential Generator

- Generates a sequence of numbers X_1, X_2, ... from a seed number X_0

$$X_{n+1} = a \cdot X_n + b \bmod m$$

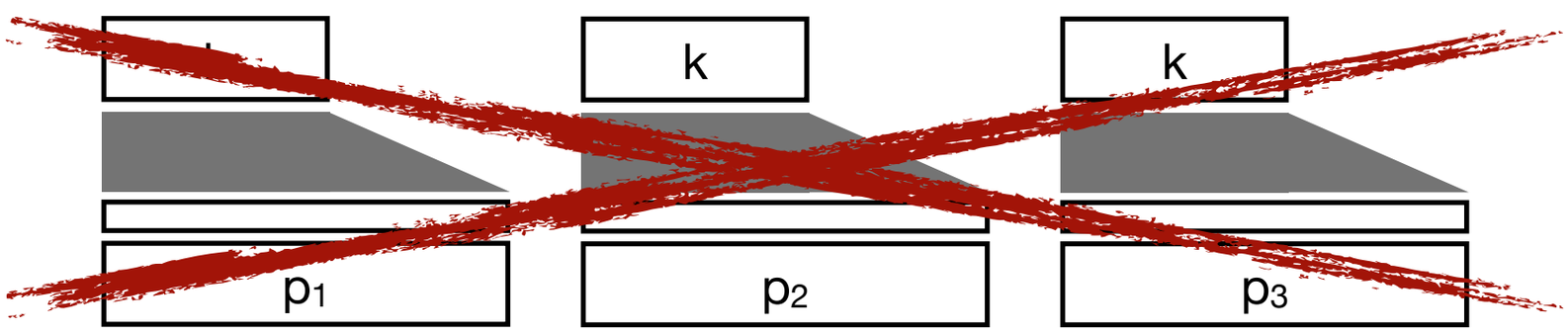
- Parameters **a**, **b**, and **m** must be carefully chosen
  - a = b = 1 → sequence 0, 1, 2, 3, ...
  - a = 7, b = 0, m = 32, and $X_0 = 7$ → sequence 7, 17, 23, 1, **7**, …
  - if m is prime and b = 0 (and a is properly chosen), then **sequence length is m - 1**
    - *example*: $X_{n+1} = 7^5 \cdot X_n \bmod (2^{31} - 1)$
- Passes many statistical tests, but provides **no security**
  - if the attacker knows the parameters, it can predict the sequence from one observed value
  - parameters can be efficiently computed from three observed values

Designing Secure Stream Ciphers

- Cryptanalytic attacks
  - attacker may know some bits of the plaintext → may know some bits of the sequence
  - *goal:* **bit sequence must be indistinguishable from true randomness**
  - security requirement
    1. **uniform distribution:**
       frequency of 0s and 1s is approximately equal in the generated sequence
    2. **independence:**
       no subsequence can be inferred from another, disjoint subsequence
- Brute-force attacks
  - **N**-bit long key can take $\mathbf{2^N}$ different values → attacker may try all of them
  - since 2014, the National Institute of Standard and Technology (NIST) recommends at least **112-bit** keys for encryption
  - as computers get faster over time, key sizes must be increased
- Key re-use: *can we use the same key to encrypt multiple plaintexts?*

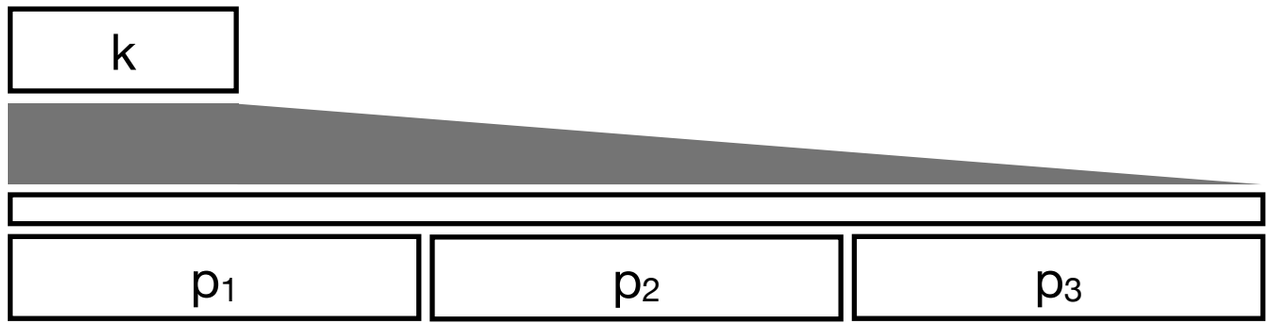
Key Reuse

- Naive approach ~ “many-time pad”



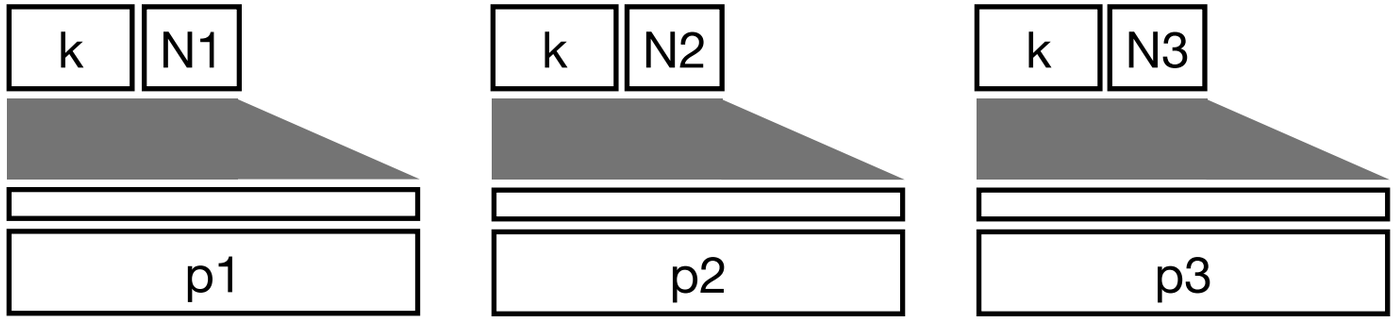
- attacker learns $p_1 \oplus p_2$, $p_2 \oplus p_3$, $p_1 \oplus p_3$, etc.
- if the attacker knows one plaintext, it can recover the others

- One continuous sequence for all plaintexts



- stream cipher must support seeking to any position in the sequence
- otherwise, full sequence must generated from beginning to decrypt p_3

- Nonces



- **nonce**: number used once
- mix (*e.g.*, prefix or XOR) key with each nonce → effectively different key for each plaintext

Next lecture:

Block Ciphers