CS 4173/5173
COMPUTER SECURITY

Security Metrics

OUTLINE LAST TIME

- Early ciphers aren't nearly strong enough
- Cryptography is a fundamental, and most carefully studied, component of security
  - Modern ciphers are based on computational difficulty.
- One-time pad is perfectly secure, but we have not proved it.

OUTLINE LAST TIME

- “Key” issues
  - Secret key cryptography
  - Public key cryptography
  - Hash

OUTLINE

- Security Metrics
  - Perfect Security
- P and NP problems

PERFECT SECURITY

PROBABILITY BASICS

- For events A and B
  - P(A): the probability that event A happens.
  - P(B): the probability that event B happens.
  - P(A,B): the probability that A and B happen at the same time.
  - P(A|B): the probability that A happens conditioned on the fact that B already happened.
    - Bayes Rule: P(A,B)= P(A|B) P(B) = P(B|A)P(A)
    - Q1: Given P(A|B) and P(A), which one is greater?
    - Q2: Does P(A|B)=1 mean
      - A=B?
      - or P(A)=P(B)?
      - Example: Suppose x is uniformly distributed in [0,1], A = $\{x<0.5\}$ and B=$\{x<0.1\}$

MORE QUESTIONS

- Q3: Given P(A|B, C) and P(A|B), which one is greater?
- Q4: if P(A,B)=1, then
  - P(A)=1?
  - P(B)=1?
- Q5: if P(A|B)=0, then
  - P(A,B)=0?
  - P(A)=0?
  - P(B)=0?

Bayes rule: $P(A,B) = P(A|B)\,P(B) = P(B|A)P(A)$

NOTATION

Notation	Meaning
$X \oplus Y$	Bit-wise exclusive-or (XOR) of X and Y
X | Y	Concatenation of X and Y e.g: 000 | 11 = 00011

XOR

- Two bits, x and y
- $x \oplus y$ is 0 if x and y are the same, and 1 otherwise.
  - e.g., $1 \oplus 0 = 1, 1 \oplus 1 = 0$
- $x \oplus y$ is also addition modulo 2.
  - i.e., $x \oplus y = (x + y) \bmod 2$
- Properties:
  - $(x \oplus y) \oplus z = x \oplus (y \oplus z)$
  - $x \oplus y = y \oplus x$
  - $x \oplus x = 0$

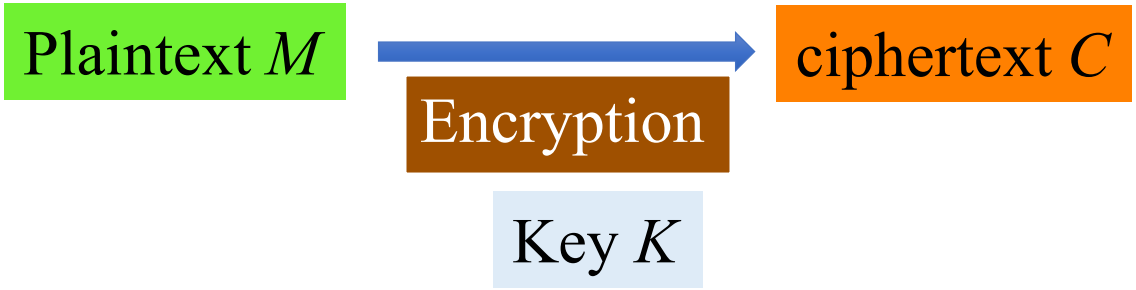
$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

BITS XOR

- Two sequences of bits, X and Y
- $X \oplus Y$ is based on bit-wise XOR.
  - e.g., $01 \oplus 10 = 11$, $100 \oplus 110 = 010$
- Properties:
  - $(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z)$
  - $X \oplus Y = Y \oplus X$
  - $X \oplus X =$ all 0 bits
  - $X \oplus$ all 0 bits $= X$
  - If $X \oplus Y = Z$, then $X \oplus Z = Y$ (used by one-time pad)

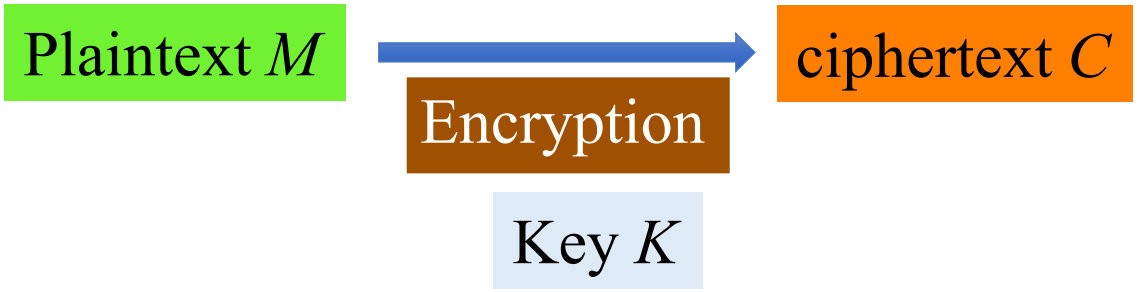
PERFECT SECURITY

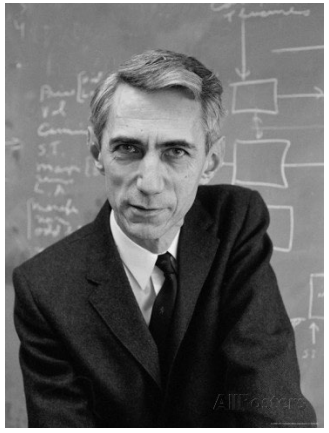
- How to define perfect security?
  - Q: Does it mean there is absolutely no way to know *M*??



SHANNON'S ELEGANT DEFINITION

- A perfectly secure system:
  - **for every *M*, *C*, P(*M*|*C*) = P(*M*)**





- If a system is perfectly secure, it also called information-theoretically secure, or achieving perfect secrecy.

UNDERSTANDING PERFECT SECURITY

- perfectly secure: **for *M*, *C*, P(*M*|*C*) = P(*M*)**
- Example: *M* is a uniformly distributed 3-bit plaintext, becomes a 3-bit ciphertext after encryption.

Value of *M*		Value of *C*
0 0 0		1 1 0
0 0 1		1 1 1
0 1 0	Need a rule to encrypt	0 0 0
0 1 1		0 0 1
1 0 0		0 1 0
1 0 1		0 1 1
1 1 0		1 0 0
1 1 1		1 0 1

  - We randomly guess *M* without looking at C, what is the probability the guess is right? P(M) = 1/8.
  - Now, look at *C*, what is the probability we can know M?
    - P(M|C) = 1/8?

UNDERSTANDING PERFECT SECURITY

- perfectly secure: **for *M*, *C*, P(*M*|*C*) = P(*M*)**
- Example: *M* is a uniformly distributed 3-bit plaintext, becomes a 3-bit ciphertext after encryption.

Value of *M*
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1

Encryption: only does some manipulation on the first 2 bits, and leaves the last bit unchanged.

Value of *C*
1 1 0
1 1 1
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1

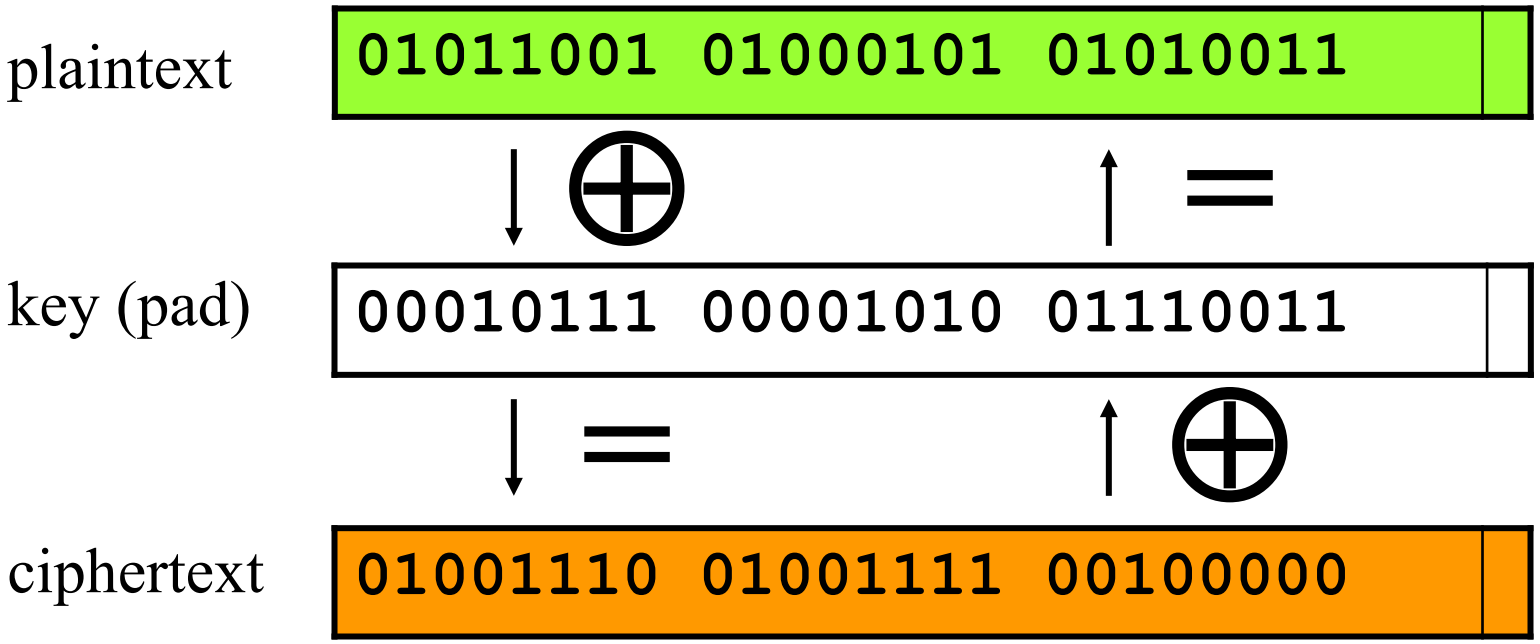
  - We randomly guess *M* without looking at C, what is the probability the guess is right? P(M) = 1/8.
  - Now, look at *C*, what is the probability we can know M?
    - P(M|C) = 1/4?

UNDERSTANDING PERFECT SECURITY

- perfectly secure: **for M, C, $P(M|C) = P(M)$**
  - Giving you the ciphertext does not increase the probability that the plaintext is revealed!
  - A perfectly secure system
    - does NOT mean there is no chance to get the plaintext!
    - makes sure that no attack strategy is better than a random guess!

ONE-TIME PAD

- One-time pad is perfectly secure



Decoding: If $X \oplus Y = Z$, then $X \oplus Z = Y$

PROVE ONE-TIME PAD IS PERFECTLY SECURE

- perfectly secure: **for *M*, *C*, P(*M*|*C*) = P(*M*)**
  - One time pad: $C = M \oplus K$
    - *K* is uniformly distributed at random.
      - Suppose the length is *n* bits, *for some k,* $P(K = k) = 2^{-n}$
  - Write in a formal way, given any message m and a cipher text c,
    - $P(C = c|M = m) = P(M \oplus K = c|M = m)$
      $= P(m \oplus K = c) = P(K = c \oplus m) = 2^{-n}$
    - $P(M = m|C = c) = \frac{P(M=m,C=c)}{P(C=c)} = \frac{P(C=c|M=m)P(M=m)}{\sum_n P(C=c|M=m_n)P(M=m_n)}$

      $= \frac{2^{-n}P(M = m)}{\sum_n 2^{-n}P(M = m_n)} = \frac{P(M = m)}{\sum_n P(M = m_n)} = P(M = m)$

      extended form of Bayes rule

Q: What are the limitations of one-time pad?

LIMITATION OF PERFECT SECURITY

- To achieve perfect security, the entropy of the key must be at least as large as the entropy of the plaintext.
  - Informally, this means that the key length must be no less than the plain text length.

  - What we know:
    - We really need a short key K for practical use.
    - Today's cryptographic systems do not achieve perfect security.

OTHER SECURITY METRICS?

- So, if every practical systems can not achieve perfect security. How to measure their security strength?
  - Gap to perfect security
  - Semantic security/ Indistinguishability
    - Widely adopted in today's cryptography research.
    - from computational complexity theory and asymptotic theory
      - Assume the attack has some **computational capability**, then under such an attack model, the performance is "close" to perfect security with "negligible" loss.

Informal Introduction to P and NP By Examples

GOALS

- We will introduce in an informal (more intuitive yet less rigorous) way the concepts of P and NP by using examples
  - Not to formally define P and NP
    - Avoid some rigorous definitions.
  - But to let you sense the meanings of P and NP, and how they are related to cryptography.

COMPUTATIONAL COMPLEXITY

- Array *A* with *n* elements:
  - {0, 21, 32, …., 1000, …, 0}

- Q1:
  - Find the maximum number in *A*.

```
max = A[0]
for i = 1 to n
        if A[i] > max, then max = A[i]
end for
```

How many times do you expect this line of code to run?

Very informally, we can consider the computational complexity to solve Q1 is *n*.

COMPUTATIONAL COMPLEXITY

- Array *A* with *n* elements:
  - {0, 21, 32, …., 1000, …, 0}

- The computational complexity in the following:
  - Q2: Find the minimum number in *A.*
  - Q3: Compute the sum of all elements in A.
  - Q4: Compute the average of all elements in A.
  - Q5: Sort the all elements in A.

SELECTION SORT

– Q5: Sort the all elements in A.

```
for i = 1 to n - 1
        pos = i;
        for j = i + 1 to n
                If A[j] < A[pos], then pos = j
        end for
        If pos is not i, then swap(A[i], A[pos])
end for
```

runs (n-1)n/2 times

SOLVING QUESTIONS AND P

- The dominant part in the computational complexity:
  - Q1: Find the maximum number in *A:* *n*
  - Q2: Find the minimum number in *A:* *n*
  - Q3: Compute the sum of all elements in A: *n*
  - Q4: Compute the average of all elements in A: n
  - Q5: selection-sort the all elements in A: *n(n-1)/2*

  - They are all polynomial functions!

P: the set of questions that can be solved in polynomial time

NP: (Nondeterministic Polynomial): the set of questions for which an answer can be verified in polynomial time

VERIFY AN ANSWER

- Array *A* with *n* elements:
  - {0, 21, 32, …., 1000, …, 0}

- Q1: Verify 1000 is the maximum in A

max = A[0]
for i = 1 to n
    if A[i] > max, then max = A[i]
end for
If max == 1000, then it is verified.

The computational complexity to verify is still *n*.

P AND NP

- Q1: Find the maximum number in *A:* *n*
  - *P? NP?*
- Q2: Find the minimum number in *A:* *n*
  - *P? NP?*
- Q3: Compute the sum of all elements in A: *n*
  - *P? NP?*
- Q4: Compute the average of all elements in A: n
  - *P? NP?*
- Q5: selection-sort the all elements in A: *n(n-1)/2*
  - *P? NP?*

QUESTIONS IN NP, BUT NOT IN P

- If a problem in P, then it is in NP.
  - Q1 – Q5 are in P and NP.

- Q6: find the right combination in a lock with *n* digits?

The complexity of verifying an answer is 1!!
What is the complexity of solving it?

- Q6: find the right combination in a lock with *n*-digit key?
  - Exhaustive search: 00…000 -> 11…111
  - Best case ?
    - 1
  - Worst case ?
    - 10^n
  - Average case ?
    - $10^n/2$ (uniformly distributed assumption)

  - In the average sense, the computational complexity of exhaustive search is exponential not polynomial!

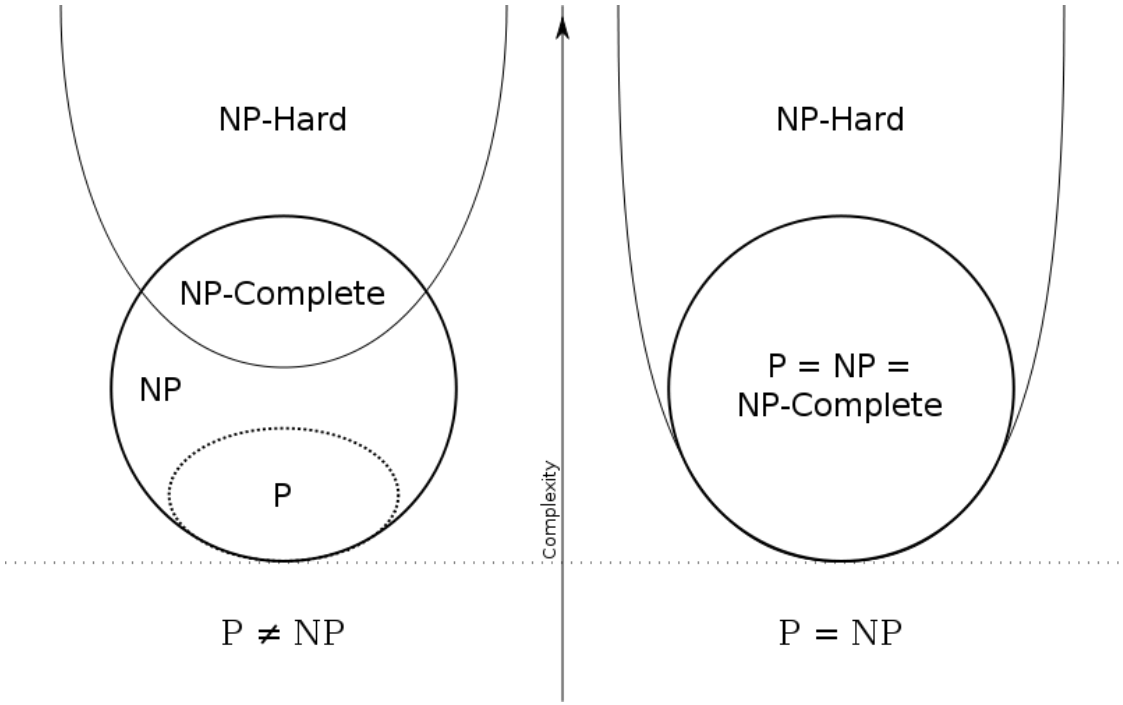
- Q6: find the right combination in a lock with *n* digits?
  - Exhaustive search: 00…000 -> 11…111
    - Average case: $10^n/2$
  - Can we say Q6 is not in P??
    - ?????

- Q6: find the right combination in a lock with *n*-digit key?
  - Verifying the solution is very easy
    - **So Q6 is clearly in NP**
  - Solving the problem is very hard
    - Exhaustive search: exponential not polynomial!
    - It unclear there is other polynomial solution!
    - **We are not sure whether Q6 is in P!**
    - **But we are sure that "currently" it is so "hard" to find a polynomial solution to solve Q6.**

MODERN CRYPTOGRAPHY

- All modern cryptographic methods
  - Aim to achieve semantic security, not perfect security.
  - Without the key, the adversary must solve an NP problem, where a solution in P with polynomial time is "currently believed very hard" to find.
    - The best solution currently is at least sub-exponential.

P AND NP (FROM WIKI)



P: the set of questions that can be solved in polynomial time

NP: the set of questions for which an answer can be verified in polynomial time

NP-Hard: the set of questions that are "at least as hard as" the hardest problems in NP

NP-Complete: the set of questions that are in both NP and NP-hard

IF P = NP

- If P = NP,
  - All modern cryptographic methods can be in danger. We may need much longer keys to ensure security.
  - We may have to use perfectly secure systems, like one-time pad.
  - Quantum computing.
  - … …