Instructor: David Wu (dwu4@virginia.edu)
TA: Rohit Musti (rm3gg@virginia.edu)

Overarching goal of cryptography: securing communication over untrusted networks

Alice ⟶ Bob
↓
third party should not be able to
1) eavesdrop of communication (confidentiality)
2) tamper with the communication (integrity)

Today: secure communication on web (https://...)
TLS protocol (transport layer security)
two components: handshake (key exchange)
record layer (confidentiality + integrity)
protecting data at rest: disk encryption

Most of this course: study mechanics for protecting confidentiality + data
- Encryption schemes for confidentiality
- Signature schemes for message integrity
- Key exchange for setting up shared secrets

End of this course: protecting communication ⟹ protecting computation
- Two users want to learn a joint function of their private inputs
  - ↳ training models on private (hidden) data
  - ↳ comparing two DNA sequences privately
  - ↳ private auction to determine winner without revealing bids
  - ↳ private voting mechanisms (can identify winner of election without revealing individual votes)
- We will show the following remarkable theorem:
  "Anything that can be computed with a trusted party can be computed without!"
- Will study concepts like zero-knowledge:
  - Can you prove to someone a theorem is true without telling them any information?
  - What does it mean to know something?

Logistics and administrivia:
- Course website: https://www.cs.virginia.edu/dwu4/courses/sp20
- See Piazza for announcements
- Homework submission via Gradescope + Collab (written assignments → Gradescope, some programming → Collab)
- Course consists of 5 homework assignments (worth 75%) and one take-home midterm (worth 25%)
  ↳ Collaboration encouraged but work must be independent (see website)
- Course TA: Rohit Musti
- Some lectures will be recorded (due to travel) — details will be provided later
  ↳ or held live via videoconference
- Three late days for the semester (see website for info)

$\swarrow$ $\mathcal{K}, \mathcal{M}, \mathcal{C}$ are sets (e.g., $\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0,1\}^{128}$)

<u>Definition</u>. A <u>cipher</u> is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ where $\mathcal{K}$ is a key-space, $\mathcal{M}$ is a message space and $\mathcal{C}$ is a ciphertext space, and consists of two algorithms (Encrypt, Decrypt):

$$\left.\begin{array}{l}\text{Encrypt} : \mathcal{K} \times \mathcal{M} \to \mathcal{C} \\ \text{Decrypt} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}\end{array}\right\}$$

functions should be "efficiently-computable"

theory: runs in probabilistic <u>polynomial</u> time [algorithm can be <u>randomized</u>]

practice: fast on an actual computer (e.g., < 10ms on my laptop)

<u>Correctness</u>: $\forall k \in \mathcal{K}, \forall m \in \mathcal{M}$:

$$\text{Decrypt}(k, \text{Encrypt}(k, m)) = m$$

"decrypting a ciphertext recovers the original message"

<u>A brief history of cryptography</u>:

Original goal was to protect communication (in times of war)

Basic idea: Alice and Bob have a shared key $k$

Alice computes $c \leftarrow \text{Encrypt}(k, m)$

ciphertext ($c$) key ($k$) message (plaintext) ($m$)

Bob computes $m \leftarrow \text{Decrypt}(k, c)$ to recover the message

This tuple (Encrypt, Decrypt) is called a <u>cipher</u>

<u>Early ciphers</u>:

- Caesar cipher: "shift by 3"

| |
|---|
| A $\mapsto$ D |
| B $\mapsto$ E |
| C $\mapsto$ F |
| $\vdots$ |
| X $\mapsto$ A |
| Y $\mapsto$ B |
| Z $\mapsto$ C |

Not a <u>cipher</u>! There is <u>no</u> key!

Anyone can decrypt!

$\hookrightarrow$ Algorithm to encrypt is assumed to be <u>public</u>.

<u>NEVER</u> RELY ON SECURITY BY OBSCURITY!

- Harder to change system than a key
- Less scrutiny for secret algorithms

- Caesar cipher++: "shift by $k$" ($k = 13$ : ROT-13)

  $k$ is the key

  $\hookrightarrow$ Still <u>totally broken</u> since there are only 26 possible keys (simply via <u>brute force guessing</u>)

- Substitution cipher: the key defines a permutation of the alphabet (i.e., substitution)

| |
|---|
| A $\mapsto$ C |
| B $\mapsto$ X |
| C $\mapsto$ J |
| $\vdots$ |
| Z $\mapsto$ T |

ABC $\mapsto$ CXJ

$\leftarrow$ substitution table is the key

How many keys? For English alphabet, $26! \approx 2^{88}$ possible keys

$\uparrow$ very large value, <u>cannot</u> brute force the key

Still broken by frequency analysis
- e is the most frequent character (~12%)
- q is the least frequent character (~0.10%)

Can also look at digram, trigram frequencies

- Vigener cipher (late 1500s) — "polyalphabetic substitution"
  key is short phrase (used to determine substitution table):

  m = HELLO

  k = CAT

  Encrypt(k, m):

  $$\begin{array}{r l} & \text{HELLO} \\ + & \underline{\text{CATCA}} \leftarrow \text{repeat the key} \\ & \text{KFFPP} \end{array}$$

  interpret letters as number between 1 and 26
  addition is modulo 26

  if we know the key length, can break using frequency analysis
  otherwise, can try all possible key lengths $\ell = 1, 2, \ldots$
  ↳ general assumption: keys will be much shorter than the message (otherwise if we have a good mechanism to deliver long keys securely, then can use that mechanism to share messages directly

- Fancier substitution ciphers: Enigma (based on rotor machines)
  but... still breakable by frequency analysis

<u>Today</u>: encryption done using computers, lots of different ciphers
- AES (advanced encryption standard, 2000) "block cipher"
- Salsa (2005) / ChaCha (2008) "stream cipher"

not ideal property...

One-time pad [Vigenère cipher where key is as long as the message!]

$\mathcal{K} = \{0,1\}^n$ Encrypt$(k, m)$: output $c = k \oplus m$

$\mathcal{M} = \{0,1\}^n$ Decrypt$(k, c)$: output $m = k \oplus c$

$\mathcal{C} = \{0,1\}^n$ bitwise exclusive OR operation (addition mod 2)

Correctness: Take any $k \in \{0,1\}^n$, $m \in \{0,1\}^n$:

$$\text{Decrypt}(k, \text{Encrypt}(k, m)) = k \oplus (k \oplus m) = (k \oplus k) \oplus m = m \quad (\text{since } k \oplus k = 0^n)$$

Is this secure? How do we define security?

- Given a ciphertext, cannot recover the key?

  NOT GOOD! Says nothing about hiding message. Encrypt$(k, m) = m$ would be secure under this definition, but this scheme is totally insecure intuitively!

- Given a ciphertext, cannot recover the message.

  NOT GOOD! Can leak part of the message. Encrypt$(k, (m_0, m_1)) = (m_0, m_1 \oplus k)$. This encryption might be considered secure but leaks half the message. [Imagine if message was "username: alice || password: 123456"

  this might be the string that is leaked!

- Given a ciphertext, cannot recover any bit of the message.

  NOT GOOD! Can still learn parity of the bits (or every pair of bits), etc. Information still leaked...

- Given a ciphertext, learn nothing about the message.

  GOOD! But how to define this?

Coming up with good definitions is difficult! Definitions have to rule out $\underline{\underline{\text{all}}}$ adversarial behavior (i.e., capture broad enough class of attacks)

↳ Big part of crypto is getting the definitions right. Pre-1970s: cryptography has relied on intuition, but intuition is often wrong! Just because I cannot break it does not mean someone else cannot...

How do we capture "learning nothing about the message"?

If the key is random, then ciphertext should not give information about the message.

Definition. A cipher (Encrypt, Decrypt) satisfies perfect secrecy if for all messages $m_0, m_1 \in \mathcal{M}$, and all ciphertexts $c \in \mathcal{C}$:

$$\underbrace{\Pr[k \xleftarrow{R} \mathcal{K} : \text{Encrypt}(k, m_0) = c]}_{\text{probability that encryption of } m_0 \text{ is } c\text{, where the probability is taken over the random choice of the key } k} = \Pr[k \xleftarrow{R} \mathcal{K} : \text{Encrypt}(k, m_1) = c]$$

Perfect secrecy says that given a ciphertext, any two messages are $\underline{\text{equally}}$ likely.

$\Rightarrow$ Cannot infer anything about underlying message given only the ciphertext (i.e., "ciphertext-only" attack)

Theorem. The one-time pad satisfies perfect secrecy.

Proof. Take any message $m \in \{0,1\}^n$ and ciphertext $c \in \{0,1\}^n$. Then,

$$\begin{aligned}\Pr[k \xleftarrow{R} \{0,1\}^n : \text{Encrypt}(k, m) = c] &= \Pr[k \xleftarrow{R} \{0,1\}^n : k \oplus m = c] \\ &= \Pr[k \xleftarrow{R} \{0,1\}^n : k = m \oplus c] \\ &= \frac{1}{2^n}\end{aligned}$$

This holds for all messages $m$ and ciphertexts $c$, so one-time pad satisfies perfect secrecy.

Are we done? We now have a perfectly-secure cipher!

No! Keys are very long! In fact, as long as the message... [if we can share keys of this length, can use same mechanism to share the message itself]

"One-time" restriction [will revisit this later]

Malleable [will revisit this later]

Issues with the one-time pad:

- **One-time**: Very important. Never reuse the one-time pad to encrypt two messages. Completely broken!

Suppose $c_1 = k \oplus m_1$ and $c_2 = k \oplus m_2$

Then, $c_1 \oplus c_2 = (k \oplus m_1) \oplus (k \oplus m_2)$

$= m_1 \oplus m_2$ ← learn the xor of two messages! ← can leverage this to recover messages (HW1)

One-time pad reuse:

- Project Verona (U.S. counter-intelligence operation against U.S.S.R during Cold War)
  ↳ Soviets reused some pages in codebook ~ led to decryption of ~3000 messages sent by Soviet intelligence over 37-year period [notably exposed espionage by Julius and Ethel Rosenberg]
- Microsoft Point-to-Point Tunneling (MS-PPTP) in Windows 98/NT (used for VPN)
  ↳ Same key (in stream cipher) used for both server → client communication AND for client → server communication
  ↳ (RC4)
- 802.11 WEP: both client and server use same key to encrypt traffic
  Many problems just beyond one-time pad reuse (can even recover key after observing small number of frames!)

- **Malleable**: one-time pad provides no integrity; anyone can modify the ciphertext:

$m \leftarrow k \oplus c$

↖ replace $c$ with $c \oplus m'$

$\Rightarrow k \oplus (c \oplus m') = m \oplus m'$ ← adversary's change now xored into original message