Perfect secrecy (formal)

- Encryption scheme \((\text{Gen, Enc, Dec})\) with message space \(\mathcal{M}\) and ciphertext space \(\mathcal{C}\) is *perfectly secret* if for every distribution over \(\mathcal{M}\), every \(m \in \mathcal{M}\), and every \(c \in \mathcal{C}\) with \(Pr[C = c] > 0\), it holds that

\[
Pr[M = m | C = c] = Pr[M = m]
\]

- i.e. the distribution of \(M\) does not change conditioned on observing the ciphertext
One-time pad

- Let $m = \{0, 1\}^n$
- $\textit{Gen}$: choose a uniform key $k \in \{0, 1\}^n$
- $\textit{Enc}_k(m) = k \oplus m$
- $\textit{Dec}_k(c) = k \oplus c$
- Correctness:

$$\textit{Dec}_k(\textit{Enc}_k(m)) = k \oplus (k \oplus m)$$

$$= (k \oplus k) \oplus m$$

$$= m$$
One-time pad
Perfect secrecy of one-time pad

- Note that *any* observed ciphertext can correspond to *any* message (why?)
  - (This is necessary, but not sufficient, for perfect secrecy)

- So, having observed a ciphertext, the attacker cannot conclude for certain which message was sent
Perfect secrecy of one-time pad for \( n \)-bit messages

Fix arbitrary distribution over \( \mathcal{M} = \{0, 1\}^n \), and arbitrary \( m, c \in \{0, 1\}^n \)

Want: \( \Pr[M = m | C = c] = \Pr[M = m] \)

\[
Pr[M = m | C = c] = Pr[C = c | M = m] \cdot \frac{Pr[M=m]}{Pr[C=c]}
\]
So need

1. \( Pr[C = c | M = m] = Pr[K = m \oplus c] = 2^{-n} \)
2. \( Pr[M = m] \)
3. \( Pr[C = c] = 2^{-n}. \)

Hence: \( Pr[M = m | C = c] = 2^{-n} \cdot \frac{Pr[M=m]}{2^{-n}} = Pr[M = m]. \)
One-time pad

- The one-time pad achieves perfect secrecy!

- One-time pad has historically been used in the real world
  - E.g. “red phone” between DC and Moscow

- I am not aware of anyone currently using it
  - Why isn’t the one-time pad used?
Can’t Use the Same Key Twice

- Say

\[ c_1 = k \oplus m_1 \]
\[ c_2 = k \oplus m_2 \]

- Attacker can compute

\[ c_1 \oplus c_2 = (k \oplus m_1) \oplus (k \oplus m_2) = m_1 \oplus m_2 \]

- This leaks information about \( m_1, m_2 \)!
Can't Use the Same Key Twice?

- $m_1 \oplus m_2$ is information about $m_1, m_2$

- Is this significant?
  - No longer perfectly secret!
  - $m_1 \oplus m_2$ reveals where $m_1, m_2$ differ
  - Frequency analysis
  - Exploiting characteristics of ASCII...
One-time pad

- Drawbacks
  - Key as long as the message
  - Only secure if each key is used to encrypt *once*
  - Trivially broken by a known-plaintext attack

Are there any other schemes that are perfectly secure?

Vote:

1. YES
2. NO
3. OTHER
One-time pad

- **Drawbacks**
  - Key as long as the message
  - Only secure if each key is used to encrypt *once*
  - Trivially broken by a known-plaintext attack

Are there any other schemes that are perfectly secure?

**Vote:**

1. YES
2. NO
3. OTHER

NO.
The other ones we saw, like 1-letter shift cipher, are 1-time pads.
Optimality of the one-time pad

- Theorem: if (Gen, Enc, Dec) with message space $\mathcal{M}$ is perfectly secret, then $|K| \geq |\mathcal{M}|$

- Intuition:
  - Given any ciphertext, try decrypting under every possible key in $\mathcal{K}$
  - This gives a list of up to $|\mathcal{K}|$ possible messages
  - If $|\mathcal{K}| < |\mathcal{M}|$, some message is not on the list
Optimality of the one-time pad

Theorem: if \((\text{Gen}, \text{Enc}, \text{Dec})\) with message space \(\mathcal{M}\) is perfectly secret, then \(|\mathcal{K}| \geq |\mathcal{M}|\)

Proof:

- Assume \(|\mathcal{K}| < |\mathcal{M}|\)
- Need to show that there is a distribution on \(\mathcal{M}\), a message \(m\), and a ciphertext \(c\) such that

\[
Pr[M = m | C = c] \neq Pr[M = m]
\]
Proof, continued

- Take the uniform distribution on $\mathcal{M}$
- Take any ciphertext $c$
- Consider the set $M(c) = \{\text{Dec}_k(c)\}_{k \in \mathcal{K}}$
  - These are the only possible messages that could yield the ciphertext $c$
- $|M(c)| \leq |\mathcal{K}| < |\mathcal{M}|$, so there is some $m$ that is not in $M(c)$
  - $\Pr[M = m|C = c] = 0 \neq \Pr[M = m]$
Defined perfect secrecy

One-time pad achieves it!

One-time pad is optimal!

Are we done...?
Perfect secrecy

- Requires that *absolutely no information* about the plaintext is leaked, even to eavesdroppers *with unlimited computational power*
  - Has some inherent drawbacks
  - Seems unnecessarily strong
A brief detour: randomness generation
Key generation

- When describing algorithms, we assume access to uniformly distributed bits/bytes

- Where do these actually come from?

- Random-number generation
Random-number generation

- Precise details depend on the system
  - Linux or unix: /dev/random or /dev/urandom
  - **Do not use rand() or java.util.Random**
    Not as random as the name would indicate!
  - Use crypto libraries instead
Random-number generation

- Two steps:
  1. Continually collect ‘unpredictable’ data
  2. Correct biases in it to make it more random.
Step 1

- Collect a “pool” of high-entropy data
- Must ultimately come from some physical process (since computation is deterministic)
  - External inputs
    - Keystroke/mouse movements
    - Delays between network events
    - Hard-disk access times
    - Other external sources
  - Hardware random-number generation (e.g. Intel)
Min-entropy

- I.e. “guessing entropy”

- The *min-entropy* of a random variable $X$ is defined as

$$H_\infty(X) = - \log_2(\max_x \{Pr[X = x]\})$$

(in bits)

- If $X$ ranges over $n$-bit strings, then $H_\infty(x) \leq n$
  - Equality iff $X$ has uniform distribution
Random-number generation

Request random bits

Processing/smoothing
Step 2: Smoothing

- Need to eliminate both *bias* and *dependencies*

- von Neumann technique for eliminating bias:
  - Collect two bits per output bit
    - 01 $\mapsto$ 0
    - 10 $\mapsto$ 1
    - 00, 11 $\mapsto$ skip
  - Note that this assumes *independence* (as well as constant bias)
Smoothing

- Can use *randomness extraction*

- Unkeyed extraction is possible for some input distributions; impossible for others

- Keyed extraction possible for all distributions
  - Extracted randomness is less than the input min-entropy
  - Where does the key come from?

- In practice, computational extraction is used
Key generation

- Read desired number of bytes from /dev/urandom
- See code
Encryption

- Plaintext = sequence of ASCII characters
- Key = sequence of hex digits, written in ASCII
- Read them; XOR them to get the ciphertext