Perfect Secrecy and the One-Time Pad

Based on *Modern Cryptography* Katz & Lindell Chapter 2

Lecture notes of Alexander Wood
awood@jjay.cuny.edu

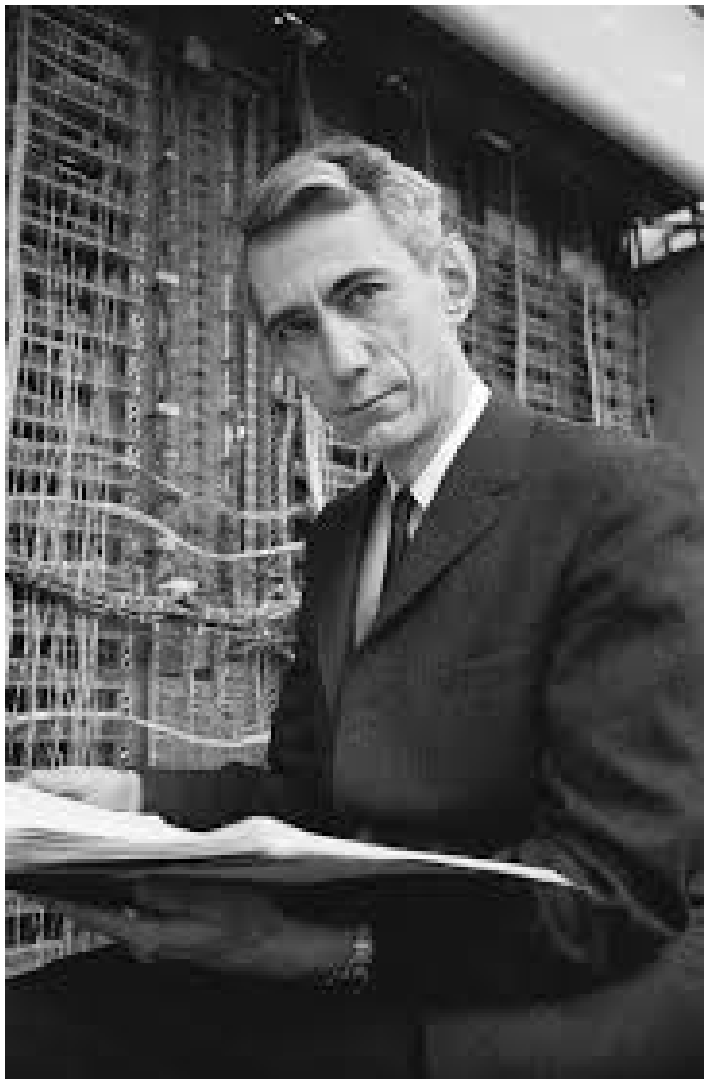
John Jay College of Criminal Justice

The One-Time Pad

Patented by Vernam in 1917.

Perfect Secrecy

In the 1940s Claude Shannon, the “father of the information age,” formulated a definition for **perfect secrecy**.



Perfect Secrecy, Informal

Informally, **perfect secrecy** means that the ciphertext reveals *no* knowledge about the underlying plaintext, and the adversary learns absolutely *nothing* about the plaintext which was encrypted.

Perfect Secrecy, Formal

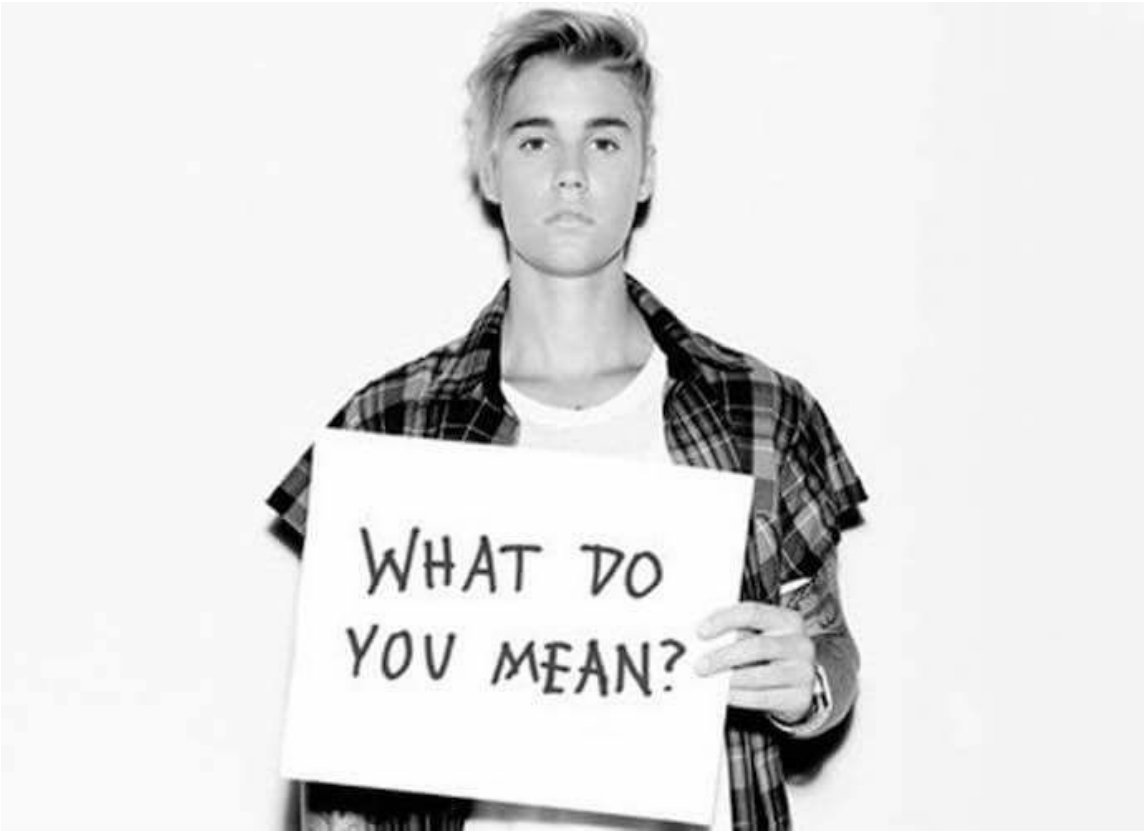
Now let's look at the formal definition of perfect secrecy provided in *Introduction to Modern Cryptography* by Katz & Lindell.

Definition

An encryption scheme $(\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$ *with message space* $\mathcal{M}$ *is* ***perfectly secret*** *if for every probability distribution over* $\mathcal{M}$*, every message* $m \in \mathcal{M}$*, and every ciphertext* $c \in \mathcal{C}$ *for which* $\Pr[C = c] > 0$*,*

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

Wait... what?



Let's unpack this definition a bit.

Message Space

The **message space** $\mathcal{M}$ is a formal way of discussing the set of all *possible* messages which could conceivably be sent.

Message Likelihood

When we say that the adversary knows the probability distribution over the message space $\mathcal{M}$, we mean that **the adversary knows the likelihood that different messages are being sent**.

What does the adversary know ?

In the perfect secrecy model, the adversary knows:

- The likelihood that different messages will be sent
- The encryption scheme

The only thing the adversary does *not* know is the private key(s).

Perfect Secrecy: The Game

As before, thinking of perfect secrecy as a game can help us understand it better. The game proceeds as follows:

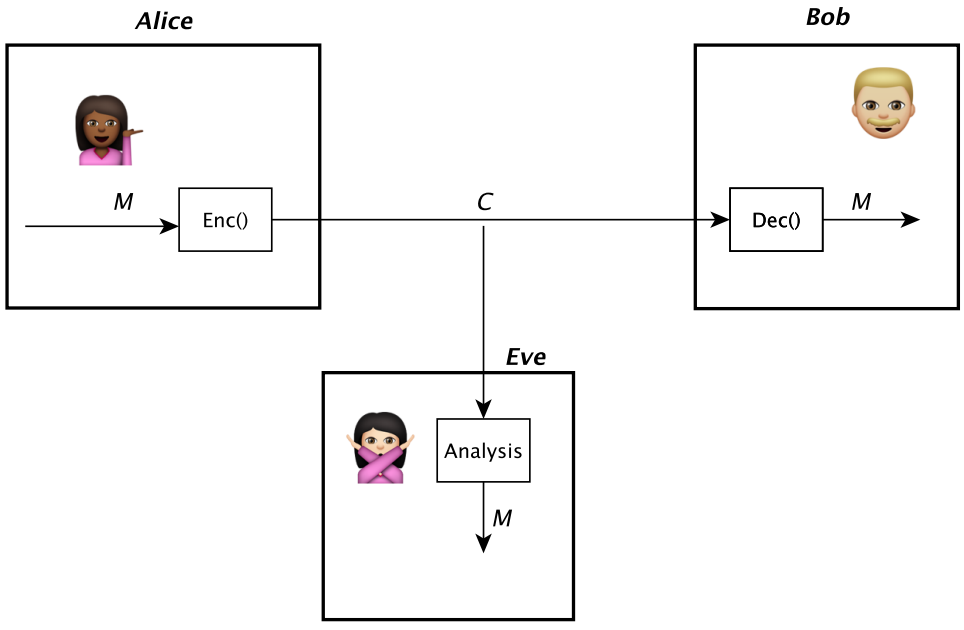
1. Alice chooses a message, encrypts it honestly, and sends it to Bob.
2. Eve eavesdrops on this message and receives the ciphertext C.
3. Eve analyses the ciphertext. If Eve knows nothing more after her analysis about the ciphertext than she knew before, we say the scheme is perfectly secret.

Perfect Secrecy: Attack Model

Which attack model does this fall under?

Perfect Secrecy: Attack Model

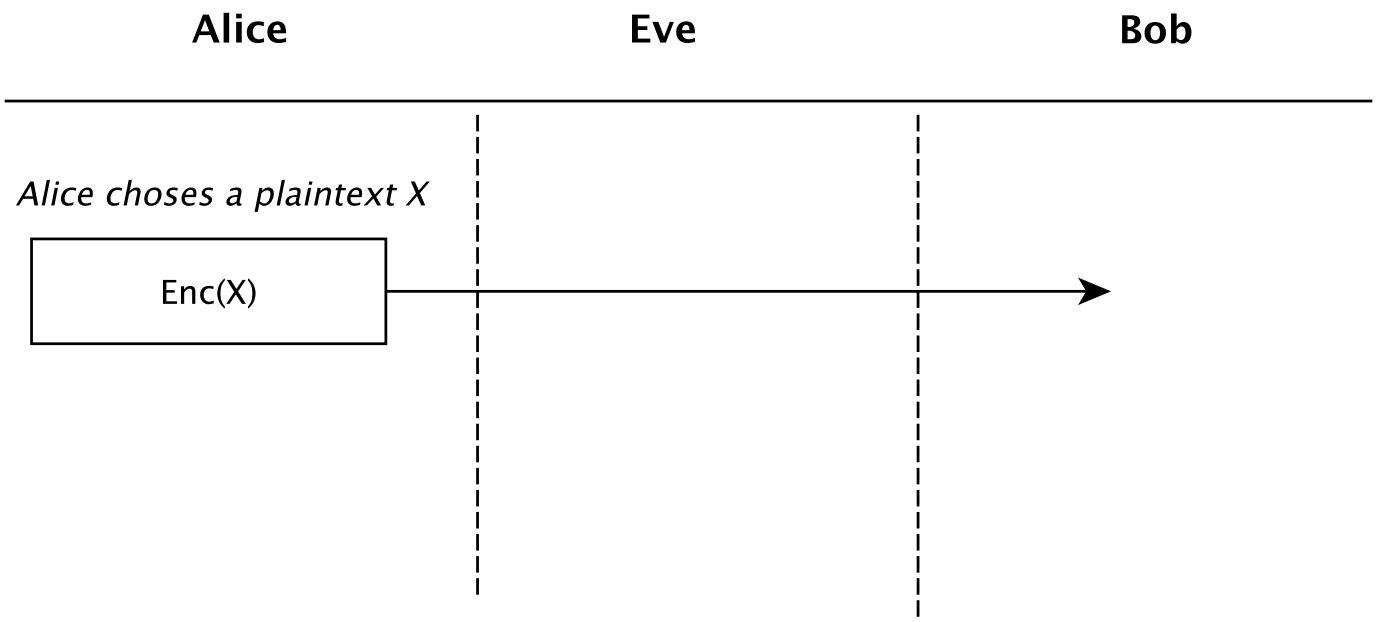
Which attack model does this fall under?



It is a ciphertext-only attack, since Eve only has access to the ciphertexts which she can eavesdrop on.

Attack Model

Here's another way of visualising the attack model.



Perfect Secrecy

When can we truly know that it is impossible for Eve to have learned **any information at all** about the message?

Probability Distributions On Plaintext vs. Ciphertext

For every two messages $m, m' \in \mathcal{M}$ and every ciphertext $c \in \mathcal{C}$,

$$\Pr[Enc_K(m) = c] = \Pr[Enc_K(m') = c].$$

This means that **it is impossible to distinguish an encryption of any message from an encryption of any other message.**

Non-example: Vigenère

Consider the the message space $\mathcal{M}$ which consists of all two-character strings, such as `AB`, `YY`, etc. Assume the keyspace consists of all one- or two-letter keys.

Non-example: Vigenère

Observe that

$$\Pr[m = aa] = \frac{1}{26} \cdot \frac{1}{26}$$

However,

$$\Pr[m = aa | c = xy] = \frac{1}{2} \cdot \frac{1}{26} \cdot \frac{1}{26}$$

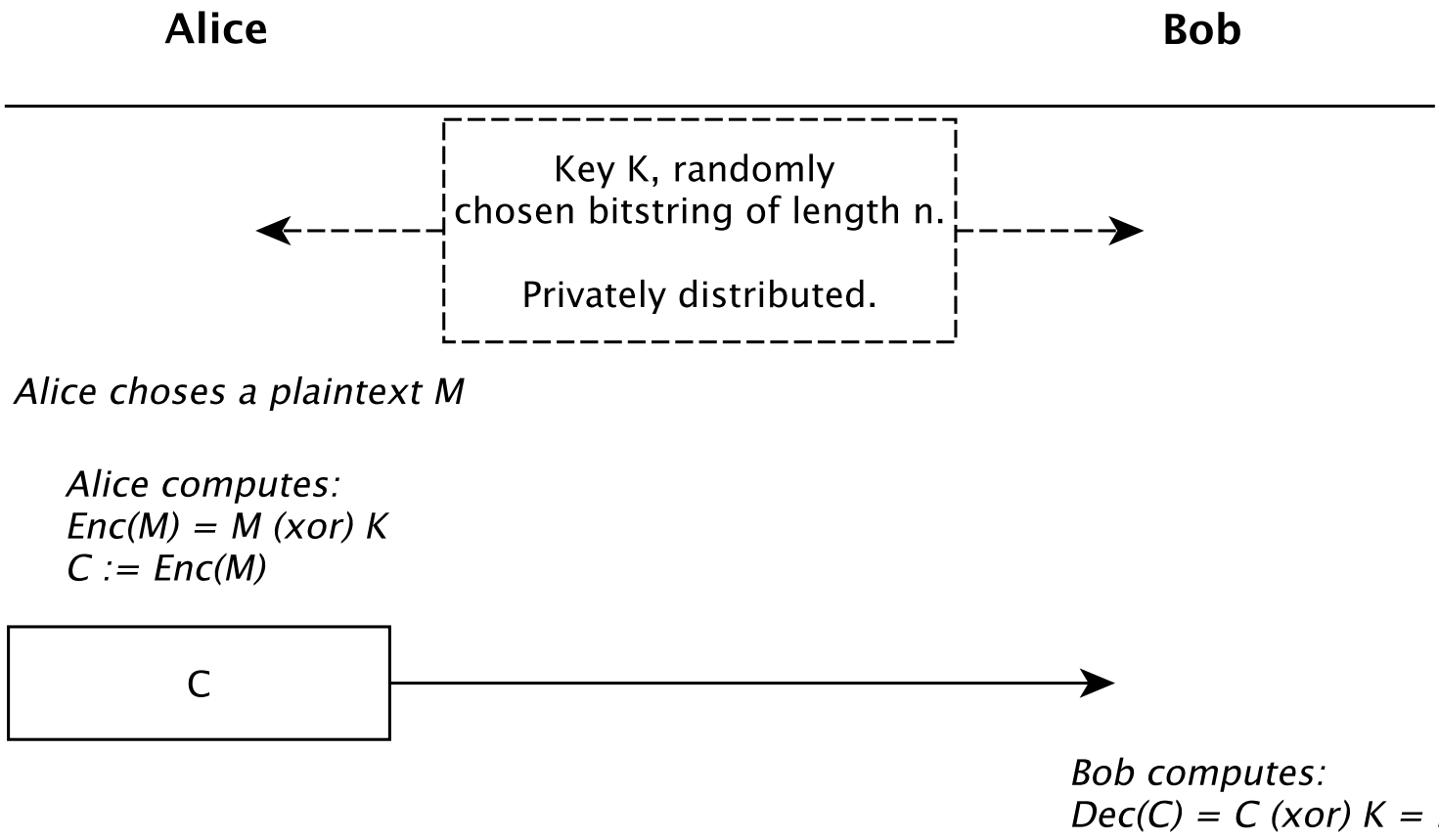
The One-Time Pad Cryptosystem

- `KeyGen`: The key generation algorithm randomly generates a key K as a binary string of length n.
- `Enc`: Represent the message m as a binary string of length n. Encrypt via the bitwise XOR operation,

$$c = K \oplus m.$$

- `Dec` Decrypt by computing $m = K \oplus c$.

The One-Time Pad Cryptosystem



Representing Characters As Bits

Each ASCII character can be converted to a bit string of length 8 (one byte). For instance,

$$\texttt{'A'} = 65$$

and

$$65 = [01000001]_2$$

Characters to Bits and Back in Python

Characters to bits:

```
>>> ord('A')
65
>>> bin(65)
'0b1000001'
>>> bin(ord('A'))
'0b1000001'
```

Bits to characters:

```
>>> int(0b1000001)
65
>>> chr(65)
'A'
>>> chr(int(0b1000001))
'A'
```

or specify base with string input:

```
>>> int('0b1000001', 2)
65
```

Coding Exercise 1: Characters To Bitstrings

```
INPUT: A string of characters.
OUTPUT: A bitstring, where each 8 bits
        corresponds to the letters in the string.

Example:
INPUT: 'A'
OUTPUT: '01000001'

INPUT: 'AA'
OUTPUT: '0100000101000001'

INPUT: 'AB'
OUTPUT: '0100000101000010'
```

Coding Exercise 2: Bits To Characters

```
INPUT: A bit string.
OUTPUT: A string of the characters
        encoded by the bit string.

Example:
INPUT: '01000001'
OUTPUT: 'A'

INPUT: '0100000101000010'
OUTPUT: 'AB'
```

Encrypting with XOR

A message `01000001` would be encoded with the One-Time Pad using secret key `10110101` using `XOR` (exclusive-or) as follows:

```
Message:     01000001
Key:         10110101

XOR:         11110100
```

OTP And Perfect Secrecy

We want to convince ourselves that encryption using the One-Time Pad does *nothing* to give away *any* information about the plaintext message.

OTP And Perfect Secrecy

Consider trying a brute-force attack on the OTP system. If the message is b bits long, there are 2^b possible keys. However, there are also 2^b possible messages.

Any ciphertext message of length b could possibly decrypt to *any* plaintext message of length $b/8$ (post-conversion from bytes to chars).

The OTP gives us nothing else to work with! The encryption is no more likely to be one message than literally any other message of that length.

Why Use XOR?

Why use `XOR`, instead of `AND` or `OR`?

	0	1
0	0	1
1	1	0

`XOR`

	0	1
0	0	1
1	1	1

`OR`

	0	1
0	0	0
1	0	1

`AND`

The `XOR` gate is the only one which is equally likely to have `0` or `1` as an output.

Why Use XOR?

Let's look at the illustrations on Khan Academy at the following link to visualize this.

`https://www.khanacademy.org/computing/computer-science/cryptography/ciphers/a/xor-and-the-one-time-pad`

Why a ONE-time Pad?

Assume Alice has two messages, m_1 and m_2, and a key k. She sends

$$c_1 = m_1 \oplus k \text{ and } c_2 = m_2 \oplus k$$

to Bob. Now, an eavesdropper Eve can compute

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

Eve can't necessarily find m_1 or m_2, but she can find the `XOR` of the plaintexts. Thus she has reduced her work from finding two messages m_1 and m_2 to finding only one message, since she has found a dependence between them.

Successful hacks of the "many-time" pad have been carried out by cryptographers.

OTP: Practical?

If the One-Time Pad is perfectly secure, why is not used for everything?!

OTP: Practical?

If the One-Time Pad is perfectly secure, why is not used for everything?!

The problem is key distribution. Once the key is distributed, the one-time pad can satisfy perfect secrecy. In real-life applications, the One-Time Pad is only as secure as the key distribution method.

Coding Exercise 3: Key Generation Algorithm

```
INPUT: Integer n, the desired length of the key.
OUTPUT: The key, a bit string of length n.

Hint: use random.randint(0,1)

Example:
INPUT: 8
OUTPUT: 10101111

INPUT: 8
OUTPUT: 01101001

INPUT: 16
OUTPUT: 0010111010010l0
```

References

- *Modern Cryptography* by Katz & Lindell Chapter 2
- emphHacking Secret Ciphers With Python Chapter 22
- Khan Academy: `https://www.khanacademy.org/computing/computer-science/cryptography/crypt/v/one-time-pad`