

# **CSE 484 / CSE M 584: Cryptography: Randomness and Symmetric Crypto**

Fall 2023

Franziska (Franzi) Roesner  
franzi@cs

UW Instruction Team: David Kohlbrenner, Yoshi Kohno, Franziska Roesner. Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

# Announcements

- Lab 1 due tonight
  - Remember to turn things in to both:
    - **Canvas** (your hashes from turnin.sh: as a group)
    - **Gradescope** (your writeups: **individual, in your own words!**)
  - Good idea to backup your sploit files (e.g., via scp to your local machine)

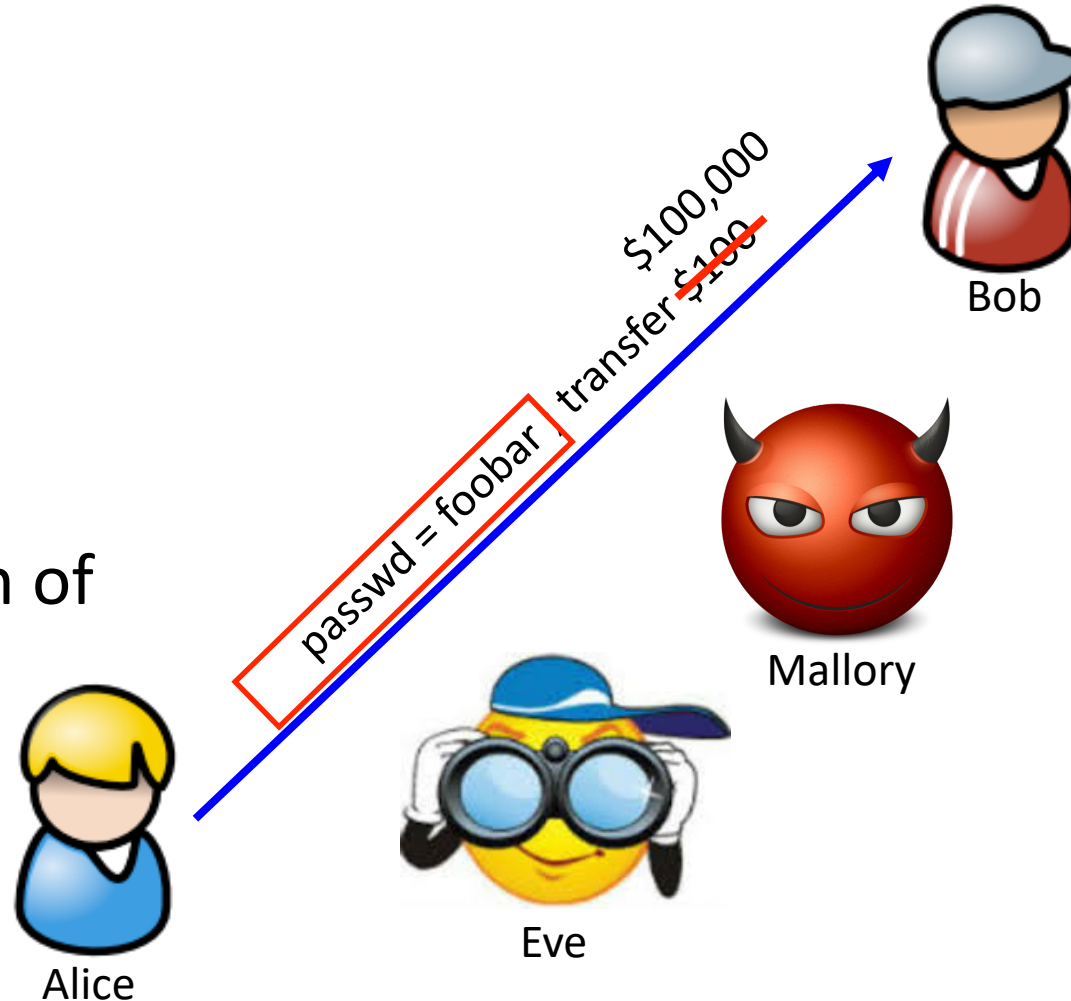
# Common Communication Security Goals

## Privacy of data:

Prevent exposure of information

## Integrity of data:

Prevent modification of information



# How Cryptosystems Work Today

- **Layered approach:** Cryptographic protocols (like “CBC mode encryption”) built on top of cryptographic primitives (like “block ciphers”)
- **Flavors of cryptography:** Symmetric (private key) and asymmetric (public key)
- Public algorithms (Kerckhoff’s Principle – next slide)
- Security proofs based on assumptions (*not this course*)
- **Be careful about inventing your own!**  
(If you just want to use some crypto in your system, use vetted libraries!)

# Kerckhoff's Principle

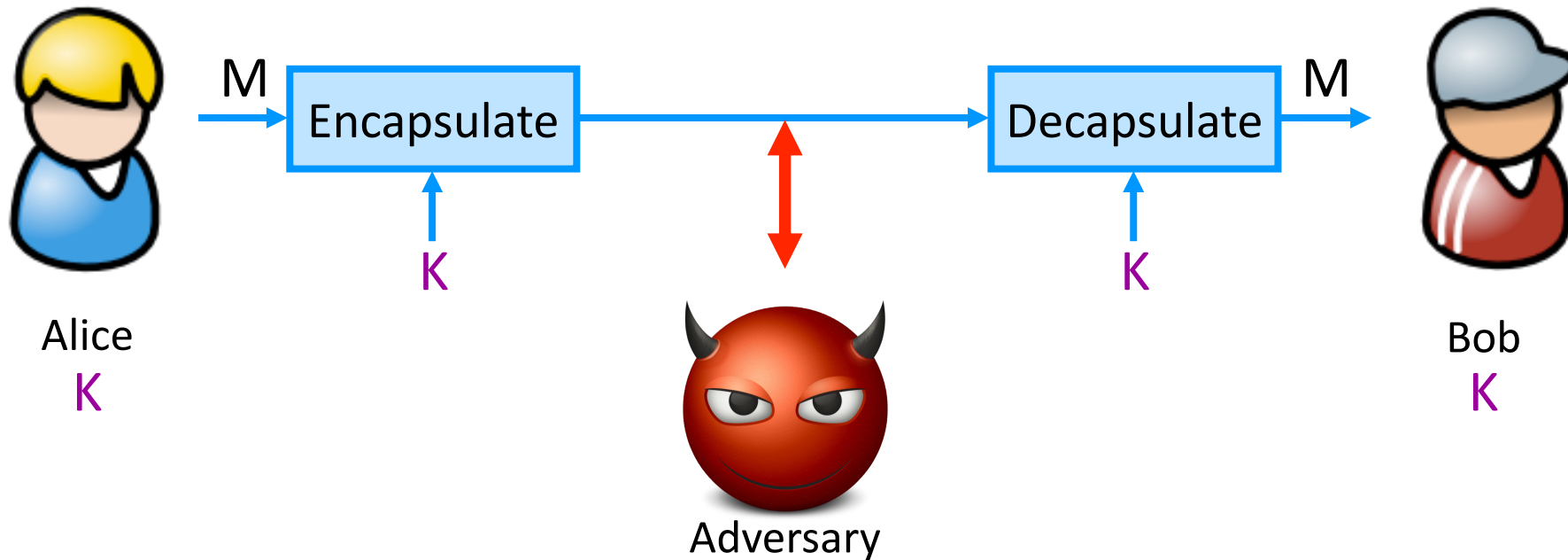
- Security of a cryptographic object should depend only on the secrecy of the secret (private) key.
- Security should not depend on the secrecy of the algorithm itself.
- (Foreshadowing: Need for randomness – the key to keep private)

# Flavors of Cryptography

- Symmetric cryptography
  - Both communicating parties have access to a **shared random string  $K$** , called the **key**.
- Asymmetric cryptography
  - Each party creates a public key  **$pk$**  and a secret key  **$sk$** .
  - *Hard concept to fully understand, and revolutionary! Inventors won the Turing Award 😊*

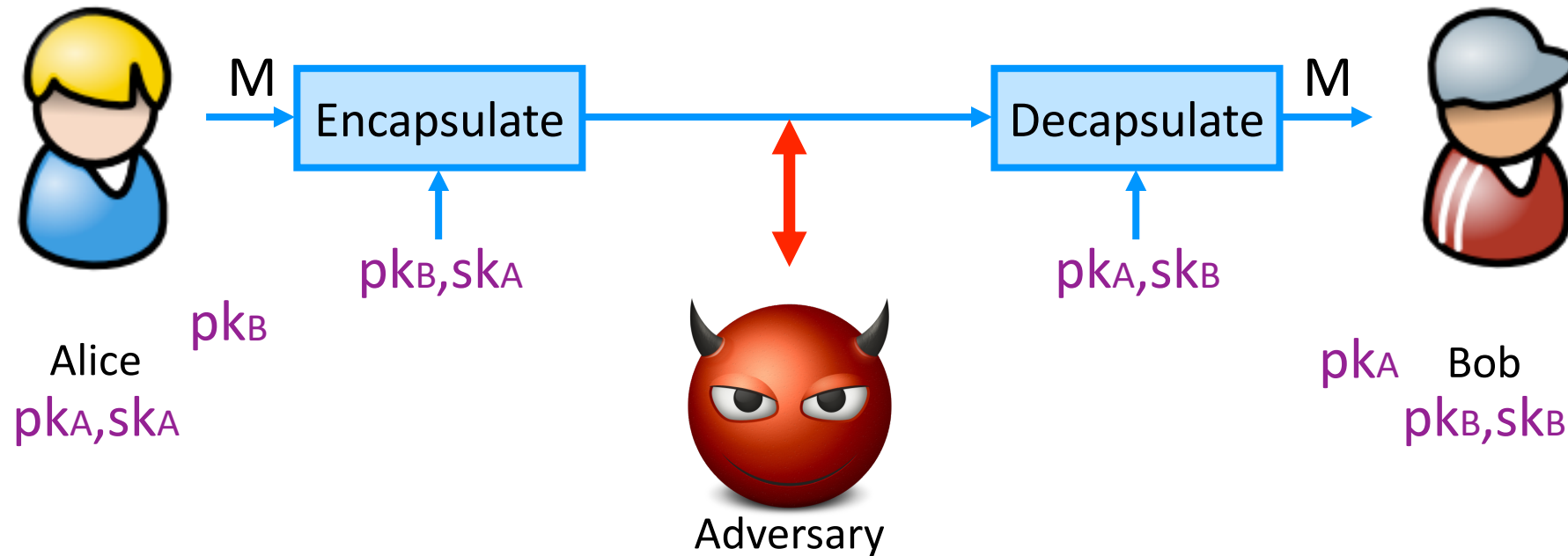
# Symmetric Setting

Both communicating parties have access to a **shared random string  $K$** , called the **key**.



# Asymmetric Setting

Each party creates a public key  $pk$  and a secret key  $sk$ .





# Flavors of Cryptography

- Symmetric cryptography
  - Both communicating parties have access to a **shared random string  $K$** , called the **key**.
- Asymmetric cryptography
  - Each party creates a public key  **$pk$**  and a secret key  **$sk$** .

# Flavors of Cryptography

- Symmetric cryptography
  - Both communicating parties have access to a **shared random string  $K$** , called the **key**.
  - **Challenge: How do you privately share a key?**
- Asymmetric cryptography
  - Each party creates a public key  **$pk$**  and a secret key  **$sk$** .
  - **Challenge: How do you validate a public key?**
- **Key building block: Randomness** – something that the adversaries don't know, can't predict, can't figure out

# Ingredient: Randomness

- Many applications (especially security ones) require randomness
- Explicit uses:
  - Generate secret cryptographic keys
  - Generate random initialization vectors for encryption
- Other “non-obvious” uses:
  - Generate passwords for new users
  - Shuffle the order of votes (in an electronic voting machine)
  - Shuffle cards (for an online gambling site)

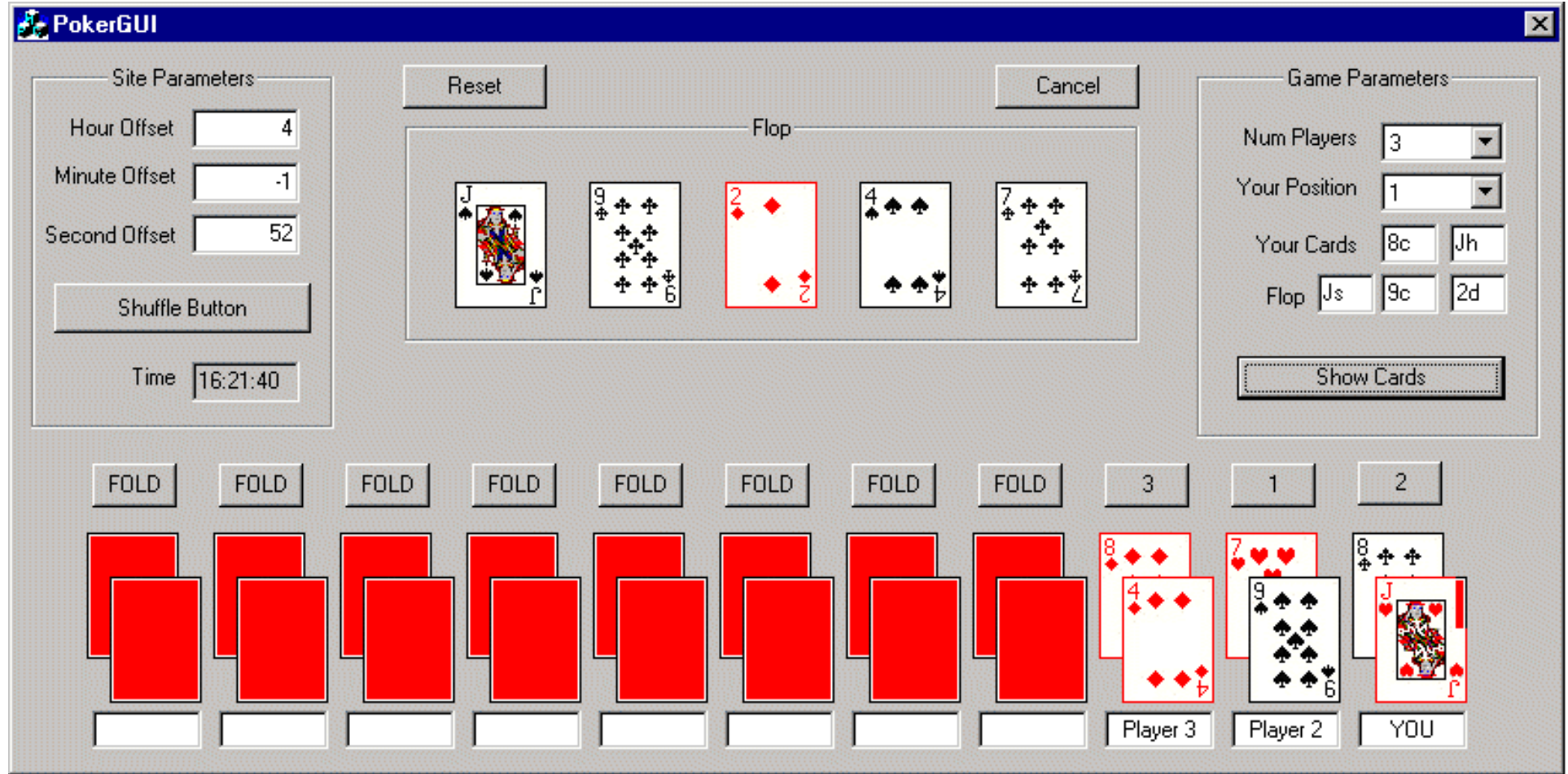
# C's rand() Function

- C has a built-in random function: `rand()`

```
unsigned long int next = 1;
/* rand:  return pseudo-random integer on 0..32767 */
int rand(void) {
    next = next * 1103515245 + 12345;
    return (unsigned int)(next/65536) % 32768;
}
/* srand:  set seed for rand() */
void srand(unsigned int seed) {
    next = seed;
}
```

- Problem: don't use `rand()` for security-critical applications!
  - Given a few sample outputs, you can predict subsequent ones





More details: “How We Learned to Cheat at Online Poker: A Study in Software Security”

[http://www.cigital.com/papers/download/developer\\_gambling.php](http://www.cigital.com/papers/download/developer_gambling.php)

# PS3 and Randomness

Hackers obtain PS3 private cryptography key due to epic programming fail? (update)

<http://www.engadget.com/2010/12/29/hackers-obtain-ps3-private-cryptography-key-due-to-epic-programm/>

- 2010/2011: Hackers **found/released private root key** for Sony's PS3
- Key used to sign software – **now can load any software on PS3** and it will execute as “trusted”
- Due to bad random number: **same “random” value used to sign all system updates**

**How might we get “good” random numbers?**



# Obtaining Pseudorandom Numbers

- For security applications, want “cryptographically secure pseudorandom numbers”
- Libraries include cryptographically secure pseudorandom number generators (CSPRNG)

# Obtaining Pseudorandom Numbers

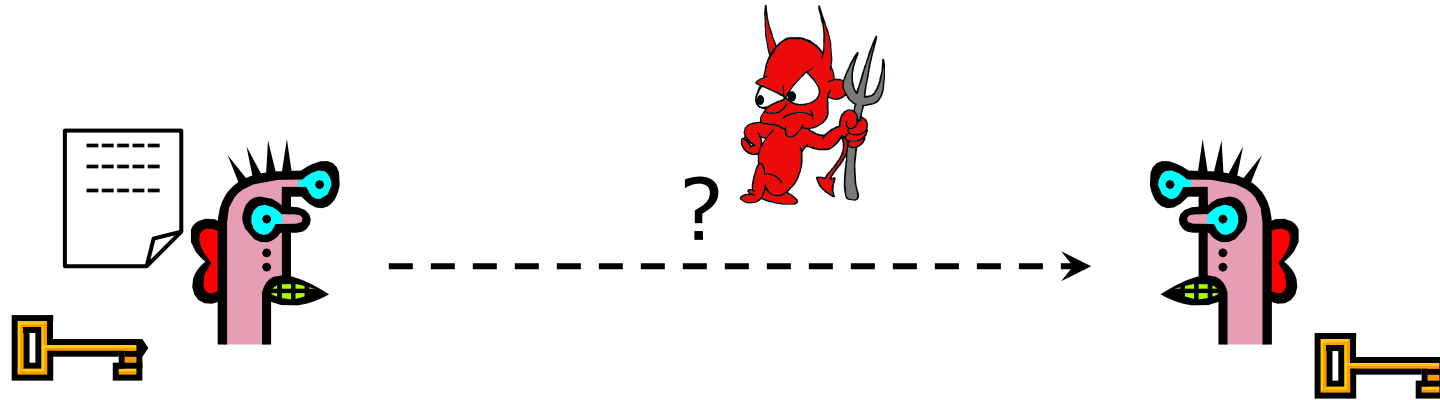
- Linux:
  - /dev/random – blocking (waits for enough entropy)
  - /dev/urandom – nonblocking, possibly less entropy
  - getrandom() – syscall! – by default, blocking
- Internally:
  - Entropy pool gathered from multiple sources
    - e.g., mouse/keyboard/network timings
- Challenges with embedded systems, saved VMs

# Obtaining *Random Numbers*

- Better idea:
  - AMD/Intel's *on-chip random number generator*
    - RDRAND
- Hopefully no hardware bugs!

# Now: Symmetric Encryption

# Confidentiality: Basic Problem

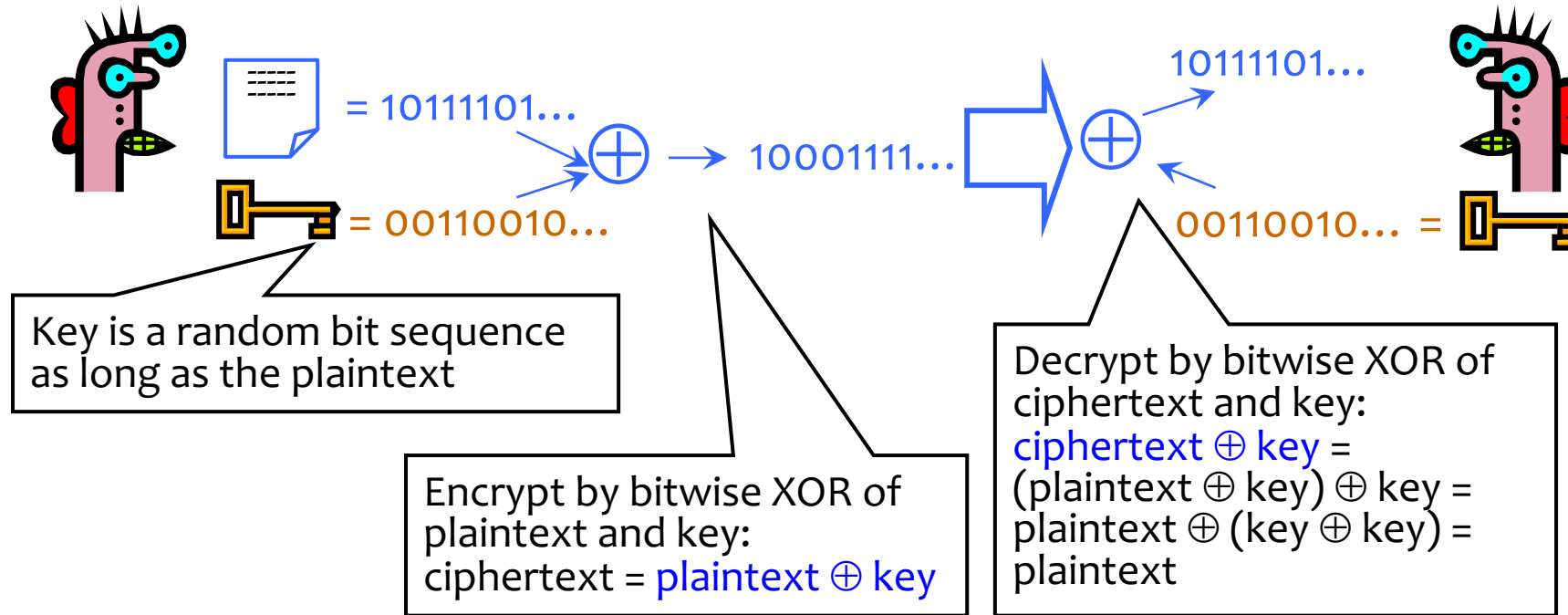


Given (Symmetric Crypto): both parties know the same **secret**.

Goal: send a message confidentially.

Ignore for now: How is this achieved in practice??

# One-Time Pad



Cipher achieves **perfect secrecy** if and only if there are **as many possible keys as possible plaintexts**, and **every key is equally likely** (Claude Shannon, 1949)

# Advantages of One-Time Pad

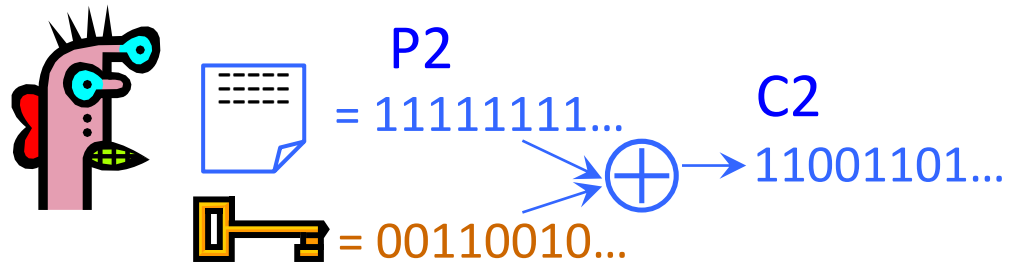
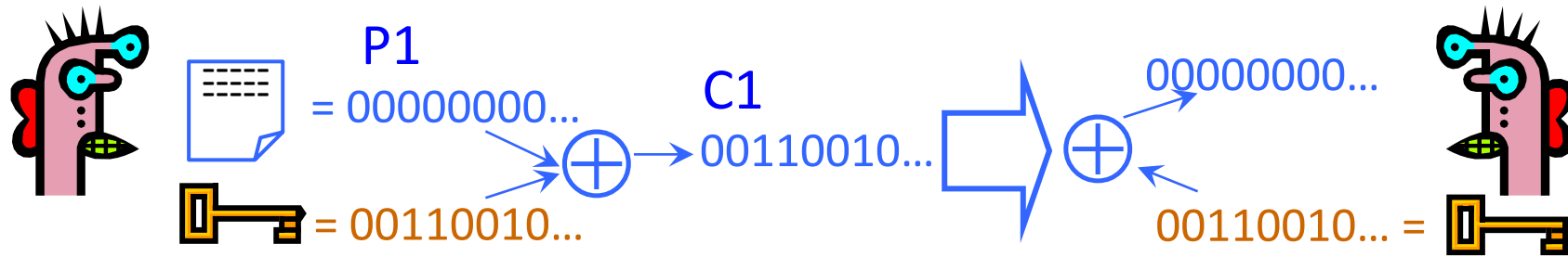
- Easy to compute
  - Encryption and decryption are the same operation
  - Bitwise XOR is very cheap to compute
- As secure as theoretically possible
  - Given a ciphertext, all plaintexts are equally likely, regardless of attacker's computational resources
  - ... as long as the key sequence is truly random
    - True randomness is expensive to obtain in large quantities
  - ... as long as each key is same length as plaintext
    - But how does sender communicate the key to receiver?

# Problems with the One-Time Pad?

- What potential security problems do you see with the one-time pad?
- Canvas -> Quizzes -> Oct 13



# Dangers of Reuse



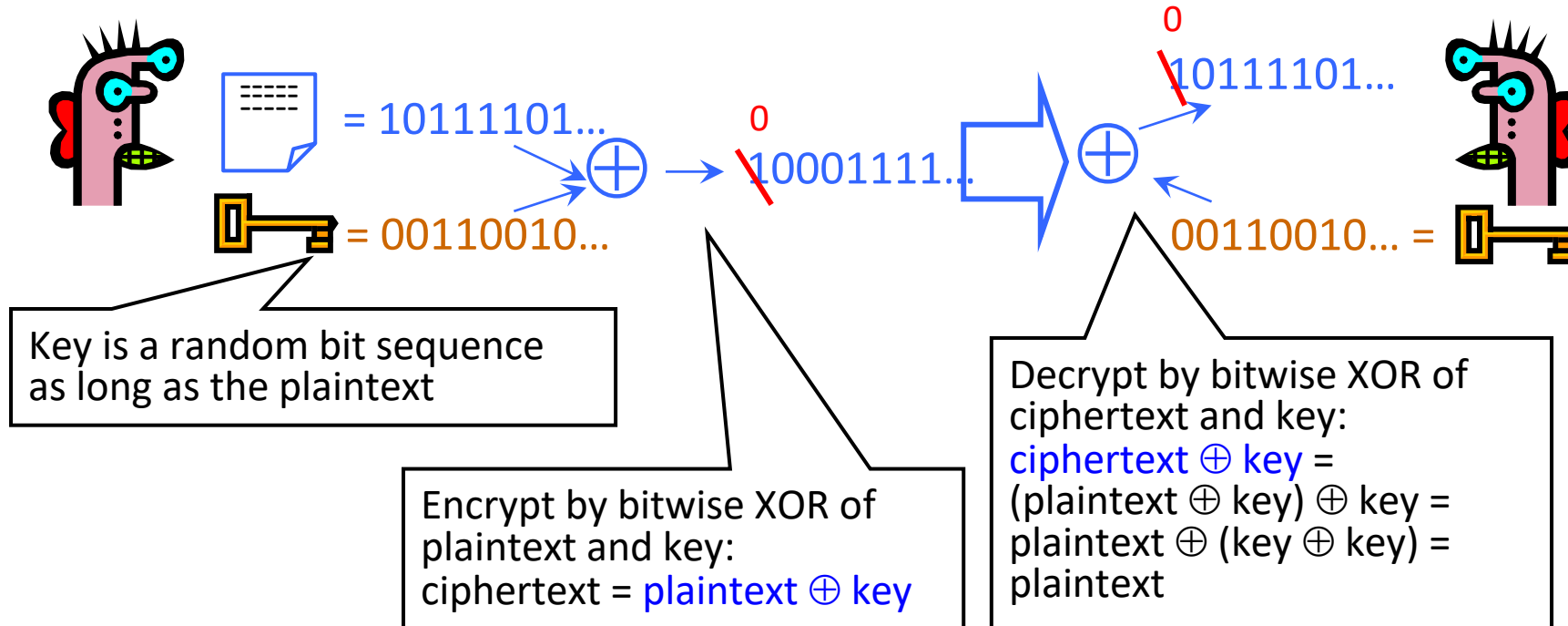
Learn relationship between plaintexts

$$\begin{aligned} C1 \oplus C2 &= (P1 \oplus K) \oplus (P2 \oplus K) = \\ &= (P1 \oplus P2) \oplus (K \oplus K) = P1 \oplus P2 \end{aligned}$$

# Problems with One-Time Pad

- (1) Key must be as long as the plaintext
  - Impractical in most realistic scenarios
  - Still used for diplomatic and intelligence traffic
- (2) **Insecure if keys are reused**
  - **Attacker can obtain XOR of plaintexts**

# Integrity?



# Problems with One-Time Pad

- (1) Key must be as long as the plaintext
  - Impractical in most realistic scenarios
  - Still used for diplomatic and intelligence traffic
- (2) Insecure if keys are reused
  - Attacker can obtain XOR of plaintexts
- (3) **Does not guarantee integrity**
  - One-time pad only guarantees confidentiality
  - Attacker cannot recover plaintext, but can easily change it to something else

# Reducing Key Size

- What to do when it is infeasible to pre-share huge random keys?
  - When one-time pad is unrealistic...
- Use special cryptographic primitives: block ciphers, stream ciphers
  - Single key can be re-used (with some restrictions)
  - Not as theoretically secure as one-time pad