# CSE 484 / CSE M 584: Software Security & Cryptography

Winter 2022

Tadayoshi (Yoshi) Kohno
yoshi@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

- Still working out hybrid lesson structure
- Wednesday also hybrid
- Friday entirely online (Emily McReynolds guest lecture)

# General Principles

- Check inputs
- Check all return values
- Least privilege
- Securely clear memory (passwords, keys, etc.)
- Failsafe defaults
- Defense in depth
  - Also: prevent, detect, respond

# General Principles

- Reduce size of trusted computing base (TCB)
- Simplicity, modularity
  - But: Be careful at interface boundaries!
- Minimize attack surface
- Use vetted components
- Security by design
  - But: tension between security and other goals
- Open design? Open source? Closed source?
  - Different perspectives

# Does Open Source Help?

- Different perspectives…
- Positive example?
  - Linux kernel backdoor attempt thwarted (2003)
    (http://www.freedom-to-tinker.com/?p=472)
- Negative example?
  - Heartbleed (2014)
    - Vulnerability in OpenSSL that allowed attackers to read arbitrary memory from vulnerable servers (including private keys)
  - Log4j (2021)

# Vulnerability Analysis and Disclosure

- What do you do if you've found a security problem in a real system?
- Say
  - A commercial website?
  - UW grade database?
  - Boeing 787?
  - TSA procedures?

**Breakout Groups:**
**What would you do? What ethical questions come up?**

# Vulnerability Analysis and Disclosure

- Suppose companies A, B, and C all have a vulnerability, but have not made the existence of that vulnerability public
- Company A has a software update prepared and ready to go that, once shipped, will fix the vulnerability; but B and C are still working on developing a patch for the vulnerability
- Company A learns that attackers are exploiting this vulnerability in the wild
- *Should Company A release their patch, even if doing so means that the vulnerability now becomes public and other actors can start exploiting Companies B and C?*
- *Or should Company A wait until Companies B and C have patches?*

# Next Major Section of the Course: Cryptography

# Terminology Note: “blockchain” and “crypto”

- Rising interest, mostly in the cryptocurrency space
- For this course: crypto means “cryptography”

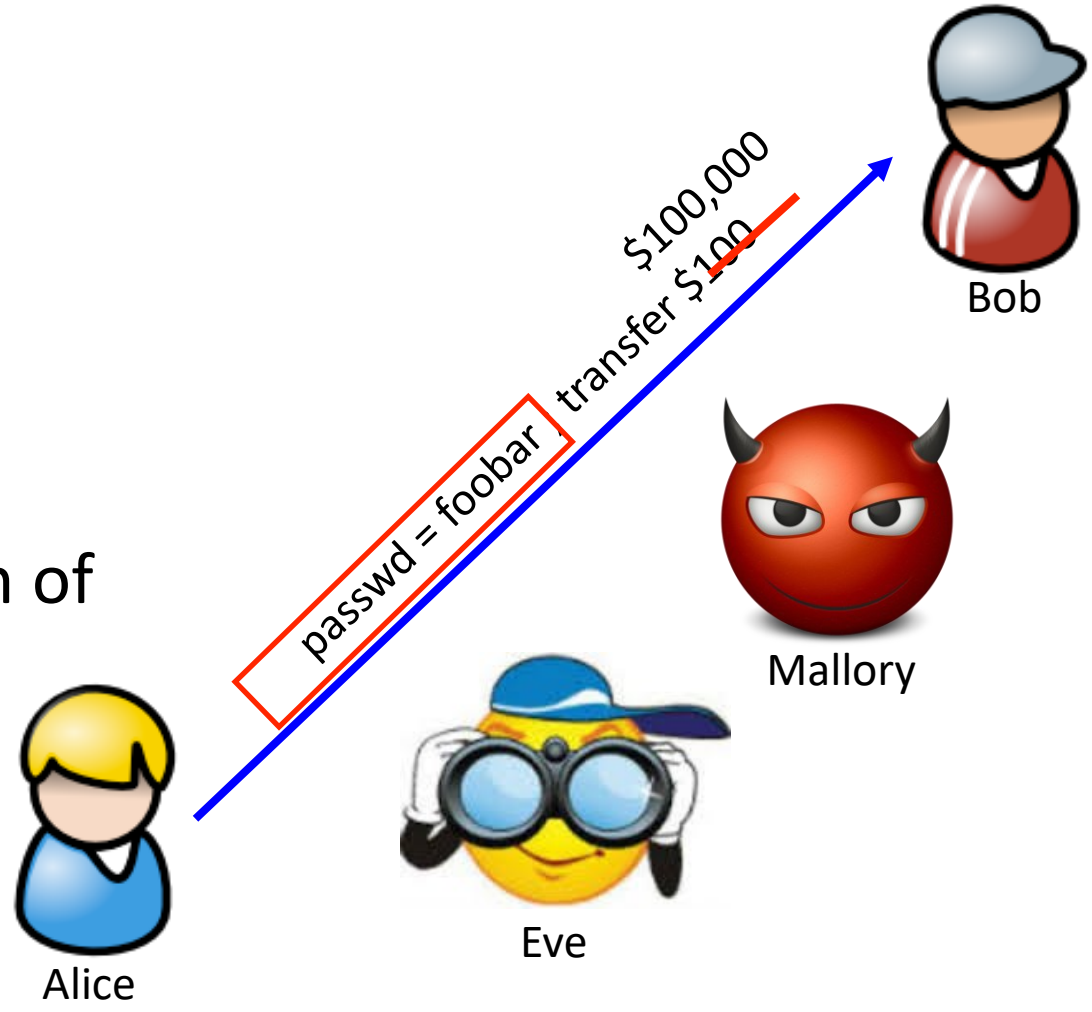
# Common Communication Security Goals

**Privacy of data:**

Prevent exposure of information

**Integrity of data:**

Prevent modification of information

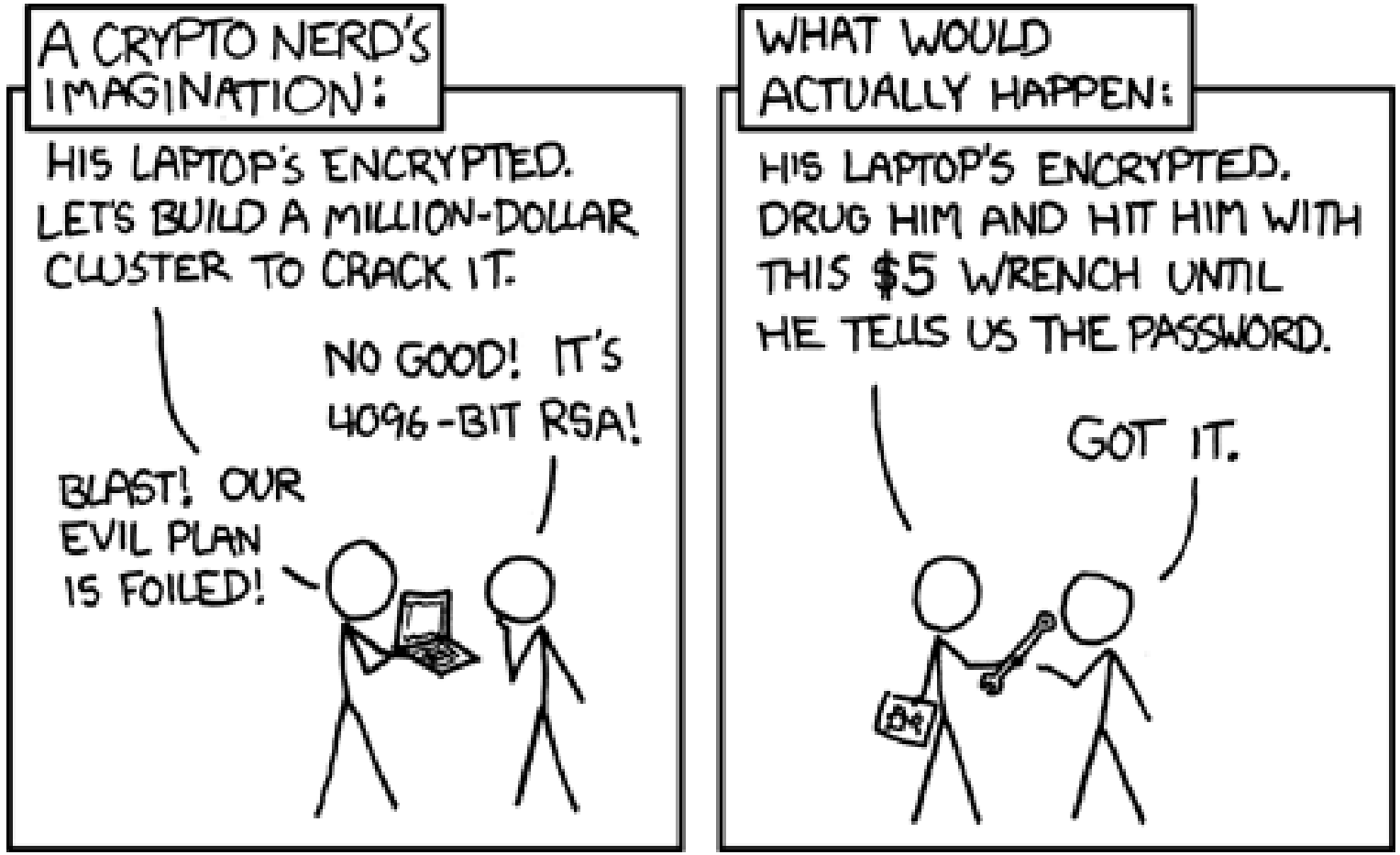


# Recall Bigger Picture

- Cryptography only one small piece of a larger system
- Must protect entire system
  - Physical security
  - Operating system security
  - Network security
  - Users
  - Cryptography (following slides)
- Recall the weakest link

- Still, cryptography is a crucial part of our toolbox

# XKCD: http://xkcd.com/538/



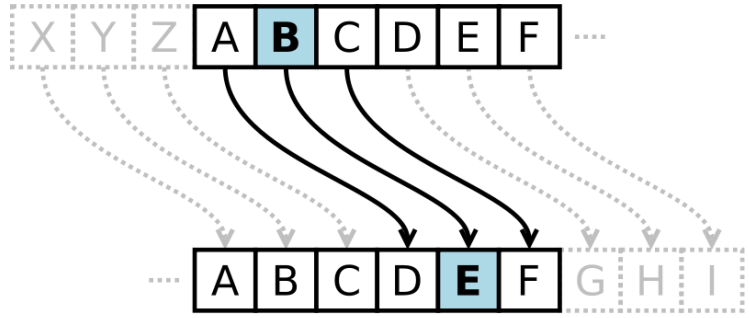
# History

- Substitution Ciphers
  - Caesar Cipher
- Transposition Ciphers
- Codebooks
- Machines

- Recommended Reading: **The Codebreakers** by David Kahn and **The Code Book** by Simon Singh.

# History: Caesar Cipher (Shift Cipher)

- Plaintext letters are replaced with letters fixed shift away in the alphabet.




a

- Example:
  - Plaintext: `The quick brown fox jumps over the lazy dog`
  - Key: Shift 3

    `ABCDEFGHIJKLMNOPQRSTUVWXYZ`

    `DEFGHIJKLMNOPQRSTUVWXYZABC`
  - Ciphertext: `WKHTX LFNEU RZQIR AMXPS VRYHU WKHOD CBGRJ`

# History: Caesar Cipher (Shift Cipher)

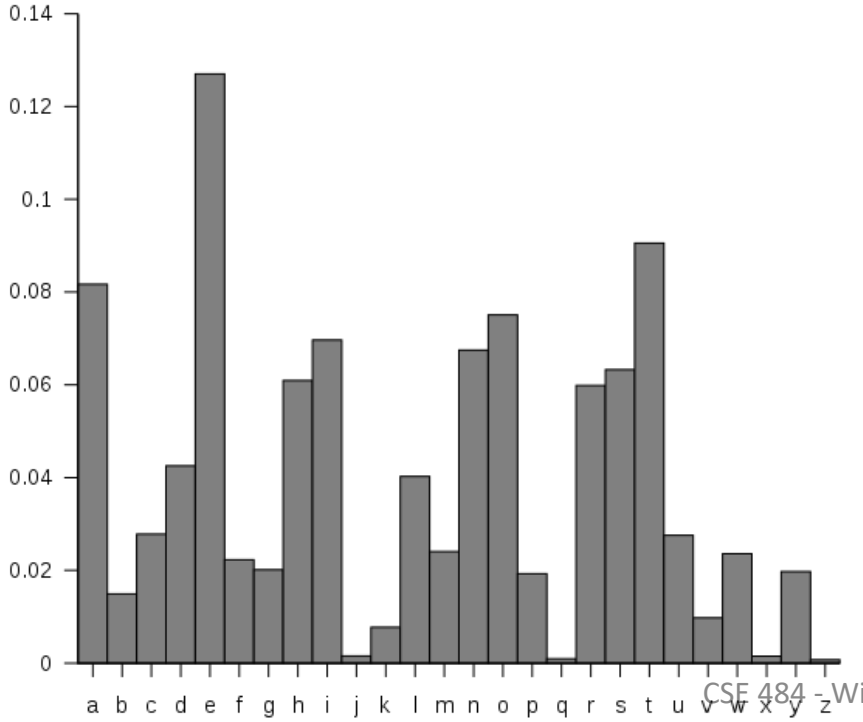
- ROT13: shift 13 (encryption and decryption are symmetric)
- What is the key space?
  - 26 possible shifts.
- How to attack shift ciphers?
  - Brute force.

# History: Substitution Cipher

- Superset of shift ciphers: each letter is substituted for another one.
- One way to implement: Add a secret key
- Example:
  - Plaintext: ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - Cipher: ZEBRASCDFGHIJKLMNOPQTUVWXY
- "State of the art" for thousands of years

# History: Substitution Cipher

- What is the key space?
- How to attack?
  - Frequency analysis.




26! ~= 2^88

**Bigrams:**

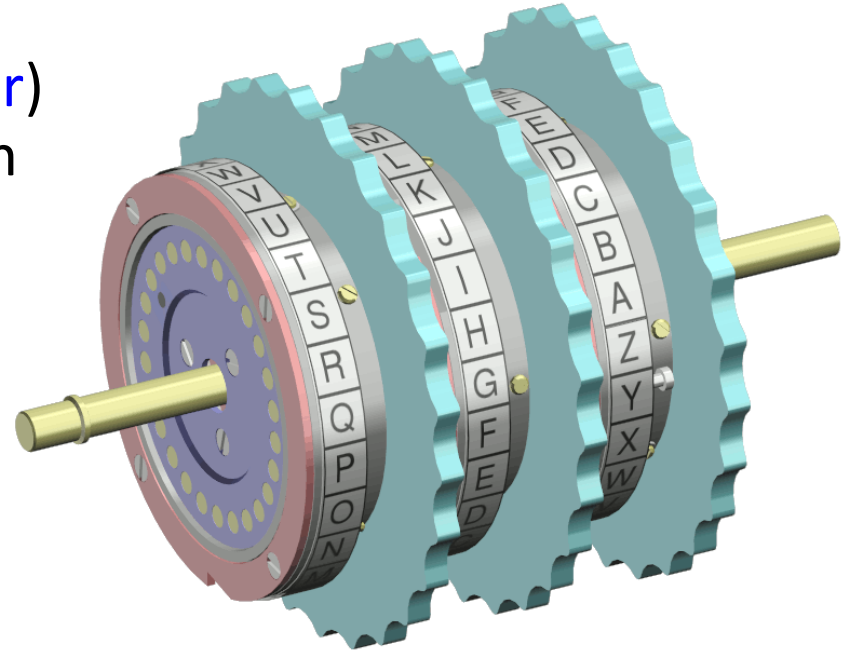
| | | |
|---|---|---|
| th 1.52% | en 0.55% | ng 0.18% |
| he 1.28% | ed 0.53% | of 0.16% |
| in 0.94% | to 0.52% | al 0.09% |
| er 0.94% | it 0.50% | de 0.09% |
| an 0.82% | ou 0.50% | se 0.08% |
| re 0.68% | ea 0.47% | le 0.08% |
| nd 0.63% | hi 0.46% | sa 0.06% |
| at 0.59% | is 0.46% | si 0.05% |
| on 0.57% | or 0.43% | ar 0.04% |
| nt 0.56% | ti 0.34% | ve 0.04% |
| ha 0.56% | as 0.33% | ra 0.04% |
| es 0.56% | te 0.27% | ld 0.02% |
| st 0.55% | et 0.19% | ur 0.02% |

**Trigrams:**

| | | |
|---|---|---|
| 1. the | 6. ion | 11. nce |
| 2. and | 7. tio | 12. edt |
| 3. tha | 8. for | 13. tis |
| 4. ent | 9. nde | 14. oft |
| 5. ing | 10.has | 15. sth |

# History: Enigma Machine

Uses rotors (substitution cipher) that change position after each key.




Key = initial setting of rotors

Key space?

26^n for n rotors

# 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)
- 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!)

# The Cryptosystem Stack

- Primitives:
  - AES / DES / etc
  - RSA / ElGamal / Elliptic Curve (ed25519)
- Modes:
  - Block modes (CBC, ECB, CTR, GCM, …)
  - Padding structures
- Protocols:
  - TLS / SSL / SSH / etc
- Usage of Protocols:
  - Browser security
  - Secure remote logins

# 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.

- Foreshadow: Need for randomness – the key to keep private

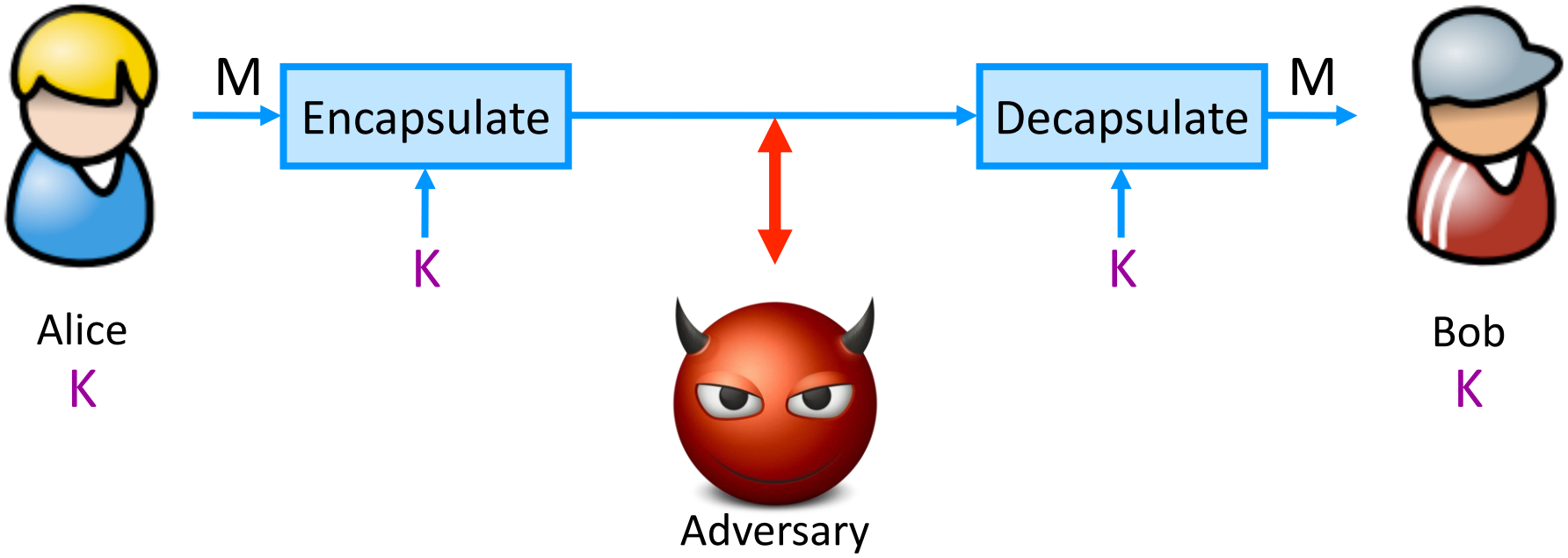
# Flavors of Cryptography

- 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 understand, and revolutionary! Inventors won Turing Award* ☺

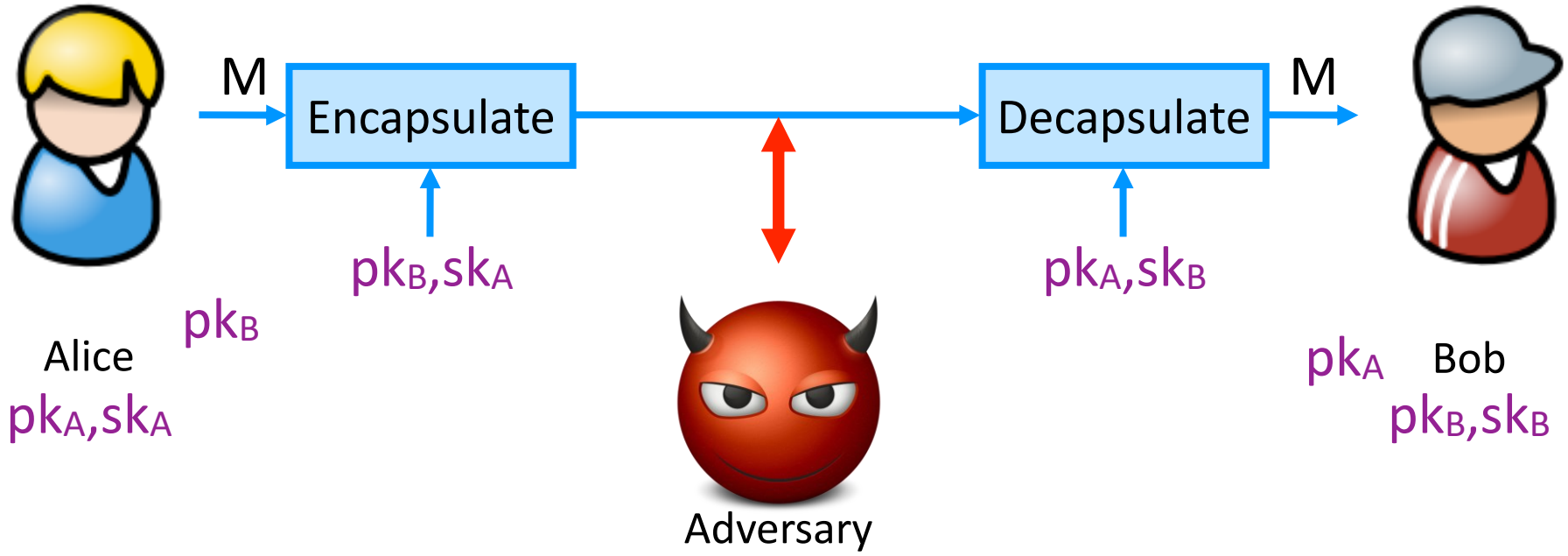
# Symmetric Setting

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.



Received April 4, 1977

# A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R.L. Rivest, A. Shamir, and L. Adleman*

**Abstract**

An encryption method is presented with the novel property that publicly revealing an encryption key does not thereby reveal the corresponding decryption key. This has two important consequences:

1. Couriers or other secure means are not needed to transmit keys, since a message can be enciphered using an encryption key publicly revealed by the intended recipient. Only he can decipher the message, since only he knows the corresponding decryption key.
2. A message can be "signed" using a privately held decryption key. Anyone can verify this signature using the corresponding publicly revealed encryption key. Signatures cannot be forged, and a signer cannot later deny the validity of his signature. This has obvious applications in "electronic mail" and "electronic funds transfer" systems.

# 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?

# 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 won't know and can't predict and 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)

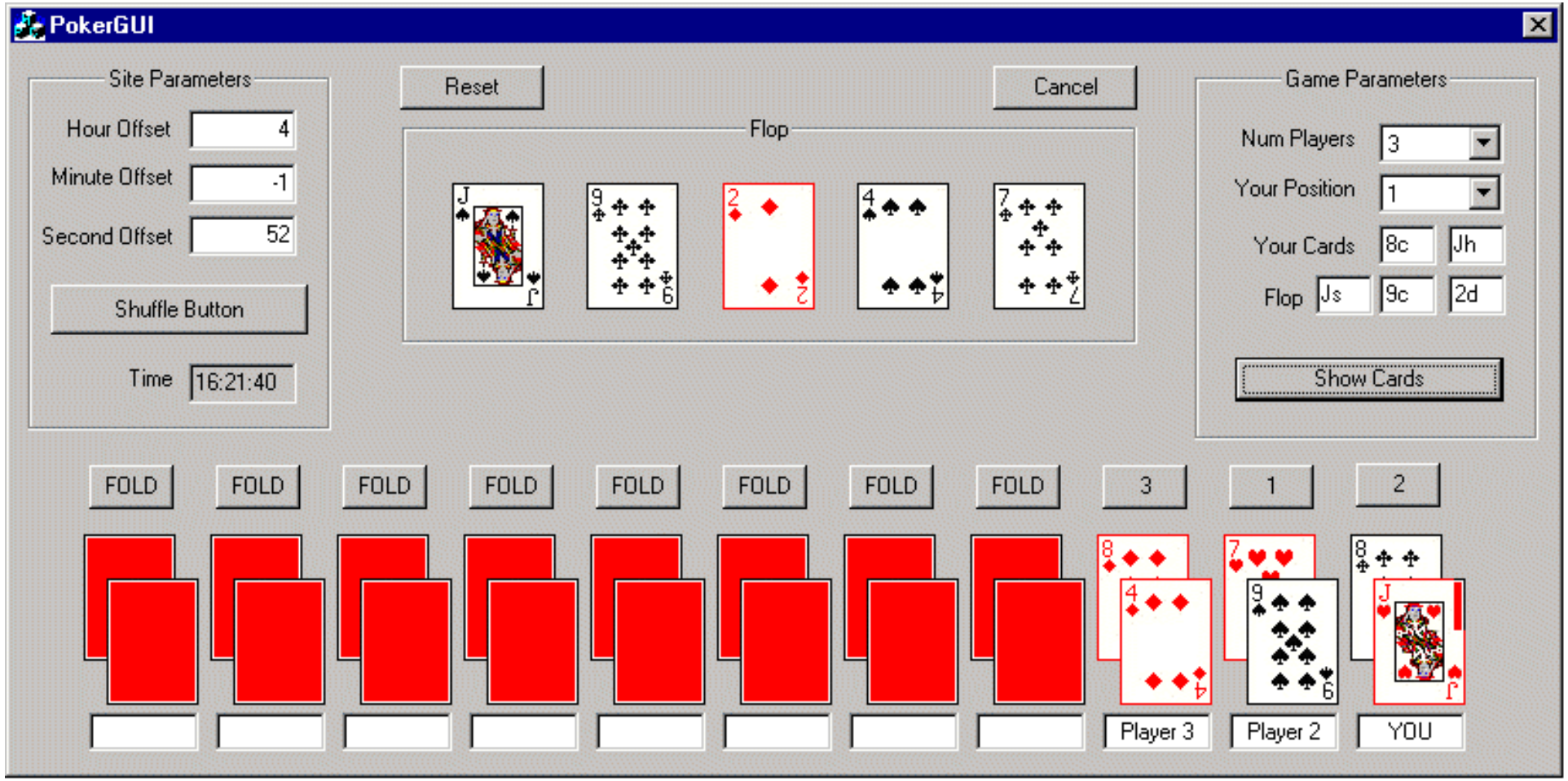
# C's rand() Function

- 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;
}
```

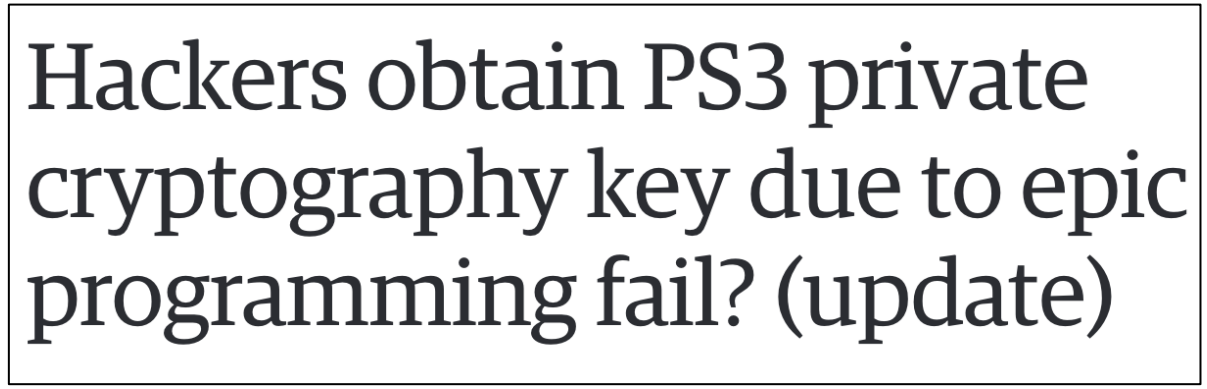
- Problem: don't use rand() for security-critical applications!
  - Given a few sample outputs, you can predict subsequent ones


A World of Action!
Help
Suggestions
Fold in Turn
$0
jonnyboy
brute
Fold
Check
Bet ($5)
mamajoe
RickG
Options
Sit Out
Leave
584330
mamajoe: Hey guys, Big B is in!
Dealer Text




More details: "How We Learned to Cheat at Online Poker: A Study in Software Security"
http://www.cigital.com/papers/download/developer_gambling.php

# PS3 and Randomness




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

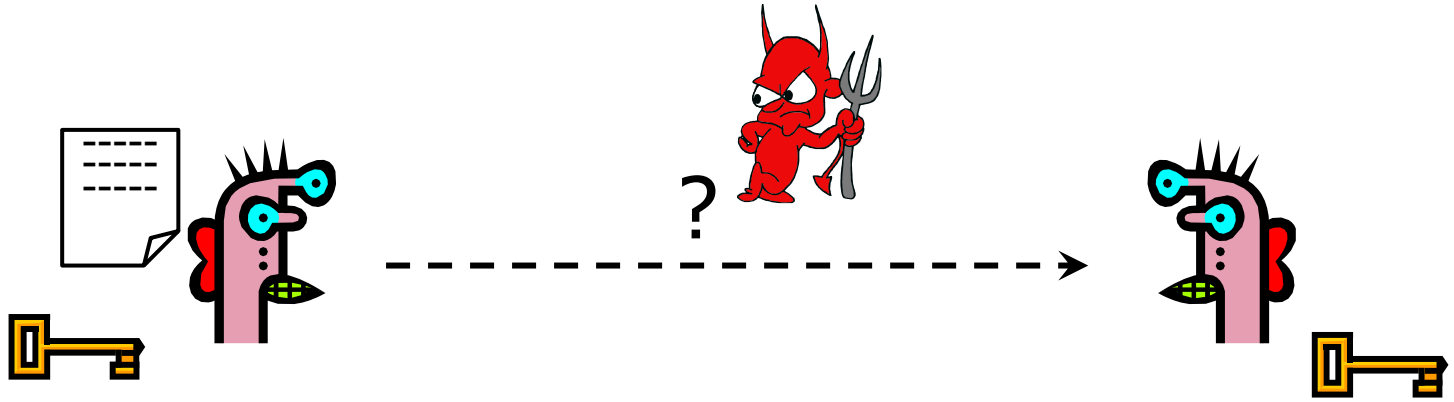
- Linux:
  - /dev/random – blocking (waits for enough entropy)
  - /dev/urandom – nonblocking, possibly less entropy
  - getrandom() – syscall! – by default, blocking
- Internally:
  - Entropy pool gathered from multiple sources
    - e.g., mouse/keyboard/network timings
- Challenges with embedded systems,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

# Obtaining *Random* Numbers

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

# Back to encryption

# Confidentiality: Basic Problem




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

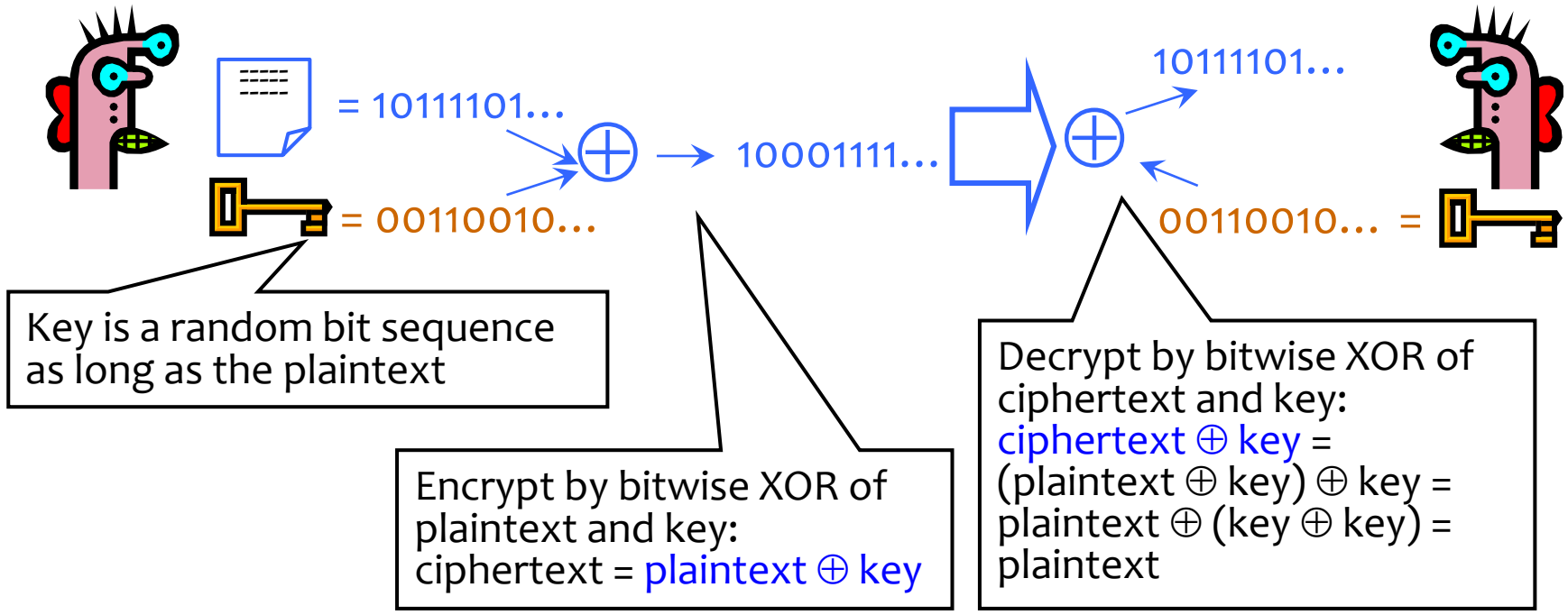
Goal: send a message confidentially.

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

# One weird bit-level trick

- XOR!
  - Just XOR with a random bit!
- Why?
  - Uniform output
  - Independent of ‘message’ bit

# 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)

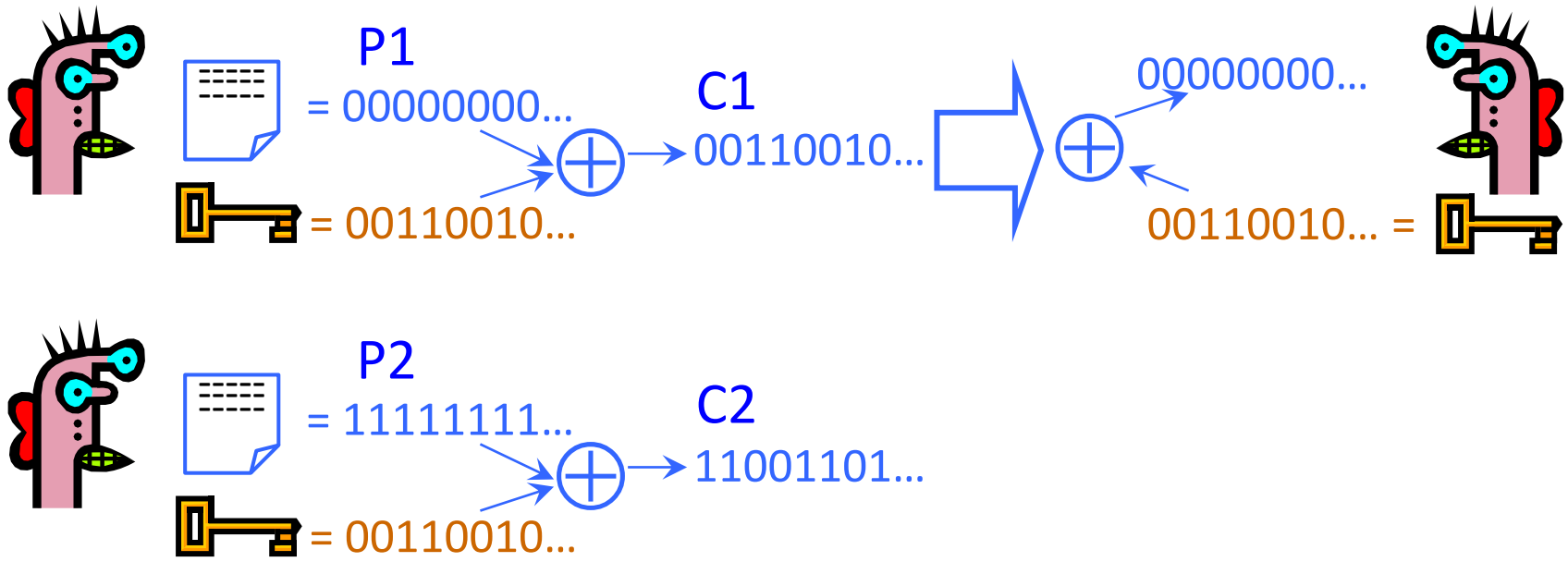
# Advantages of One-Time Pad

- 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?

- Breakout Discussions
- What potential security problems do you see with the one-time pad?
- (Try not to look ahead and next slides)
- Recall two key goals of cryptography: confidentiality and integrity

# Dangers of Reuse




Learn relationship between plaintexts

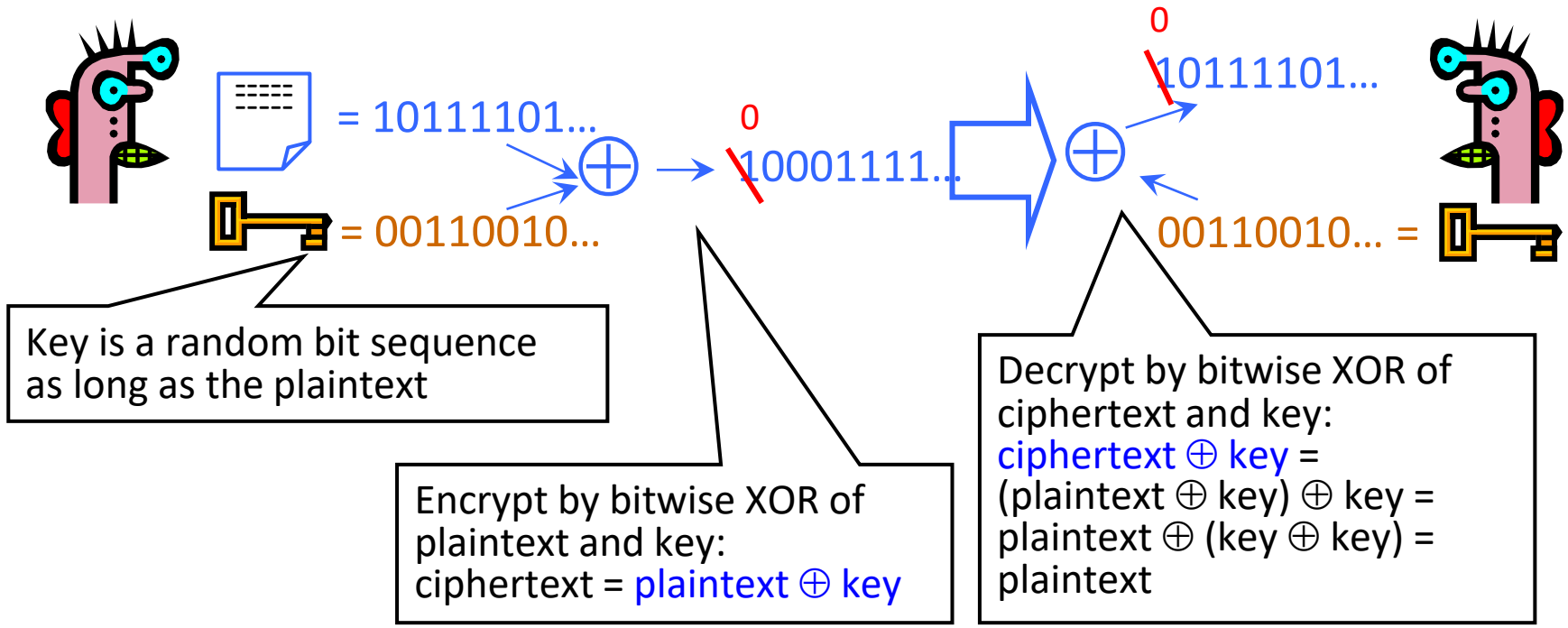
C1⊕C2 = (P1⊕K)⊕(P2⊕K) =

(P1⊕P2)⊕(K⊕K) = P1⊕P2

# 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?


= 10111101…
= 00110010…
0
10001111…
0
10111101…
00110010… =
Key is a random bit sequence as long as the plaintext
Encrypt by bitwise XOR of plaintext and key: ciphertext = plaintext ⊕ key
Decrypt by bitwise XOR of ciphertext and key: ciphertext ⊕ key = (plaintext ⊕ key) ⊕ key = plaintext ⊕ (key ⊕ key) = plaintext

# 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**