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

Fall 2024

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 Monday
  - Remember to turn things in both:
    - **Together:** Your sploitX.c files
    - **Individually:** Writeups for each sploit

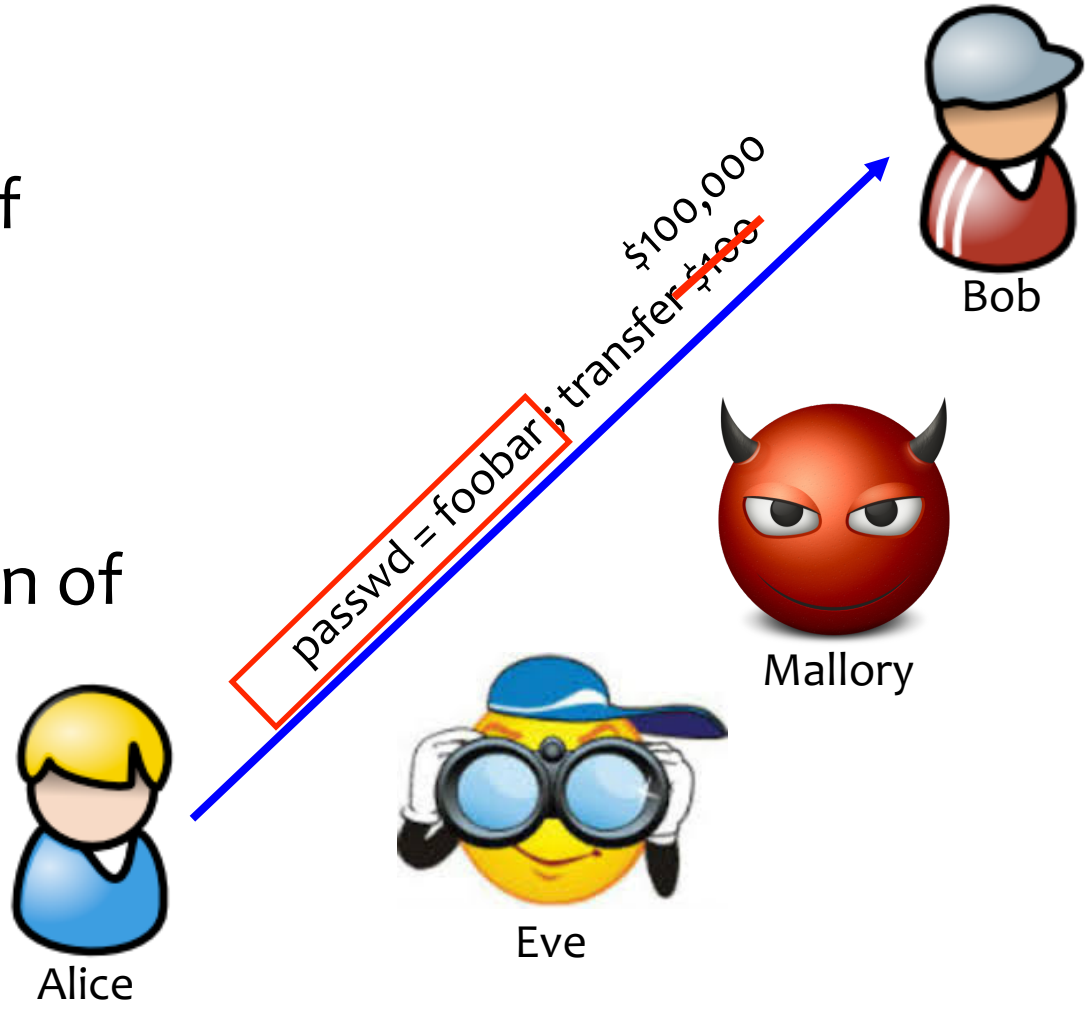
Recall: Common Communication Security Goals

Privacy of data:

Prevent exposure of information

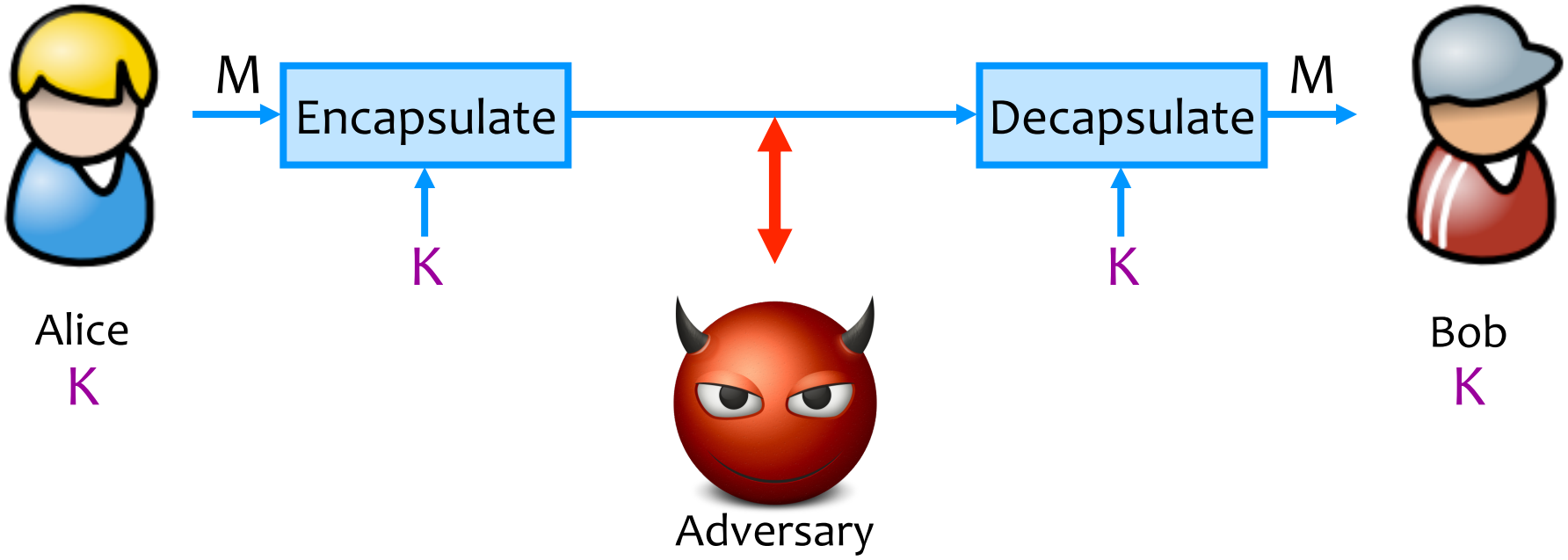
Integrity of data:

Prevent modification of information



Recall: Symmetric Setting

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



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)

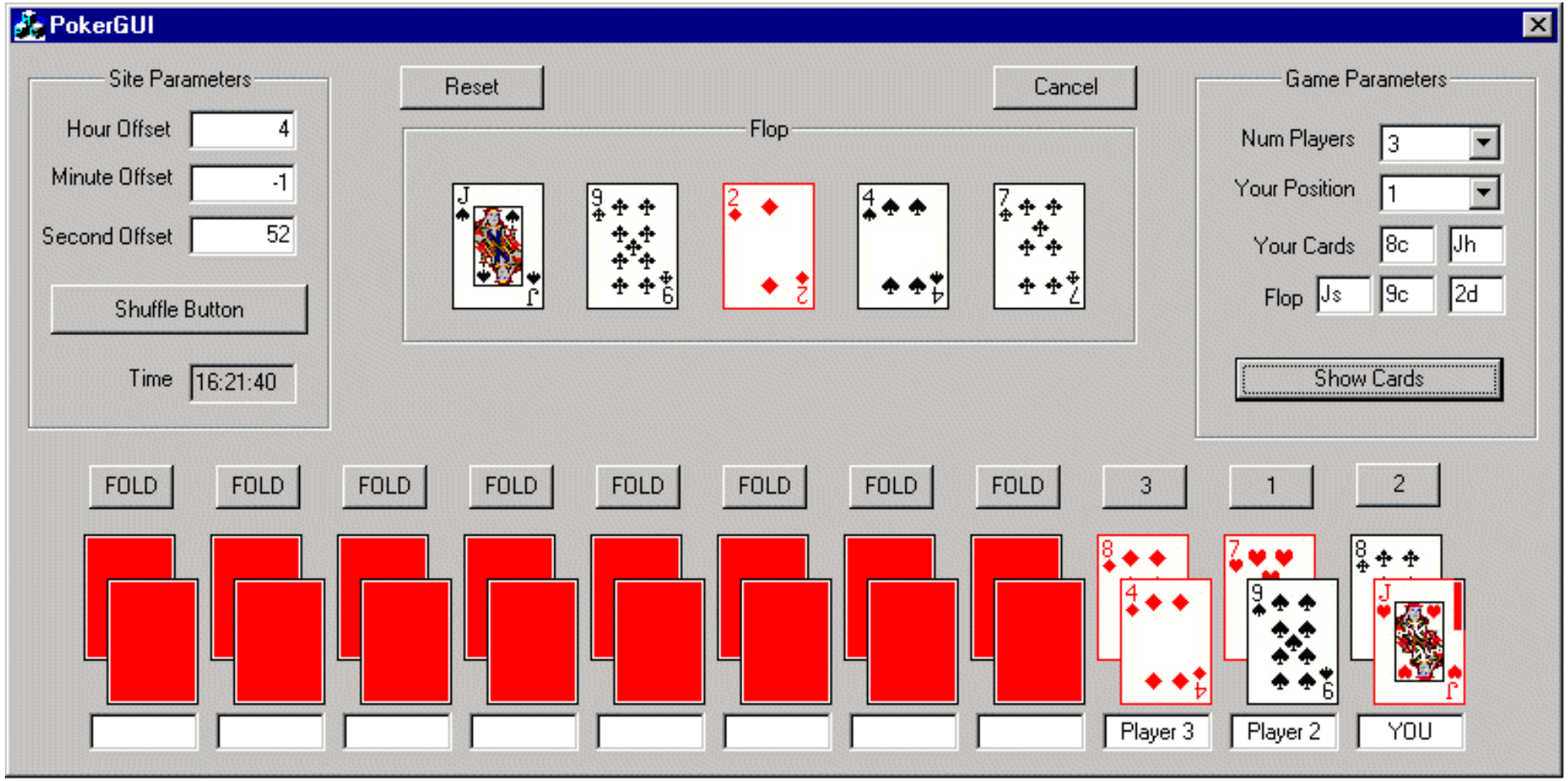
C's rand() Function

- 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

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

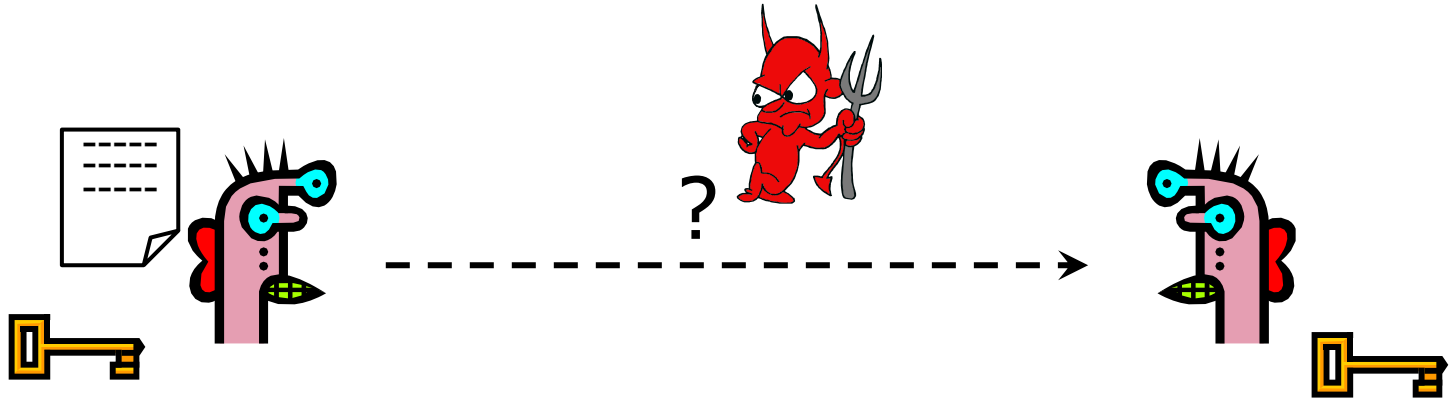
Obtaining *Random* Numbers

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

Now that we have some randomness, let's do:

Symmetric Encryption

Confidentiality: Basic Problem

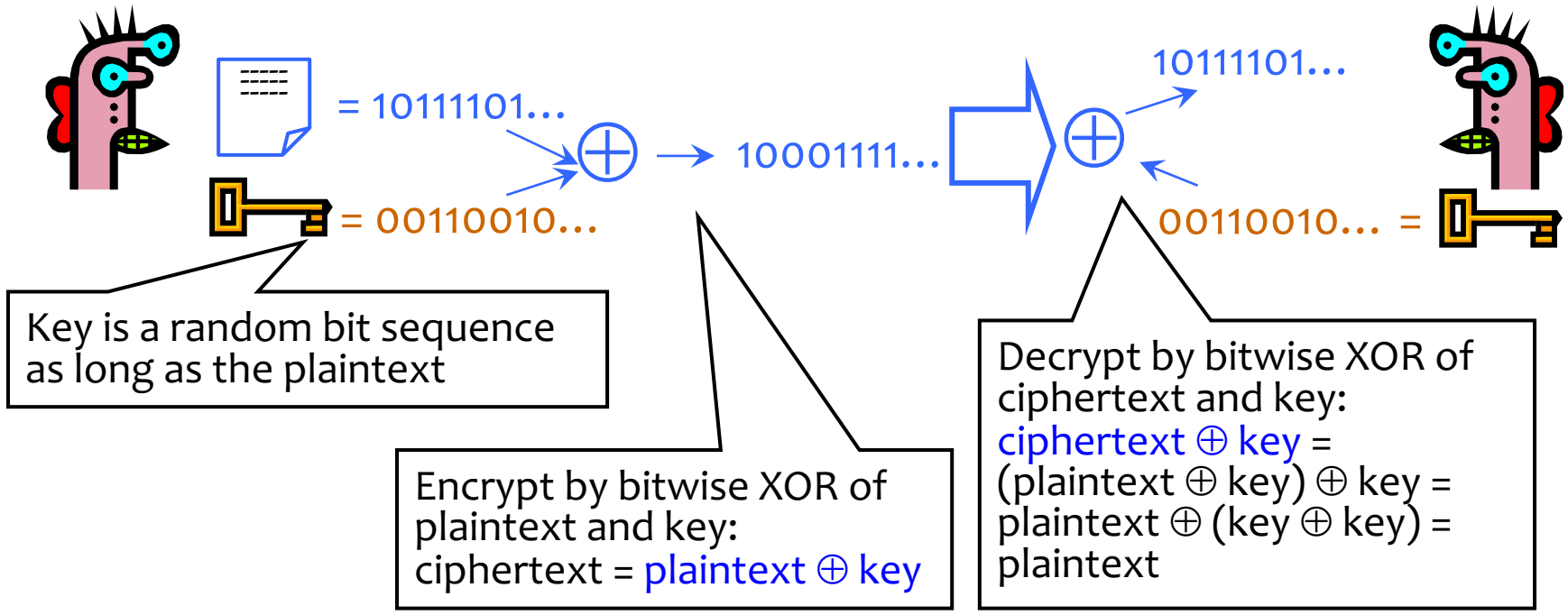


Given (Symmetric Crypto): both parties know the same secret.

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)

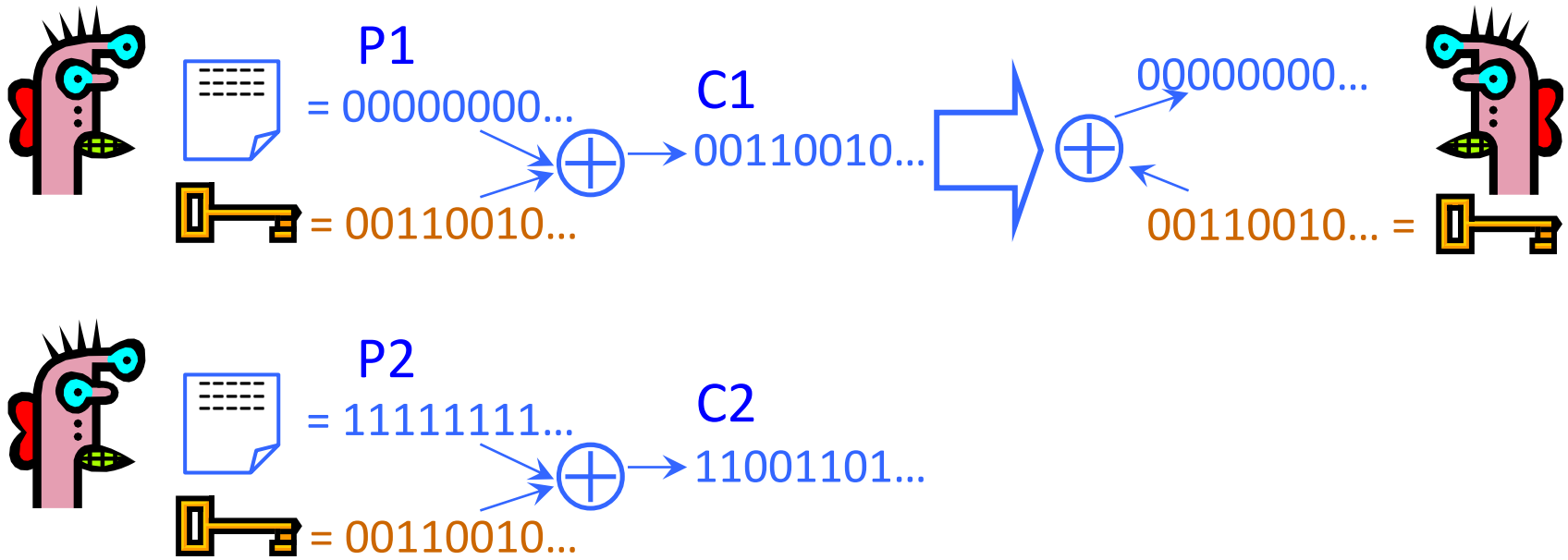
Advantages of One-Time Pad

- 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?

Dangers of Reuse



Learn relationship between plaintexts

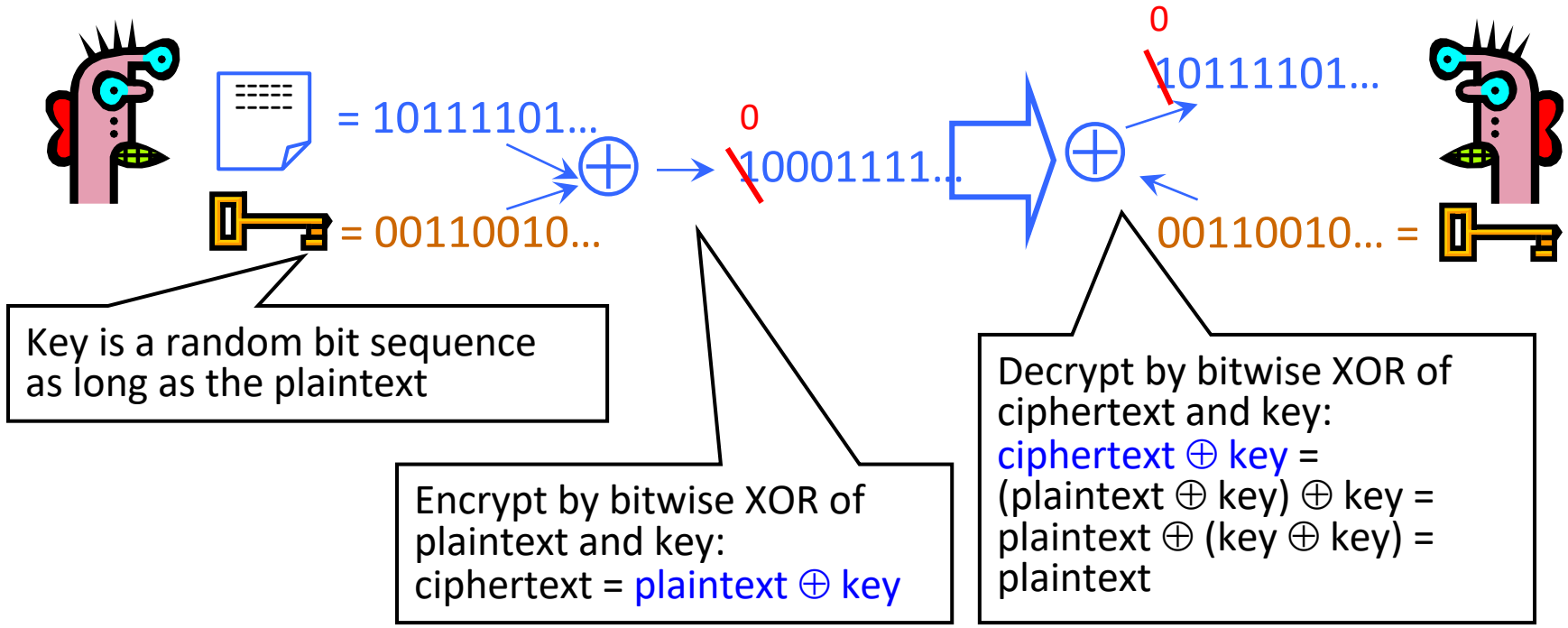
C1⊕C2 = (P1⊕K)⊕(P2⊕K) =

(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

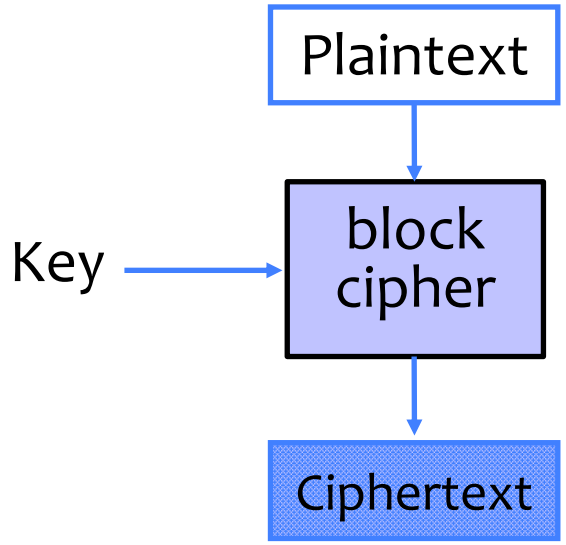
- (1) Key must be as long as the plaintext
  - Impractical in most realistic scenarios
  - Still used for diplomatic and intelligence traffic
- (2) Insecure if keys are reused
  - Attacker can obtain XOR of plaintexts
- **(3) Does not guarantee integrity**
  - **One-time pad only guarantees confidentiality**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

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

Block Ciphers

- Operates on a single chunk ("block") of plaintext
  - For example, 64 bits for DES, 128 bits for AES
  - Each key defines a different **permutation** of inputs to possible outputs
  - Same key is reused for each block (can use short keys)



Keyed Permutation

	Key = 00	Key = 01	
input	**possible output**	**possible output**	**etc.**
000	010	111	…
001	111	110	…
010	101	000	…
011	110	101	…
…	…		…
111	000	110	…

For N-bit input:

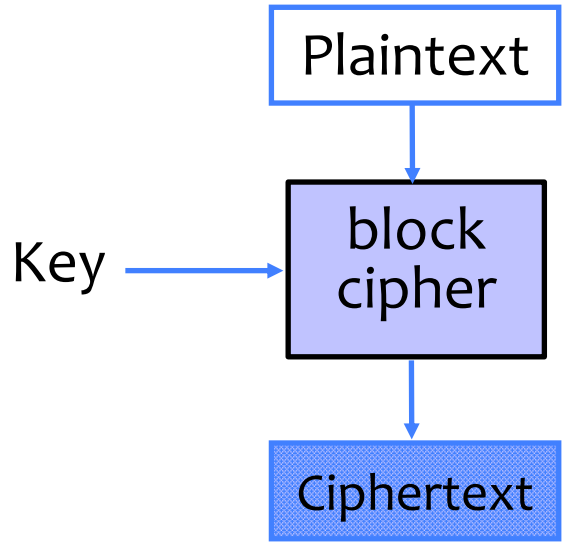
2^N! possible permutations

For K-bit key:

2^K possible keys

Keyed Permutation

- Not just shuffling of input bits!
  - Suppose plaintext = "111".
  - Then "111" is not the only possible ciphertext!
- Instead:
  - **Permutation of possible outputs**
  - Use secret key to pick a permutation



Block Cipher Security

- Result should "look like" a random permutation on the inputs
  - Recall: not just shuffling bits. N-bit block cipher permutes over 2^N inputs.

- Only computational guarantee of secrecy
  - Not impossible to break, just very expensive
    - If there is no efficient algorithm (unproven assumption!), then can only break by brute-force, try-every-possible-key search
  - Time and cost of breaking the cipher exceed the value and/or useful lifetime of protected information