CSE 484: Computer Security and Privacy

Cryptography
[Symmetric Encryption]

Spring 2021

Tadayoshi Kohno

yoshi@cs

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, David Kohlbrenner, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Admin

- Lab 1 checkpoint on Wednesday

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

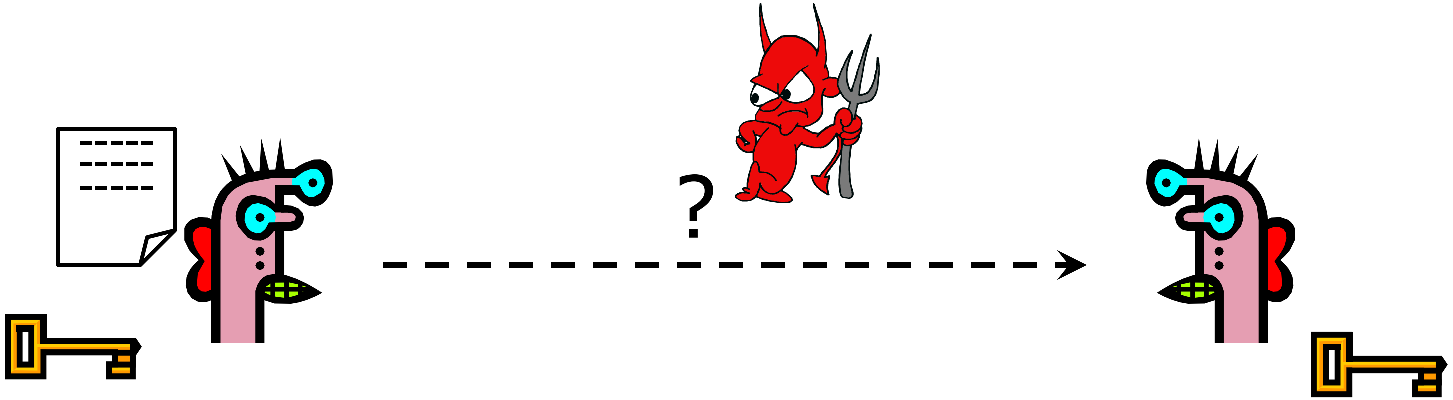
- Linux:
  - /dev/random – blocking (waits for enough entropy)
  - /dev/urandom – nonblocking, possibly less entropy
  - getrandom() – syscall! – by default, blocking
- Internally:
  - Entropy pool gathered from multiple sources
    - e.g., mouse/keyboard/network timings
- Challenges with embedded systems,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

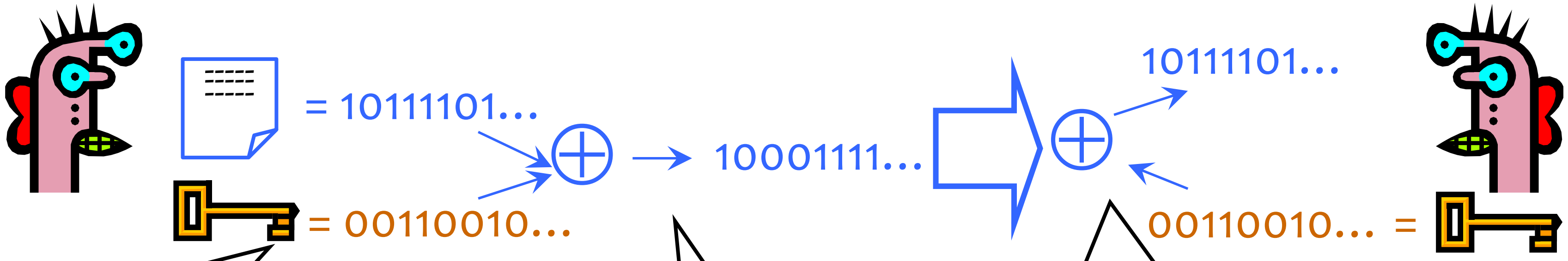
Goal: send a message confidentially.

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

One weird bit-level trick

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

One-Time Pad



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

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?

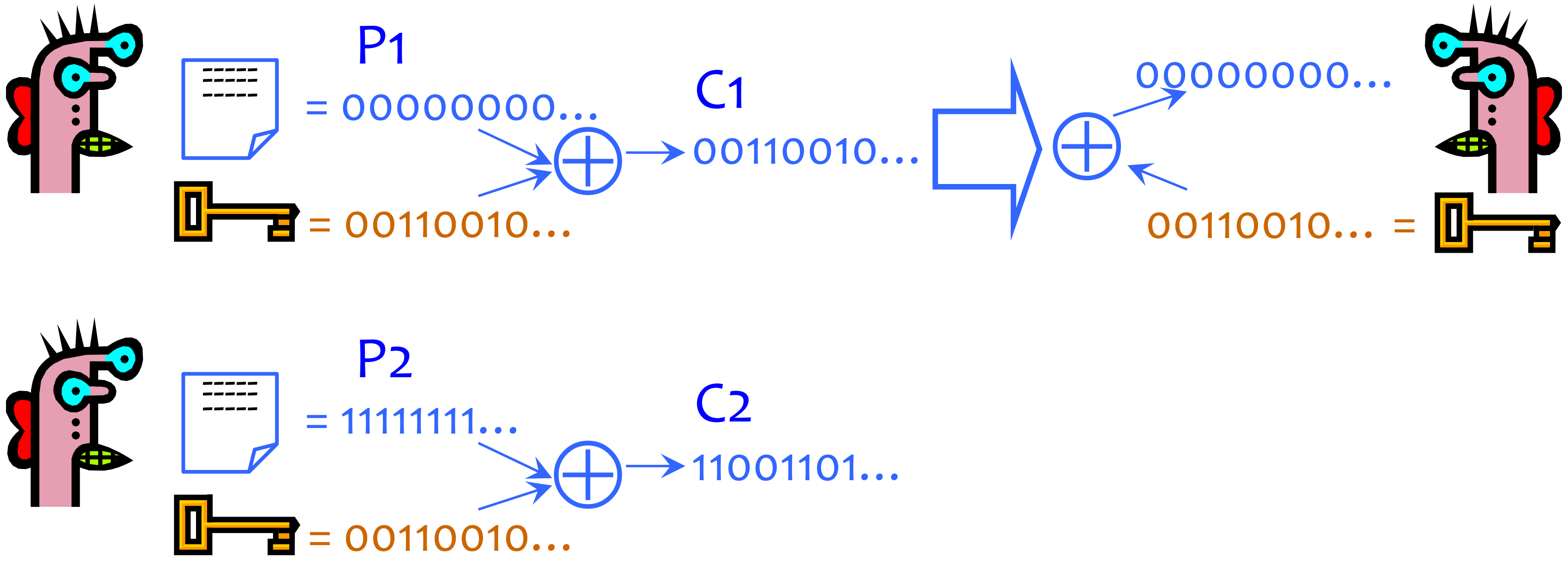
Problems with the One-Time Pad?

- 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

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

Dangers of Reuse



Learn relationship between plaintexts

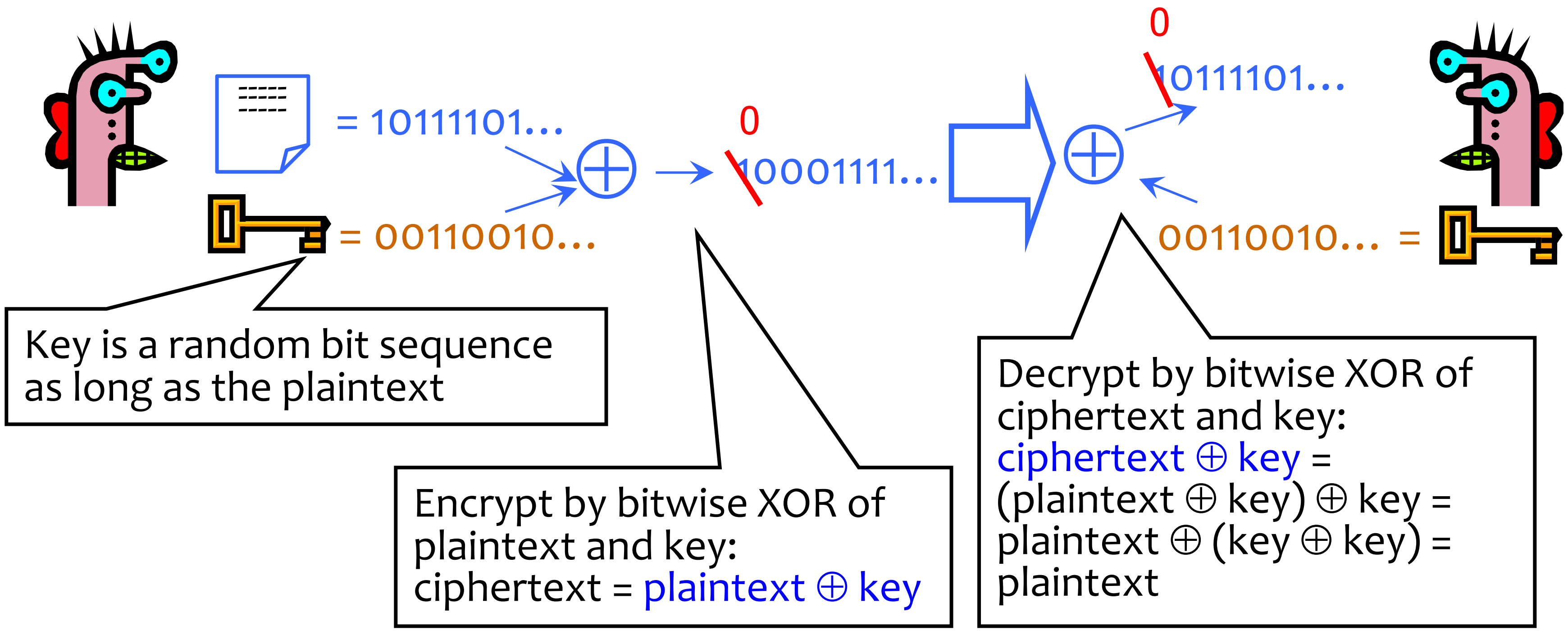
$C1 \oplus C2 = (P1 \oplus K) \oplus (P2 \oplus K) =$

$(P1 \oplus P2) \oplus (K \oplus K) = P1 \oplus P2$

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

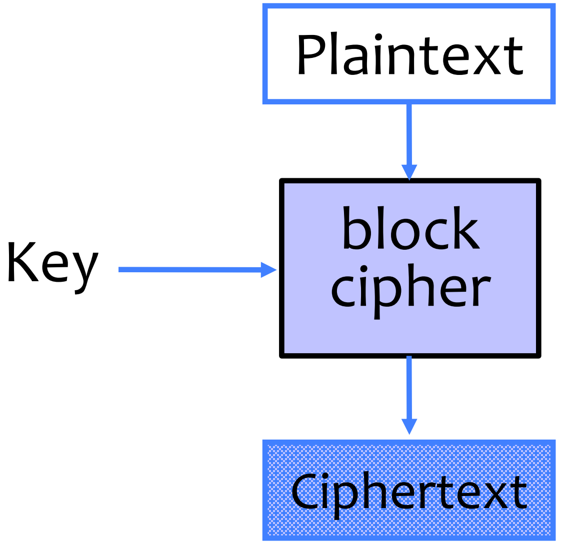
- (1) Key must be as long as the plaintext
  - Impractical in most realistic scenarios
  - Still used for diplomatic and intelligence traffic
- (2) Insecure if keys are reused
  - Attacker can obtain XOR of plaintexts
- **(3) Does not guarantee integrity**
  - **One-time pad only guarantees confidentiality**
  - **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
  - Same key is reused for each block (can use short keys)



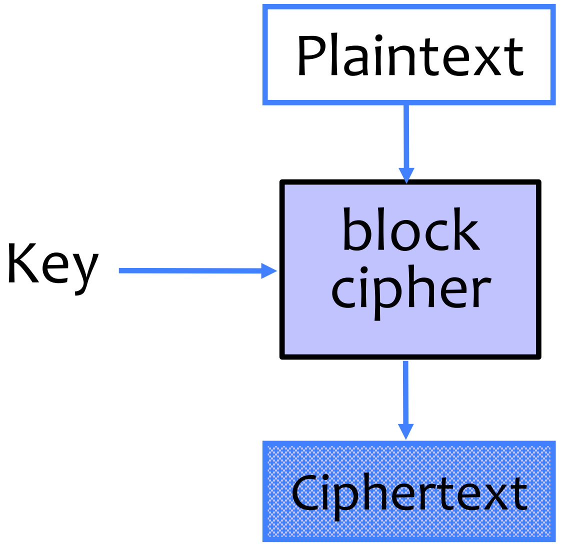
Keyed Permutation

input	possible output (K=00)	possible output (K=01)	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
  - "Break" could mean recovering key, or it could mean distinguishing the block cipher's behavior from that of a randomly selected permutation over the 2^N possible inputs