CSE 484 / CSE M 584: Computer Security and Privacy

Autumn 2019

Tadayoshi (Yoshi) Kohno
yoshi@cs.Washington.edu

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

Announcements

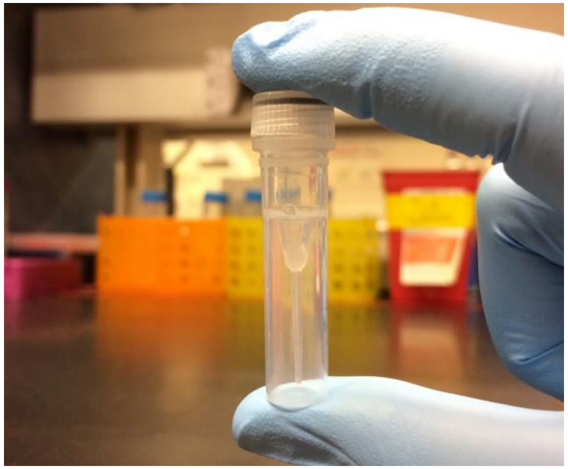
- TA office hours as usual, and great place for lab discussions
- I will again mute HW1 ☺, until I’ve looked at all of them
- Guest lecturers next Monday and Wednesday ☺ But no office hours for me next week ☹

Misc

- How to think about security, paranoia, etc

Research Discussions

- Monday (10/14): Peter Ney on Bio-Cyber Security and Cell Site Simulators
- Monday (10/21): Karl Koscher on Automotive Cyber Security
- Wednesday (10/23): Ivan Evtimov on Adversarial Machine Learning
- Monday (10/28): Emily McReynolds on Law and Policy



Broad Classes of Security Research

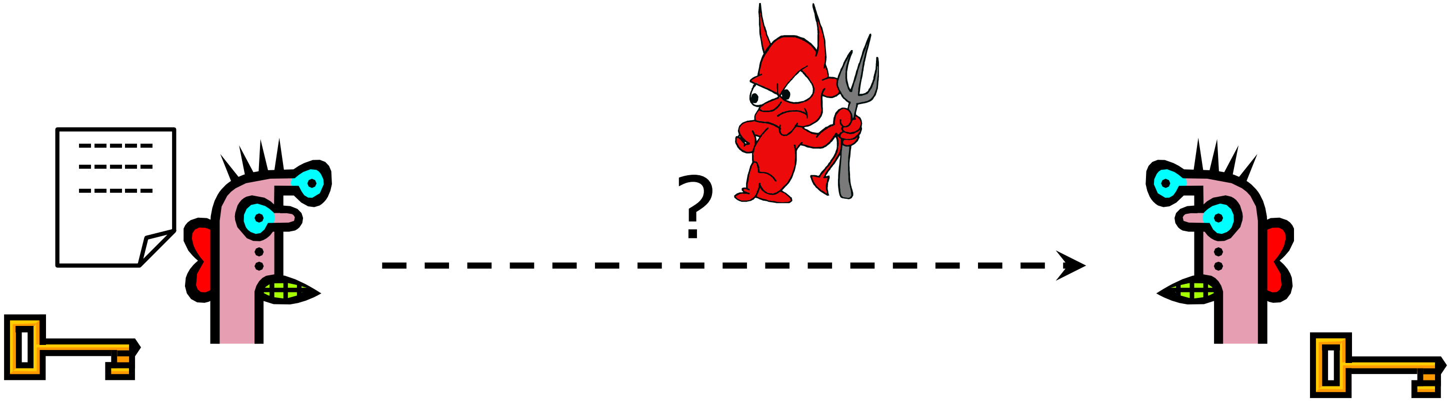
- Measurement
- Analysis / attack exploration
- Building secure systems
- Human-computer interaction

- Guest lectures connected to threat modeling and to buffer overflows as well

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?

Confidentiality: Basic Problem

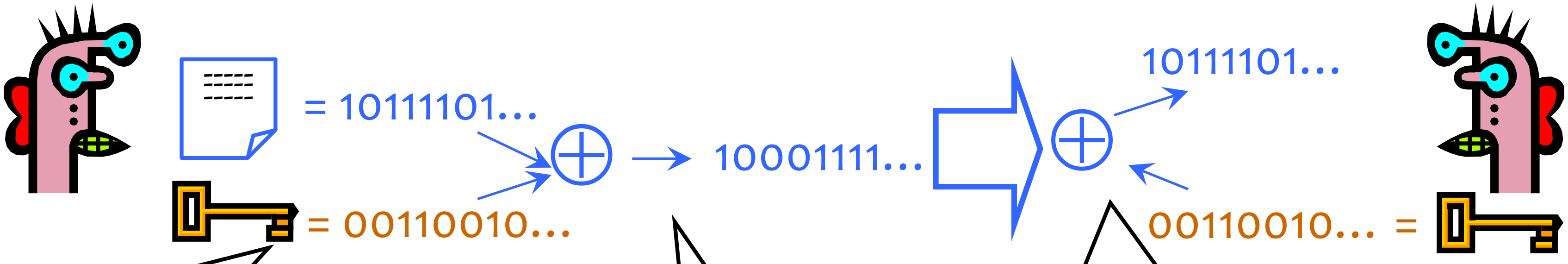


Given (Symmetric Crypto): both parties know the same secret.

Goal: send a message confidentially.

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

One-Time Pad



= 10111101…

= 00110010…

⊕ → 10001111…

⊕

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

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)

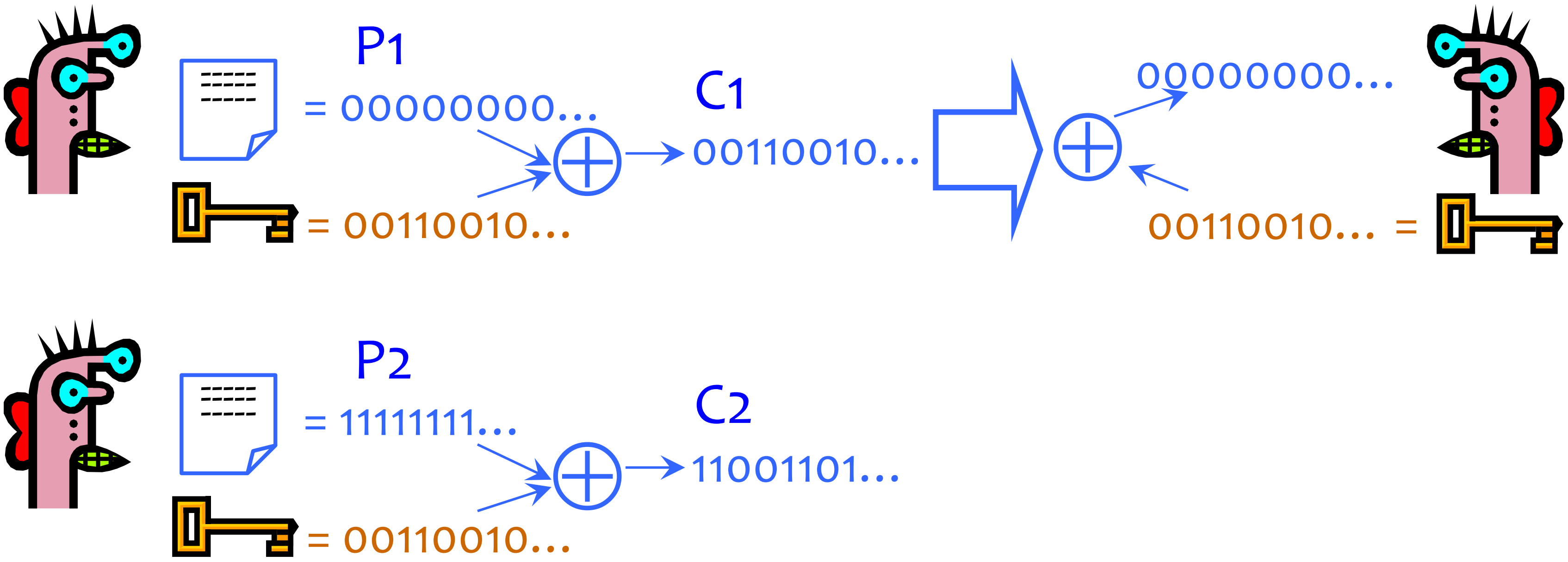
Advantages of One-Time Pad

- 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 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



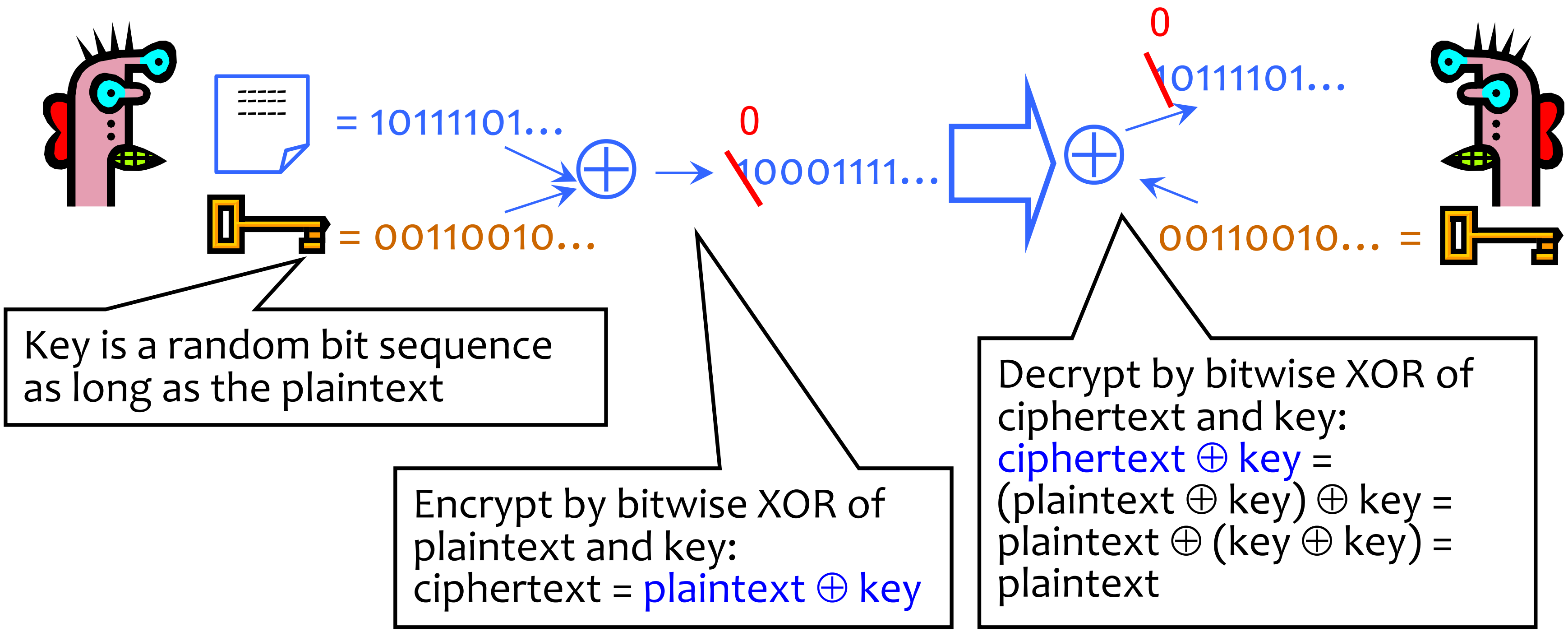
Learn relationship between plaintexts

C1⊕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?



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

Stream Ciphers

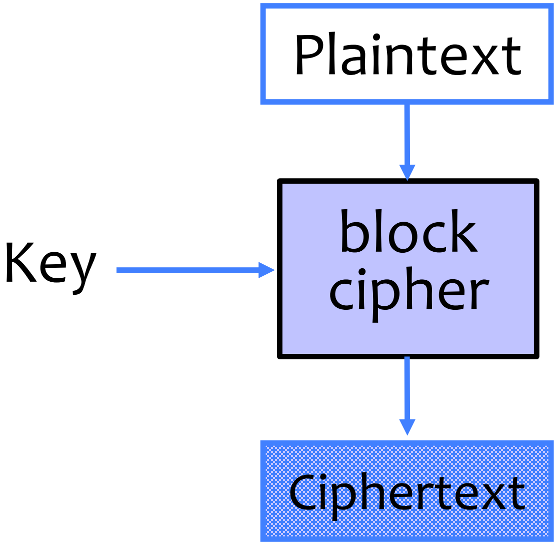
- One-time pad: Ciphertext(Key,Message)=Message⊕Key
  - Key must be a random bit sequence as long as message
- Idea: replace "random" with "pseudo-random"
  - Use a pseudo-random number generator (PRNG)
  - PRNG takes a short, truly random secret seed and expands it into a long "random-looking" sequence
    - E.g., 128-bit seed into a 10^6-bit pseudo-random sequence

No efficient algorithm can tell this sequence from truly random

- Ciphertext(Key,Msg)=Msg⊕PRNG(Key)
  - Message processed bit by bit (unlike block cipher)

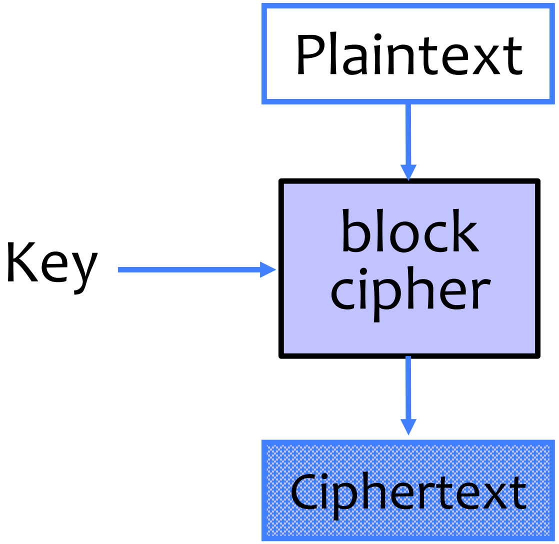
Block Ciphers

- 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

- Not just shuffling of input bits!
  - Suppose plaintext = "111". Then "111" is not the only possible ciphertext!
- Instead:
  - Permutation of possible outputs
  - For N-bit input, $2^N!$ possible permutations
  - Use secret key to pick a permutation



Example: With 3-bit Blocks

Key = 0000000

Input	Output
000	111
001	101
010	001
011	000
100	110
101	010
110	100
111	011

Key = 0000001

Input	Output
000	000
001	101
010	010
011	001
100	100
101	011
110	111
111	110

Key = 0000010

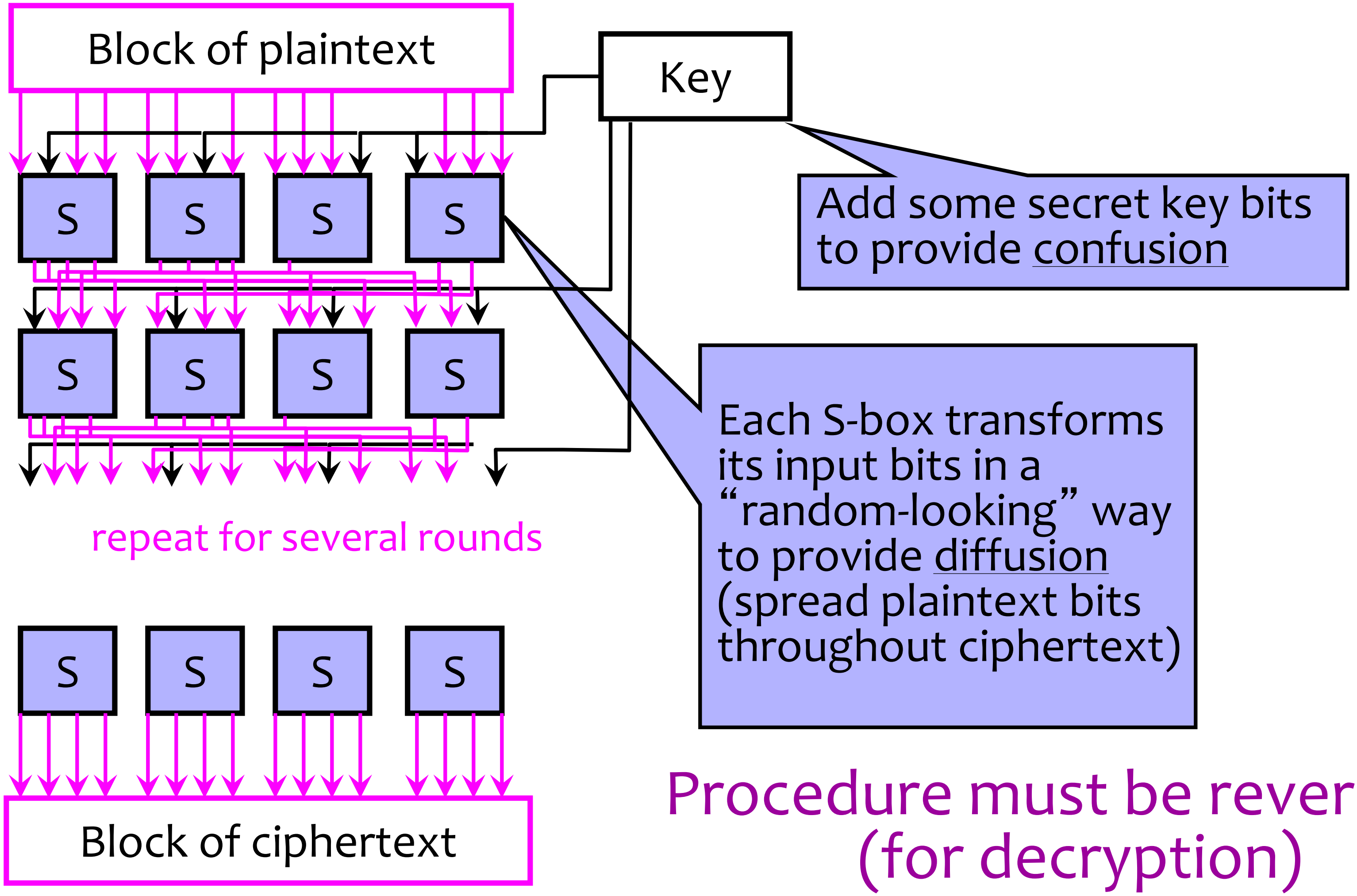
Input	Output
000	001
001	000
010	010
011	011
100	111
101	101
110	100
111	110

…

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

Block Cipher Operation (Simplified)



Add some secret key bits to provide confusion

Each S-box transforms its input bits in a "random-looking" way to provide diffusion (spread plaintext bits throughout ciphertext)

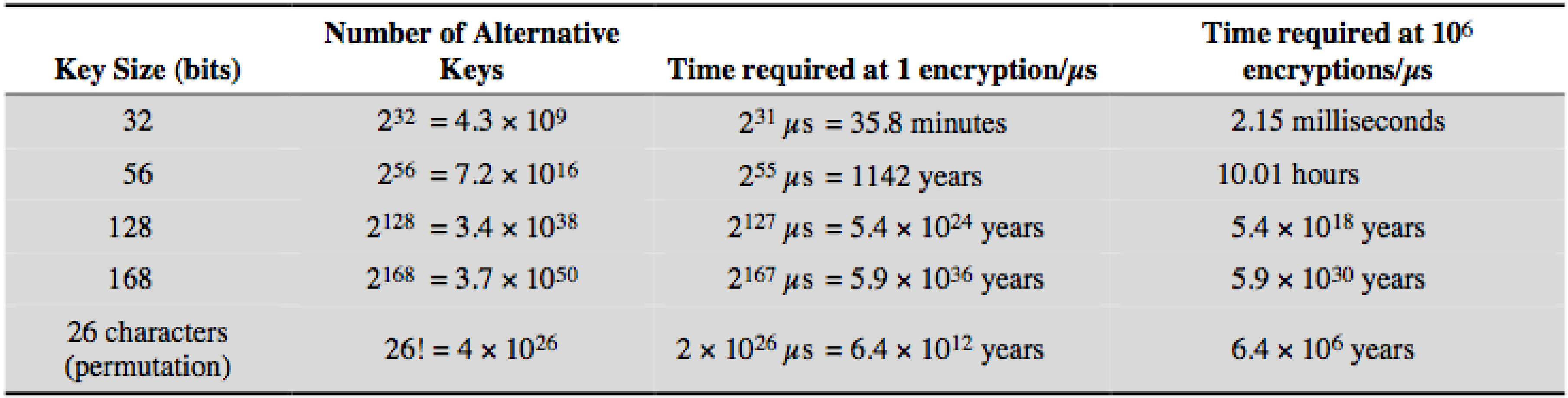
Procedure must be reversible (for decryption)

Standard Block Ciphers

- **DES: Data Encryption Standard**
  - Feistel structure: builds invertible function using non-invertible ones
  - Invented by IBM, issued as federal standard in 1977
  - 64-bit blocks, 56-bit key + 8 bits for parity

DES and 56 bit keys

- 56 bit keys are quite short



Key Size (bits)	Number of Alternative Keys	Time required at 1 encryption/μs	Time required at 10^6 encryptions/μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31}\ \mu s = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55}\ \mu s = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127}\ \mu s = 5.4 \times 10^{24}$ years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167}\ \mu s = 5.9 \times 10^{36}$ years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26}\ \mu s = 6.4 \times 10^{12}$ years	6.4×10^6 years

- 1999: EFF DES Crack + distributed machines
  - < 24 hours to find DES key
- DES ---> 3DES
  - 3DES: DES + inverse DES + DES (with 2 or 3 diff keys)

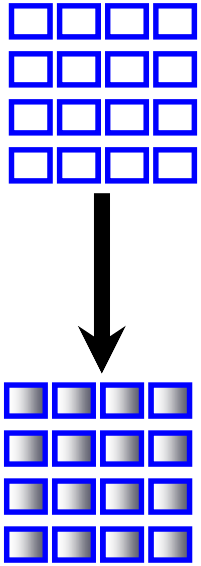
Standard Block Ciphers

- **DES: Data Encryption Standard**
  - Feistel structure: builds invertible function using non-invertible ones
  - Invented by IBM, issued as federal standard in 1977
  - 64-bit blocks, 56-bit key + 8 bits for parity

- **AES: Advanced Encryption Standard**
  - New federal standard as of 2001
    - NIST: National Institute of Standards & Technology
  - Based on the Rijndael algorithm
    - Selected via an open process
  - 128-bit blocks, keys can be 128, 192 or 256 bits

Encrypting a Large Message

- So, we have got a good block cipher, but our plaintext is larger than 128-bit block size



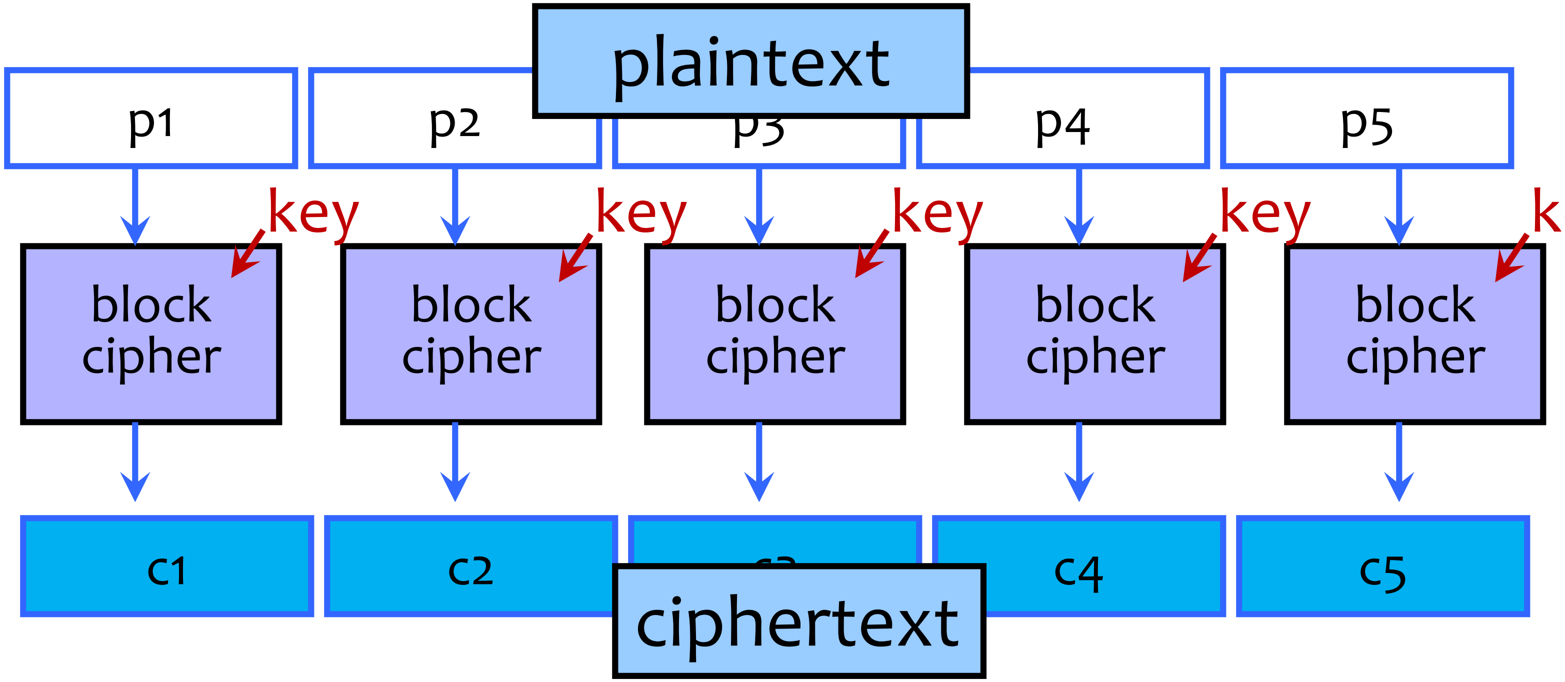
128-bit plaintext

(arranged as 4x4 array of 8-bit bytes)

128-bit ciphertext

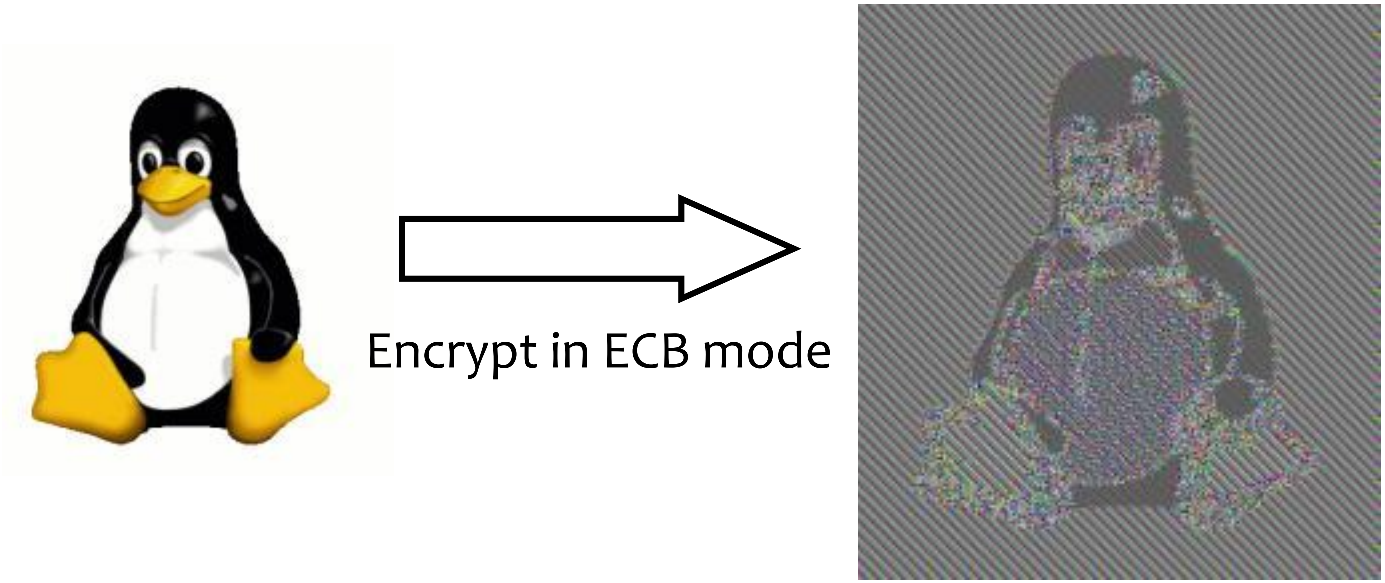
- What should we do?

Electronic Code Book (ECB) Mode



- Identical blocks of plaintext produce identical blocks of ciphertext
- No integrity checks: can mix and match blocks

Information Leakage in ECB Mode



Encrypt in ECB mode

[Wikipedia]