

CSE 484 / CSE M 584: Randomness + Symmetric Encryption

Fall 2022

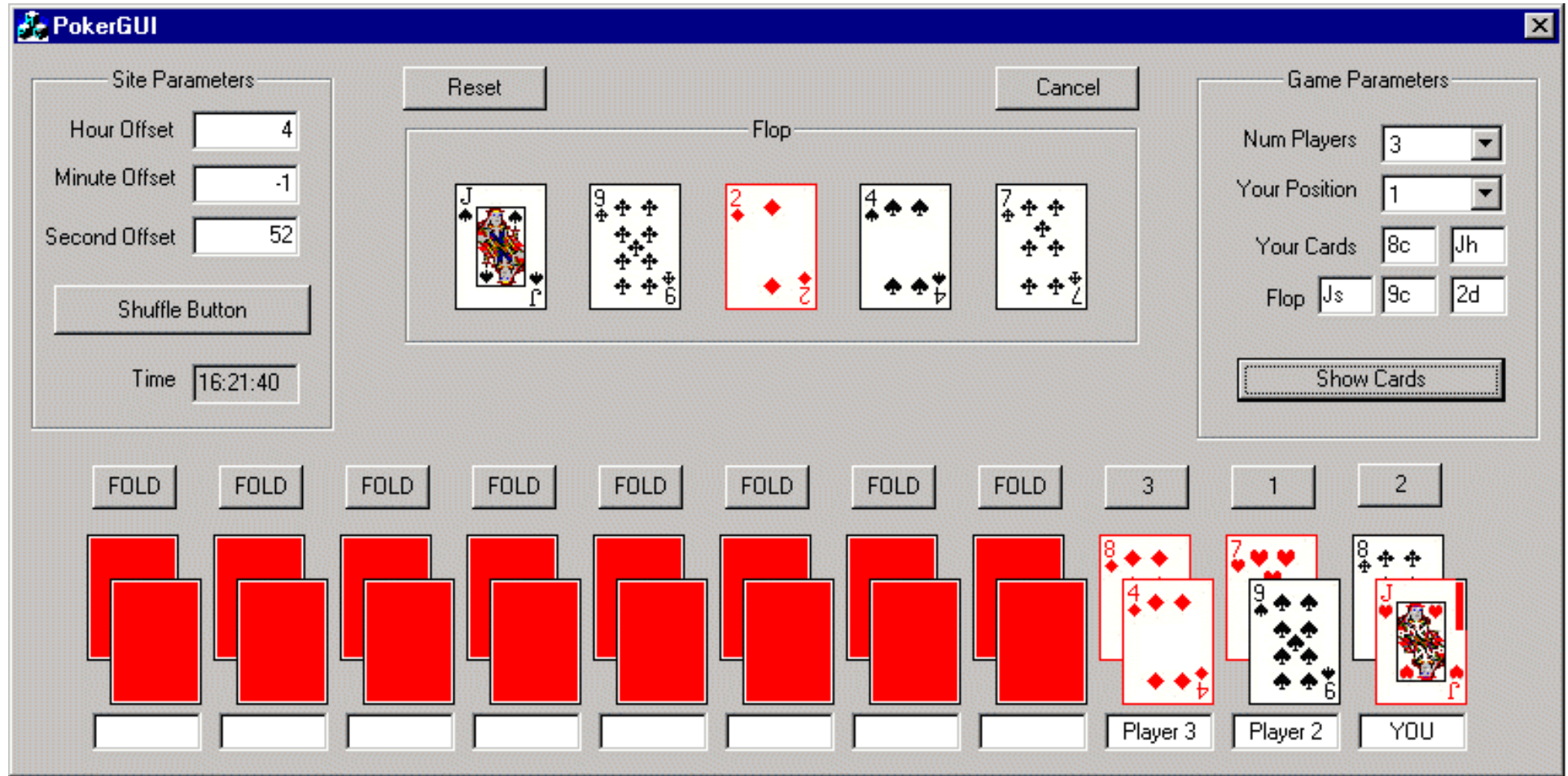
Franziska (Franzi) Roesner
franzi@cs

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

Reminder: 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



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

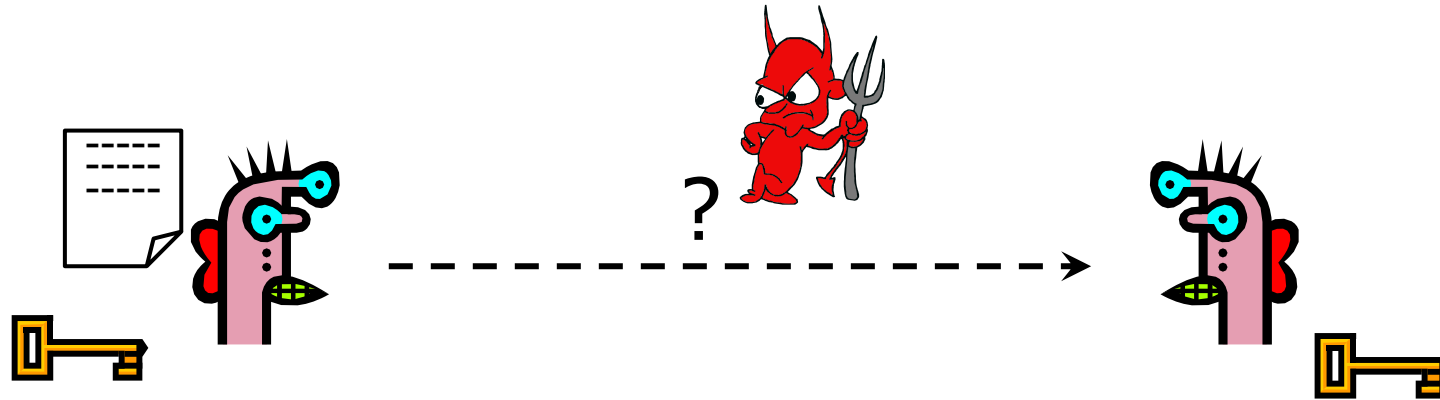
- Linux:
 - /dev/random – blocking (waits for enough entropy)
 - /dev/urandom – nonblocking, possibly less entropy
 - getrandom() – syscall! – by default, blocking
- Internally:
 - Entropy pool gathered from multiple sources
 - e.g., mouse/keyboard/network timings
- Challenges with embedded systems,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

Obtaining *Random Numbers*

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

Now: Symmetric Encryption

Confidentiality: Basic Problem

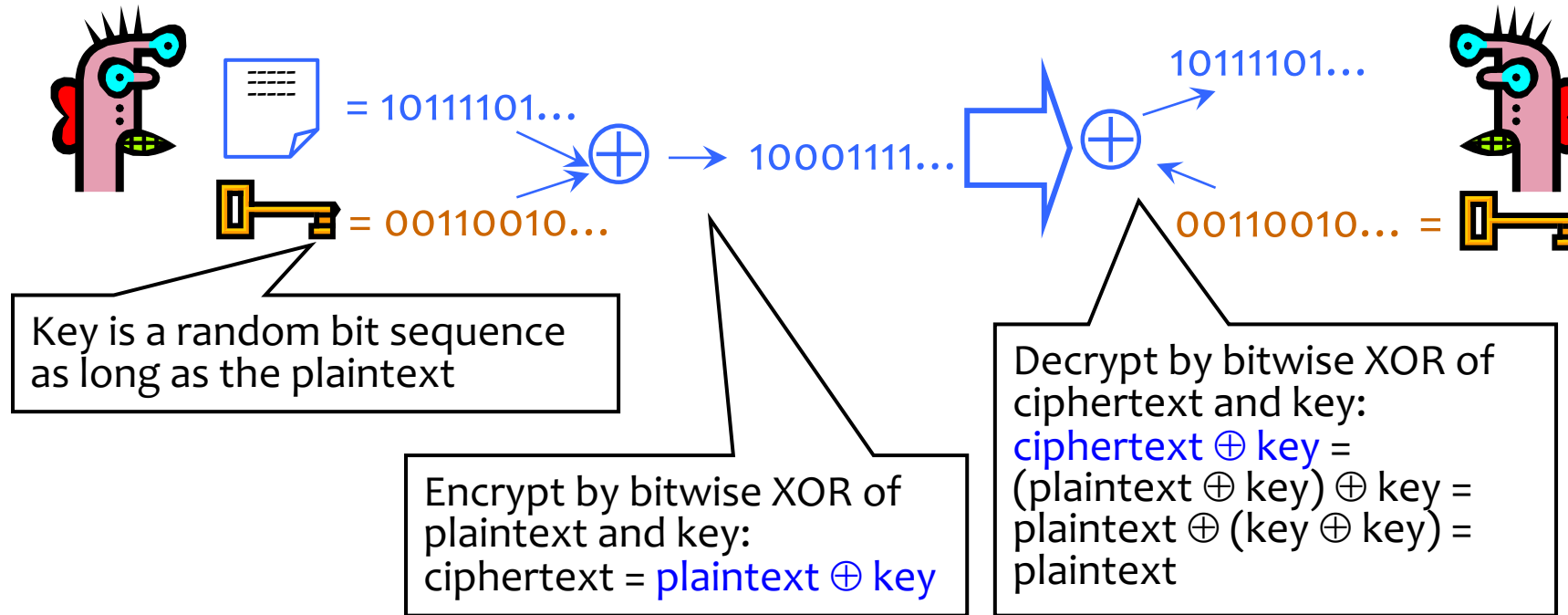


Given (Symmetric Crypto): both parties know the same **secret**.

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)

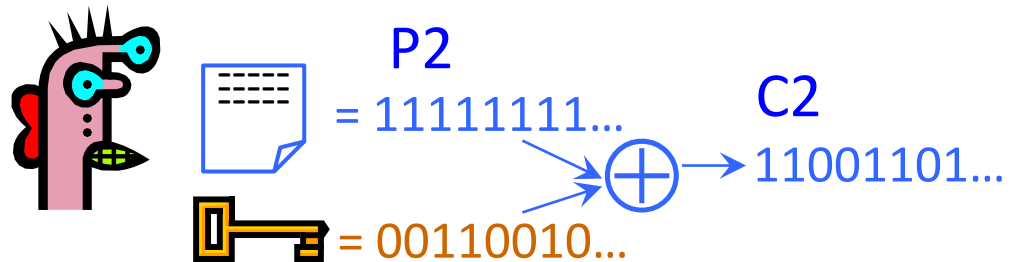
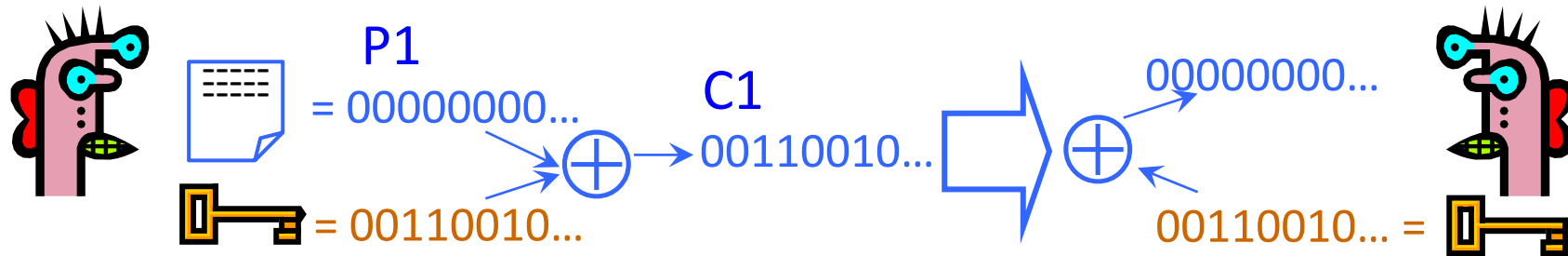
Advantages of One-Time Pad

- 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 17

Dangers of Reuse



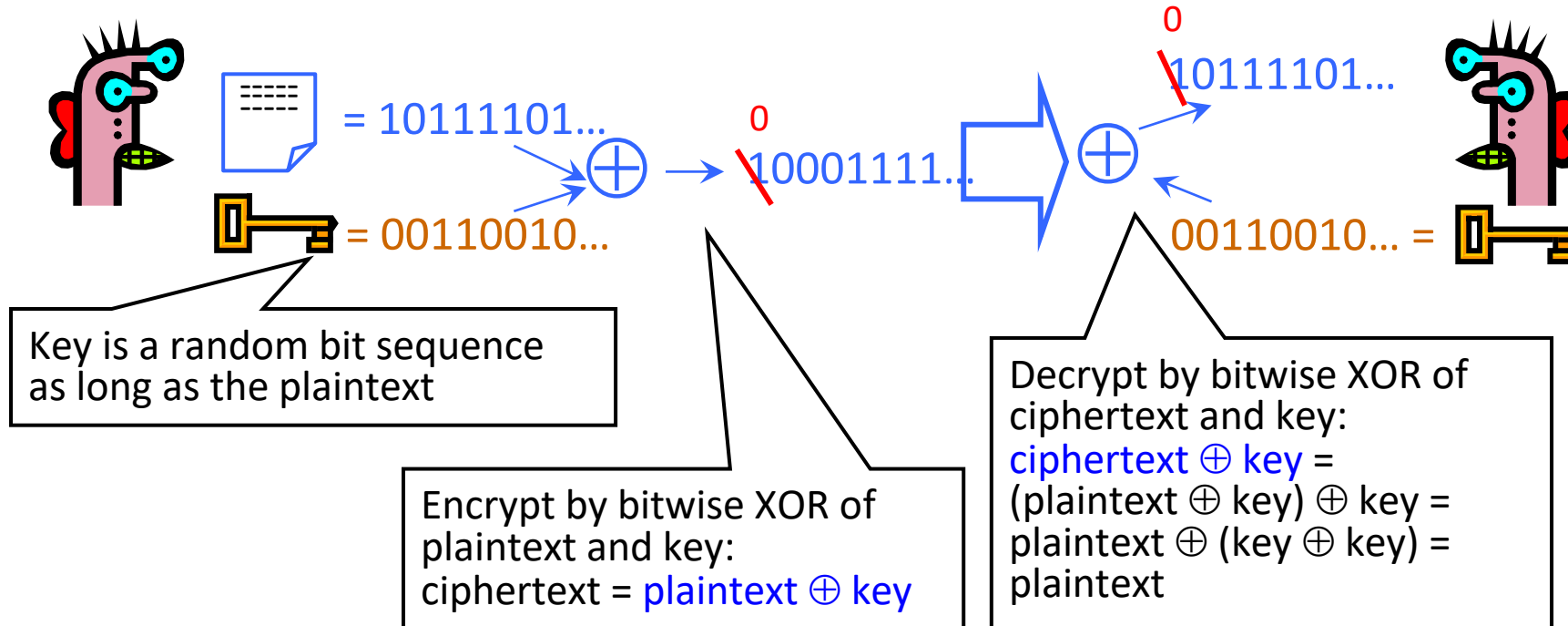
Learn relationship between plaintexts

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

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

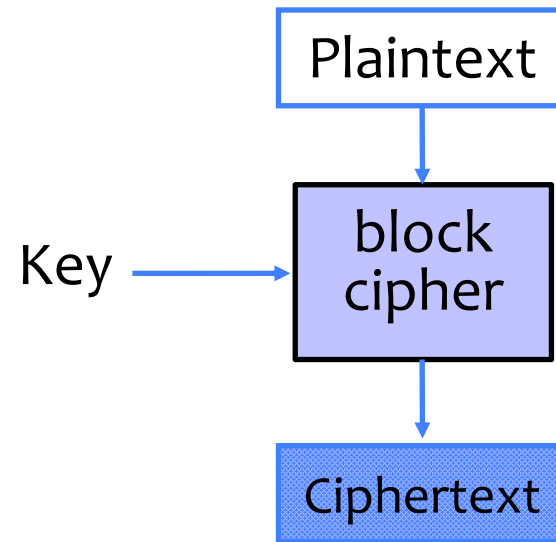
- (1) Key must be as long as the plaintext
 - Impractical in most realistic scenarios
 - Still used for diplomatic and intelligence traffic
- (2) Insecure if keys are reused
 - Attacker can obtain XOR of plaintexts
- (3) **Does not guarantee integrity**
 - One-time pad only guarantees confidentiality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

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

Key = 00 Key = 01

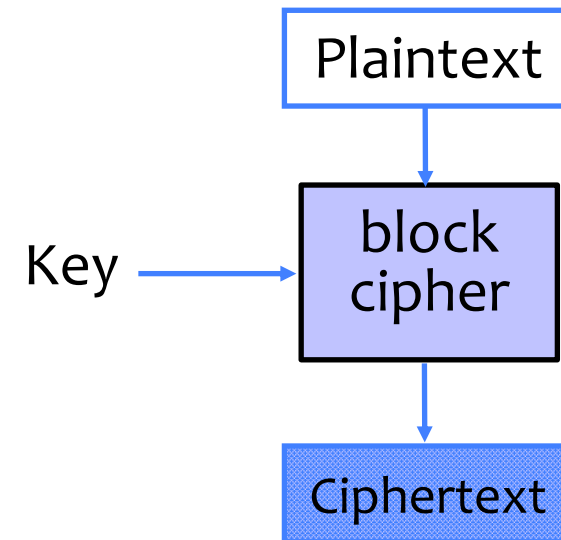
input	possible output	possible output	etc.
000	010	111	...
001	111	110	...
010	101	000	...
011	110	101	...
...
111	000	110	...

For N-bit input:
 $2^N!$ possible permutations

For K-bit key:
 2^K possible keys

Keyed Permutation

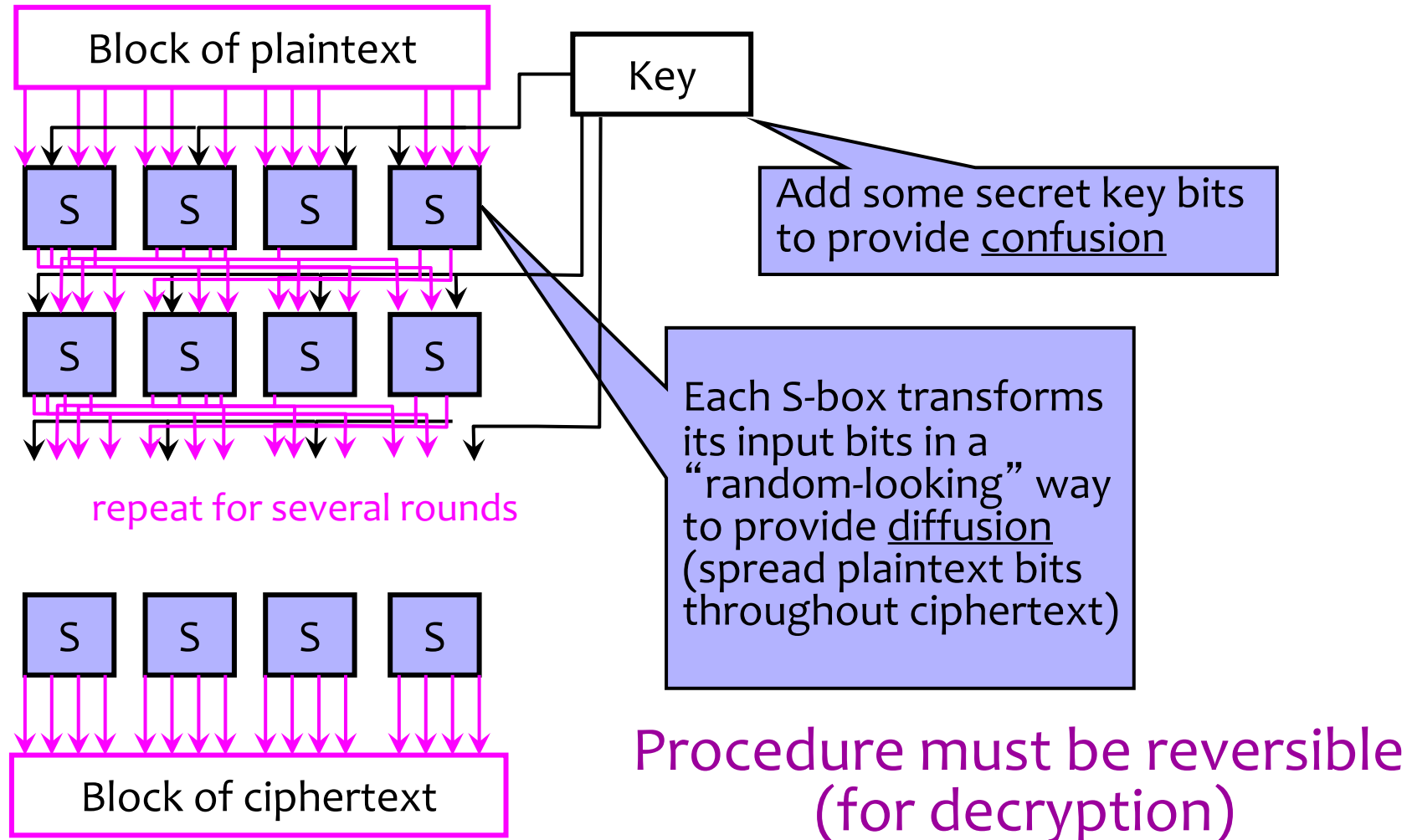
- Not just shuffling of input bits!
 - Suppose plaintext = “111”.
 - Then “111” is not the only possible ciphertext!
- Instead:
 - **Permutation of possible outputs**
 - Use secret key to pick a permutation



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

Block Cipher Operation (Simplified)



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

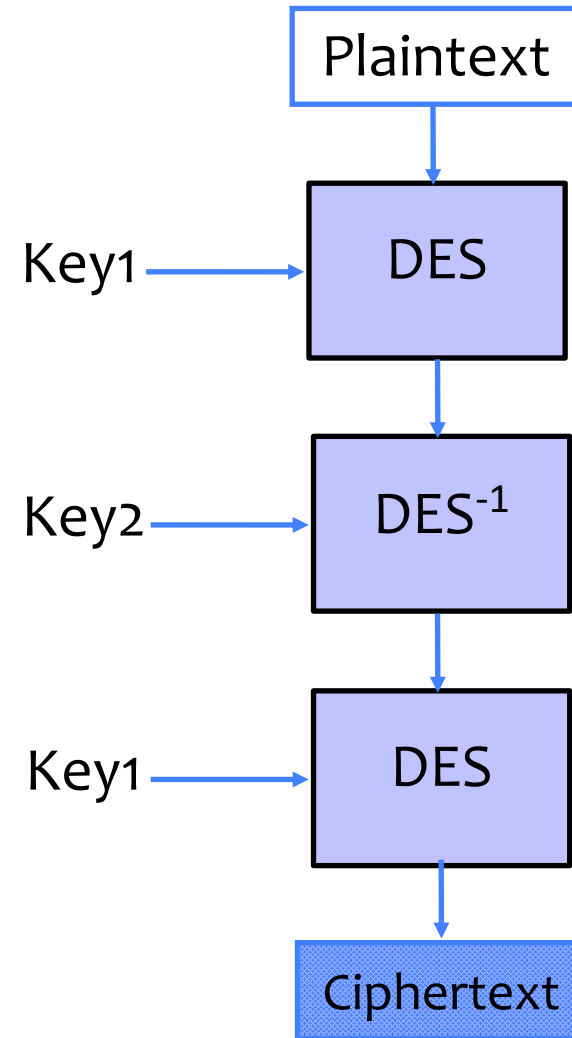
- 56 bit keys are quite short

Key Size (bits)	Number of Alternative Keys	Time required at 1 encryption/ μ s	Time required at 10^6 encryptions/ μ s
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 = 5.4×10^{24} years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167} \mu$ s = 5.9×10^{36} years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26} \mu$ s = 6.4×10^{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)

3DES

- Two-key 3DES increases security of DES by doubling the key length
- Why not 2DES?
 - Minimal gain in security compared to DES due to **meet-in-the-middle attack**

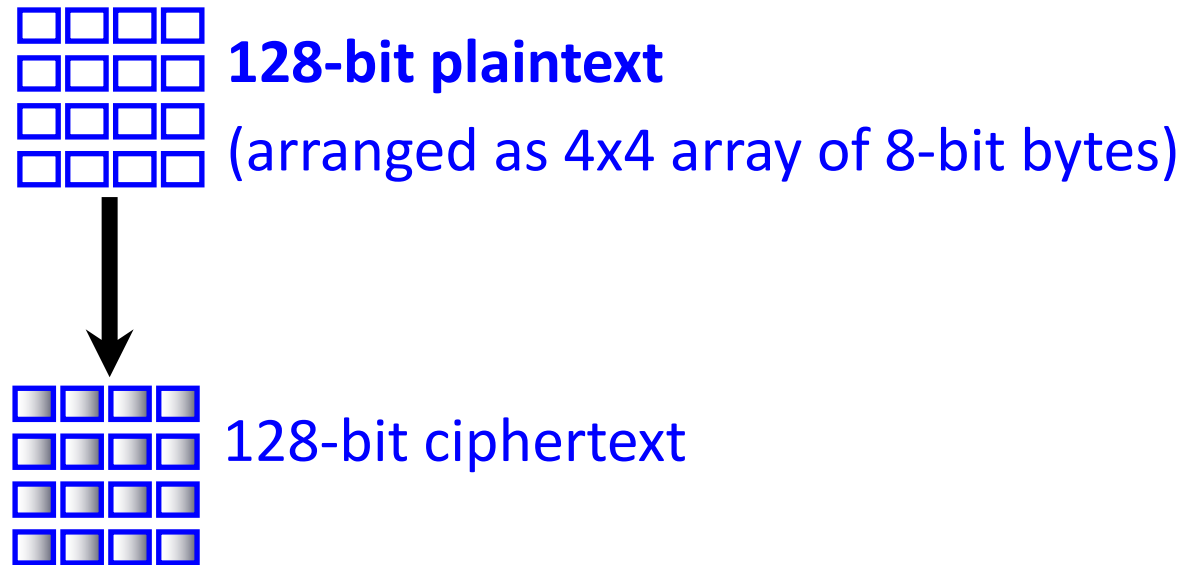


Standard Block Ciphers

- **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've got a good block cipher, but our plaintext is larger than 128-bit block size



- What should we do?