CSEP 564: Computer Security and Privacy

Software Security [Wrap-Up] Cryptography

Fall 2022

David Kohlbrenner

dkohlbre@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 1a: Oct 21
  - That is, sploits 1-3
  - Make sure to use the handin script!

Timing Attacks

- Assume there are no “typical” bugs in the software
  - No buffer overflow bugs
  - No format string vulnerabilities
  - Good choice of randomness
  - Good design
- The software may still be vulnerable to timing attacks
  - Software exhibits input-dependent timings
- Complex and hard to fully protect against

Hey what about if its over a network?

- “Remote timing attacks are practical” - 2005
  - David Brumley, Dan Boneh

Other Examples

- Plenty of other examples of timings attacks
  - Timing cache misses
    - Extract cryptographic keys…
    - Spectre/Meltdown attacks
  - Duration of a rendering operation
    - Extract webpage information
  - Duration of a *failed* decryption attempt
    - Different failures mean different thing (e.g., Padding oracles)

Side-channels

- Timing is only one possibility
- Consider:
  - Power usage
  - Audio
  - EM Outputs

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)

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: “crypto”

- For this course: crypto means “cryptography”

“If you think cryptography will solve your problem, you don't understand cryptography and you don't understand your problem”

- A cryptographer (its complicated)

“If you think cryptography will solve your problem, you don't understand cryptography and you don't understand your problem”

- A cryptographer (its complicated)

Probably either wJim Morris or Lampson or Needham

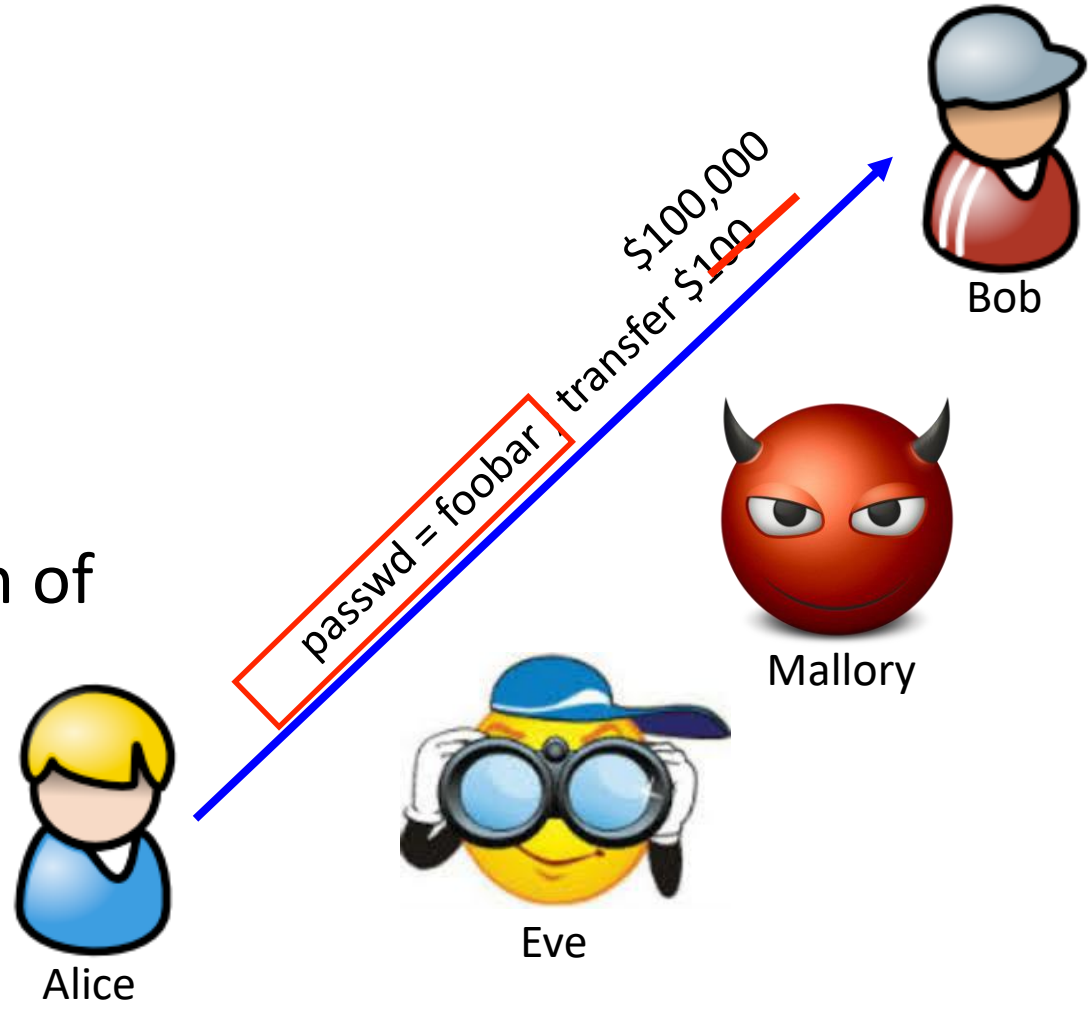
Common Communication Security Goals

Privacy of data:

Prevent exposure of information

Integrity of data:

Prevent modification of information

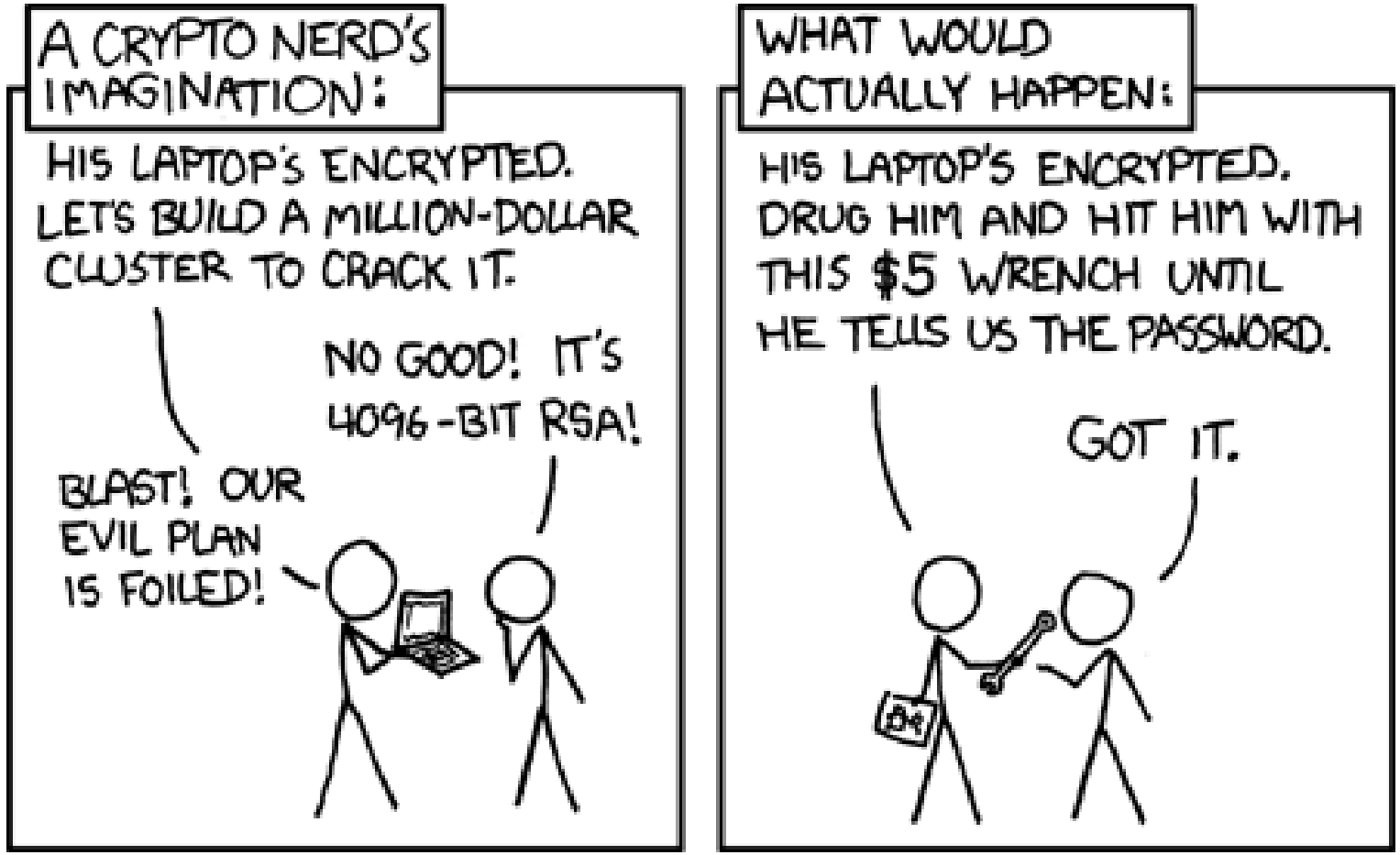


Recall Bigger Picture

- 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

Rubber-hose cryptanalysis: http://xkcd.com/538/



Paper Discussion Time!

"An Empirical Study of Cryptographic Misuse in Android Applications"

Manuel Egele, David Brumley, Yanick Fratantonio, Christopher Kruegel (2013)

- Pick one of these and ask about it/describe it to your neighbor briefly
  - Any of the "six rules" for cryptography
  - PBE – Password-Based Encryption
  - Any of the specific broken usages they found
  - Building better cryptographic APIs
  - Something else from the paper

Paper Discussion Time!

“An Empirical Study of Cryptographic Misuse in Android Applications“

Manuel Egele, David Brumley, Yanick Fratantonio, Christopher Kruegel

(2013)

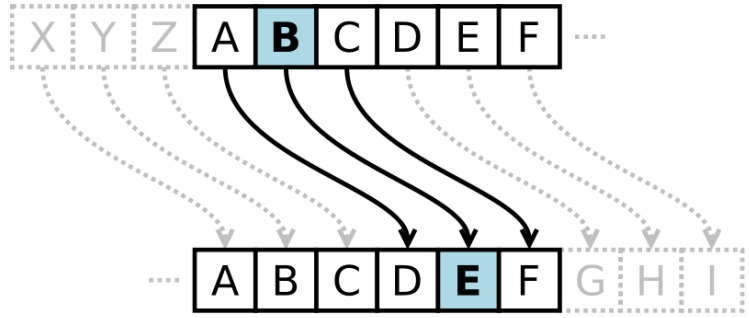
- What is something you learned from this paper?

History of cryptography

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

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)

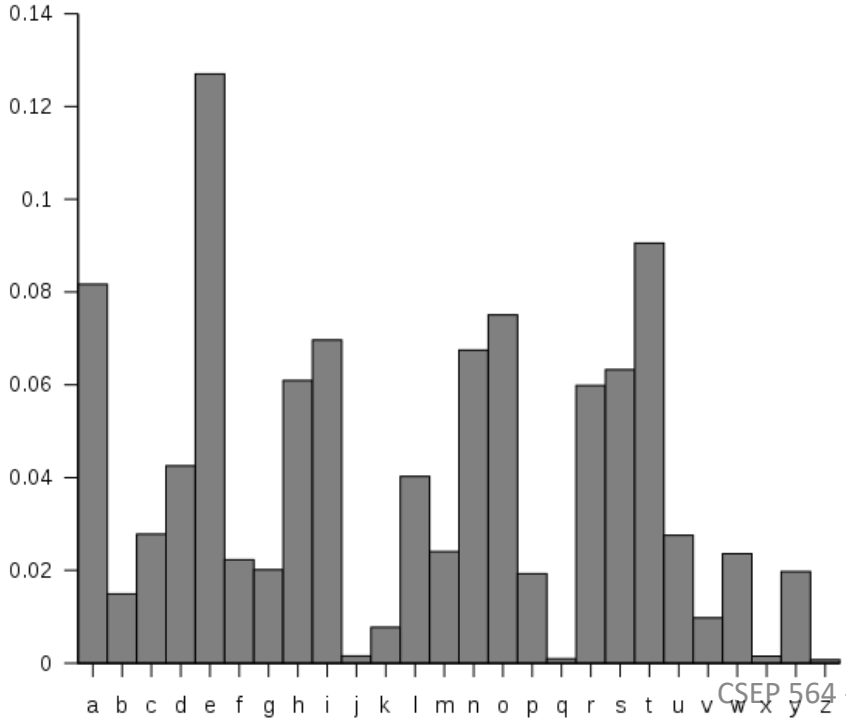
- ROT13: shift 13 (encryption and decryption are symmetric)
- What is the key space?
  - 26 possible shifts.
- How to attack shift ciphers?
  - Brute force.

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:

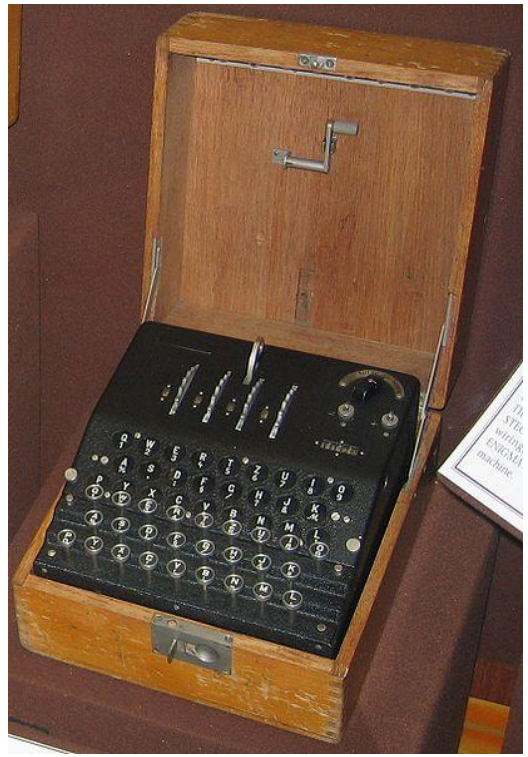
th 1.52%	en 0.55%	ng 0.18%
he 1.28%	ed 0.53%	of 0.16%
in 0.94%	to 0.52%	al 0.09%
er 0.94%	it 0.50%	de 0.09%
an 0.82%	ou 0.50%	se 0.08%
re 0.68%	ea 0.47%	le 0.08%
nd 0.63%	hi 0.46%	sa 0.06%
at 0.59%	is 0.46%	si 0.05%
on 0.57%	or 0.43%	ar 0.04%
nt 0.56%	ti 0.34%	ve 0.04%
ha 0.56%	as 0.33%	ra 0.04%
es 0.56%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

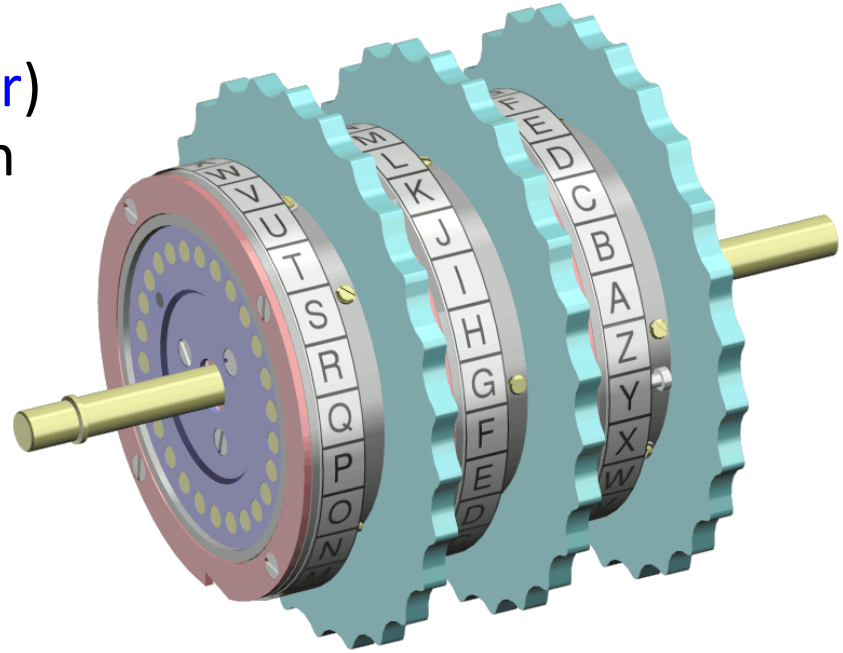
Trigrams:

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)*

- Don’t go 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 / tc
- 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

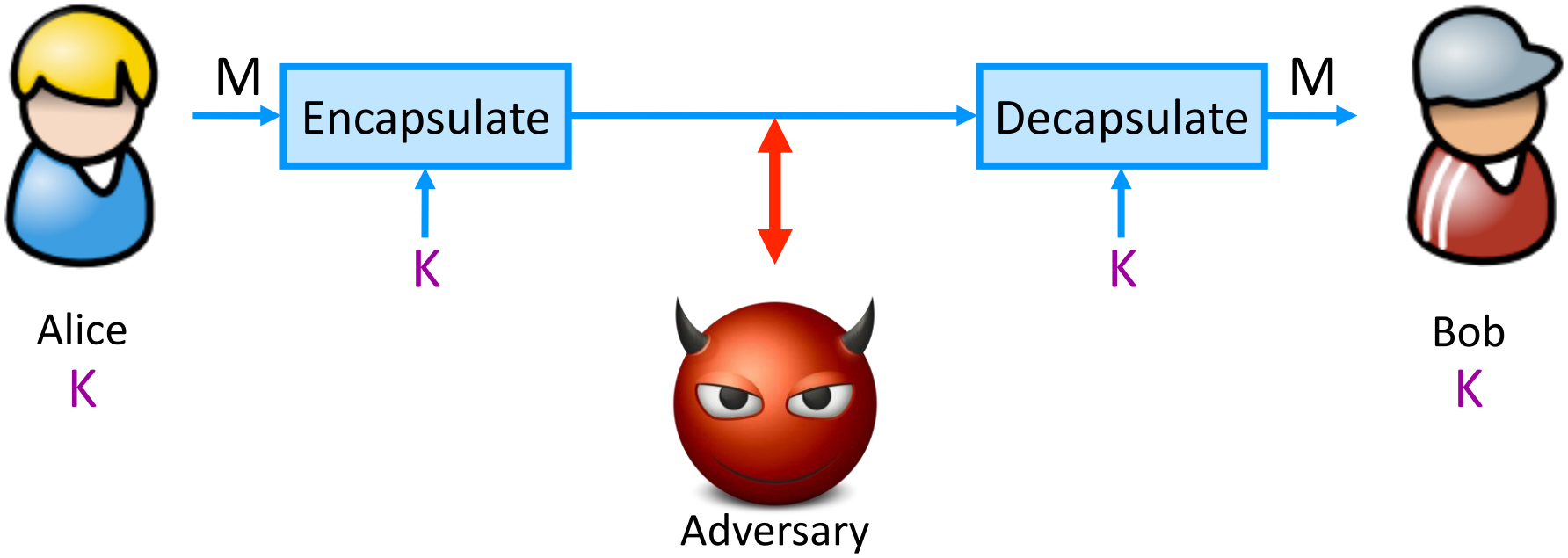
Flavors of Cryptography

- 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* ☺

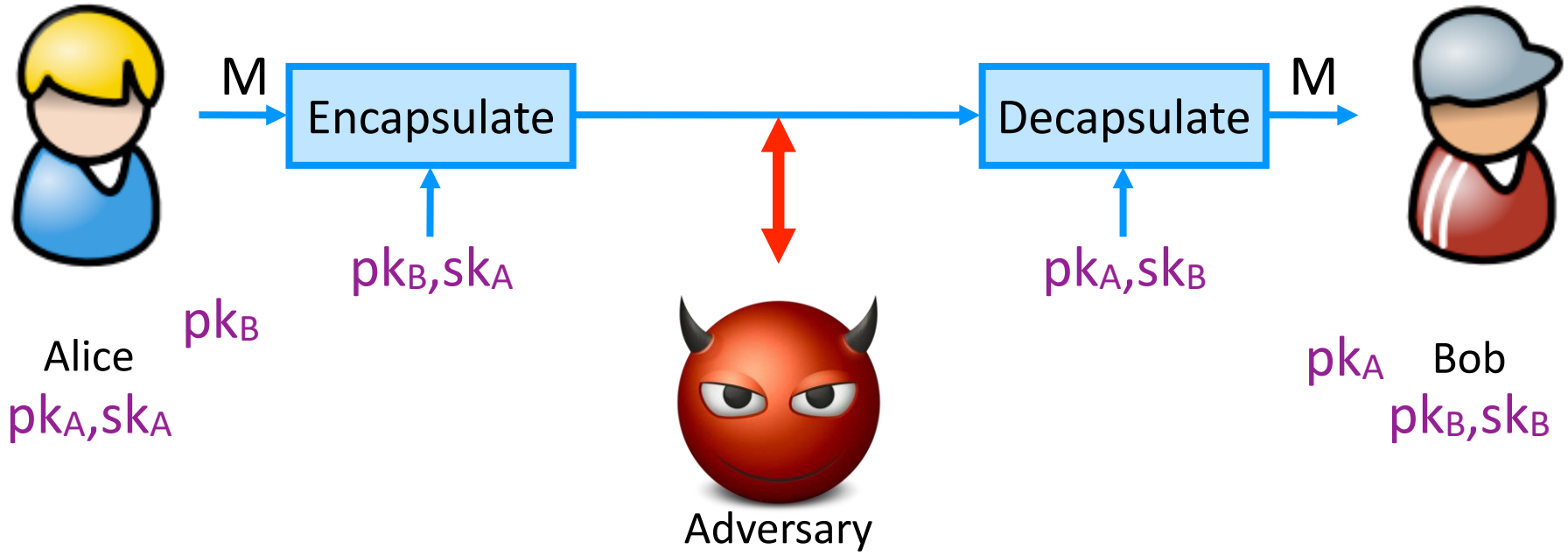
Symmetric Setting

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.



Public keys, Private keys, Secret keys…

- Secret key
  - The single key used in symmetric encryption
  - The non-public key in asymmetric
- Private keys
  - The non-public key in asymmetric
- Public key
  - The… public key in asymmetric
- Key
  - Generally means private/secret

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

Detour: Randomness

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)

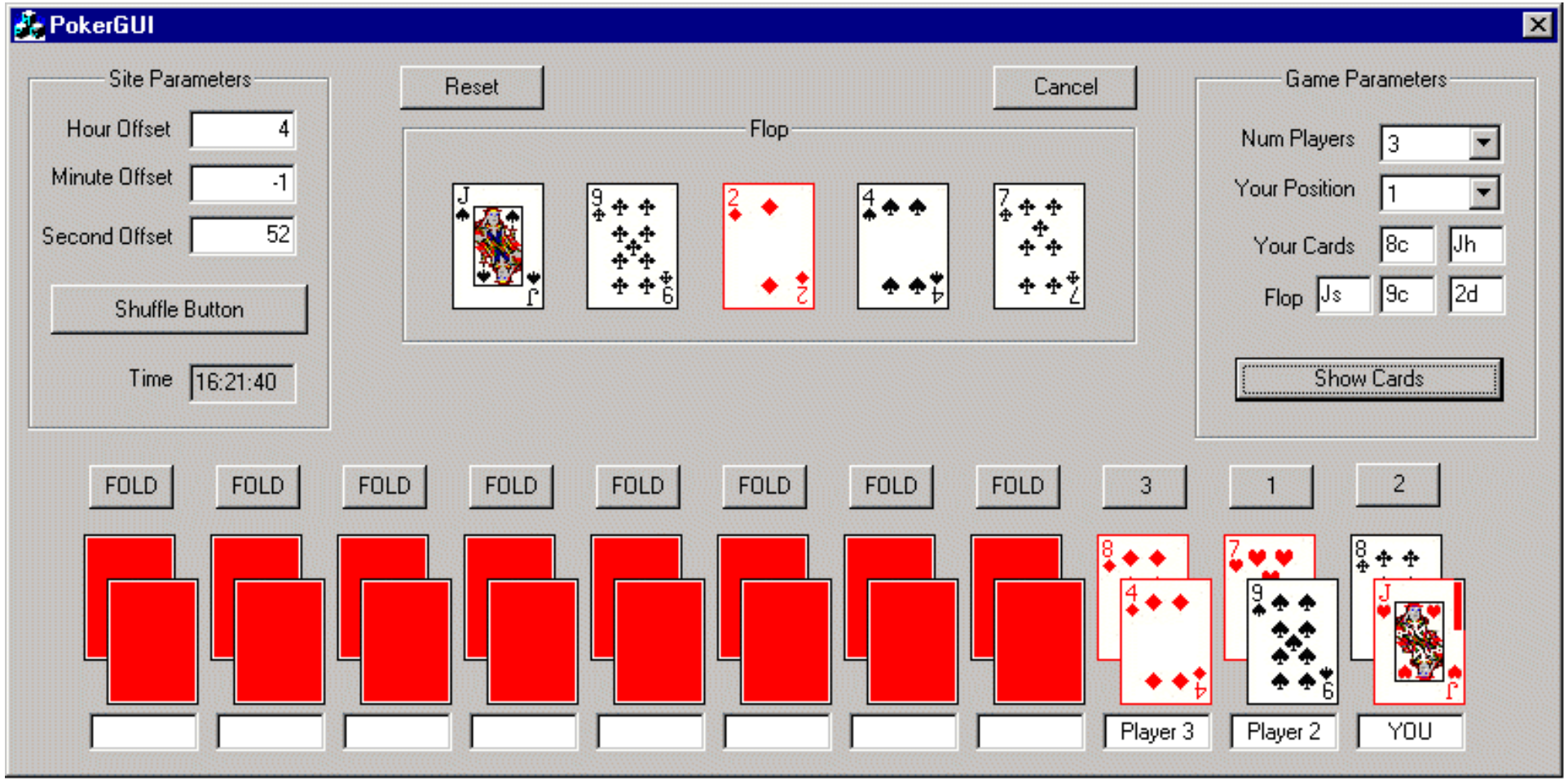
C's rand() Function

- 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;
}
```

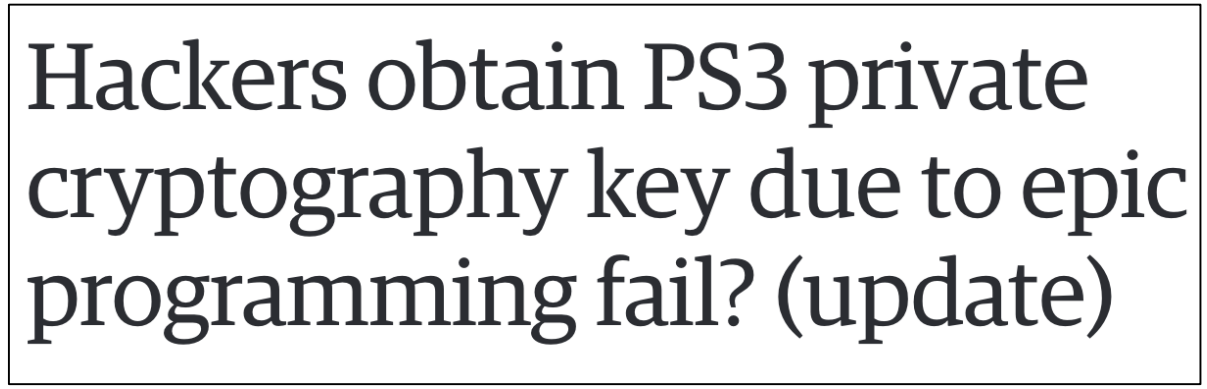
- Problem: don't use rand() for security-critical applications!
  - Given a few sample outputs, you can predict subsequent ones


A World of Action!
Help
Suggestions
Fold in Turn
$0
jonnyboy
brute
mamajoe
RickG
Fold
Check
Bet ($5)
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

A recent example: keypair

https://securitylab.github.com/advisories/GHSL-2021-1012-keypair/

- keypair is a JS library for generating (asymmetric) keypairs

The output from the Lehmer LCG is encoded incorrectly. The specific line with the flaw is:

```
b.putByte(String.fromCharCode(next & 0xFF))
```

The definition of putByte is

```
[…]putByte = function(b) { this.data += String.fromCharCode(b); };
```

Since we are masking with 0xFF, we can determine that 97% of the output from the LCG are converted to zeros. The only outputs that result in meaningful values are outputs 48 through 57, inclusive.

The impact is that each byte in the RNG seed has a 97% chance of being 0 due to incorrect conversion. When it is not, the bytes are 0 through 9.

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

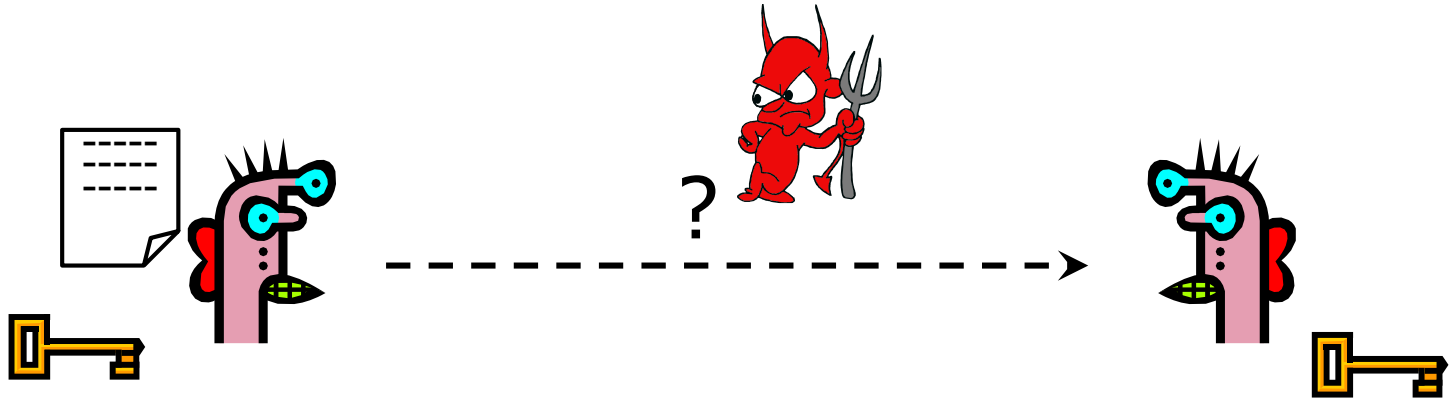
- Linux:
  - /dev/random – blocking (waits for enough entropy)
  - /dev/urandom – nonblocking, possibly less entropy
  - getrandom() – syscall! – by default, blocking
- Internally:
  - Entropy pool gathered from multiple sources
    - e.g., mouse/keyboard/network timings
- Challenges with embedded systems,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

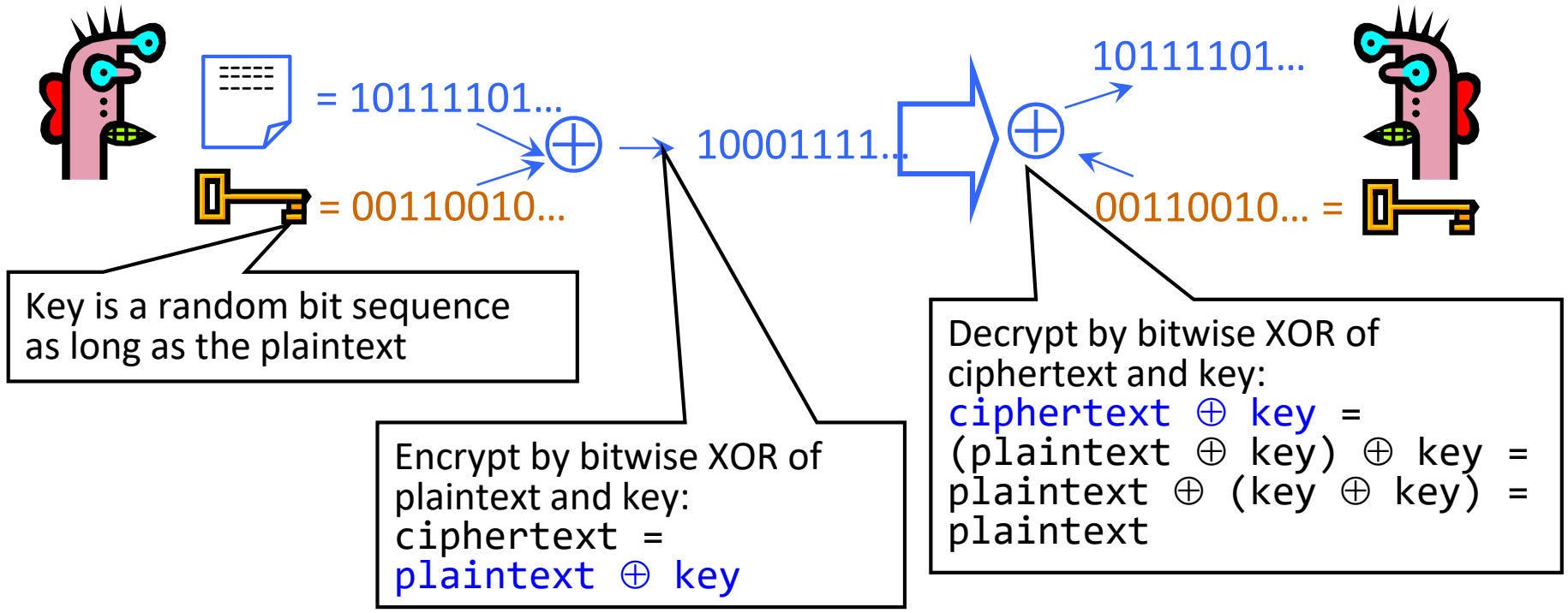
Goal: send a message confidentially.

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)

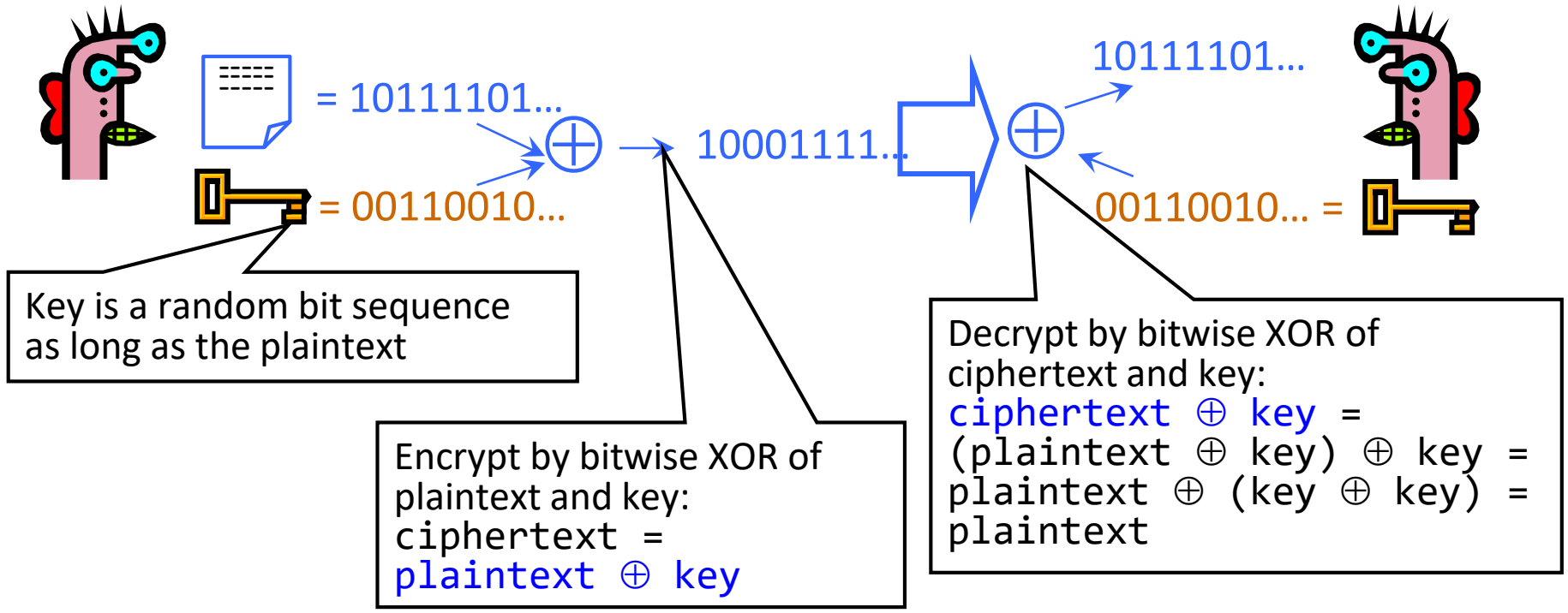
Advantages of One-Time Pad

- 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?

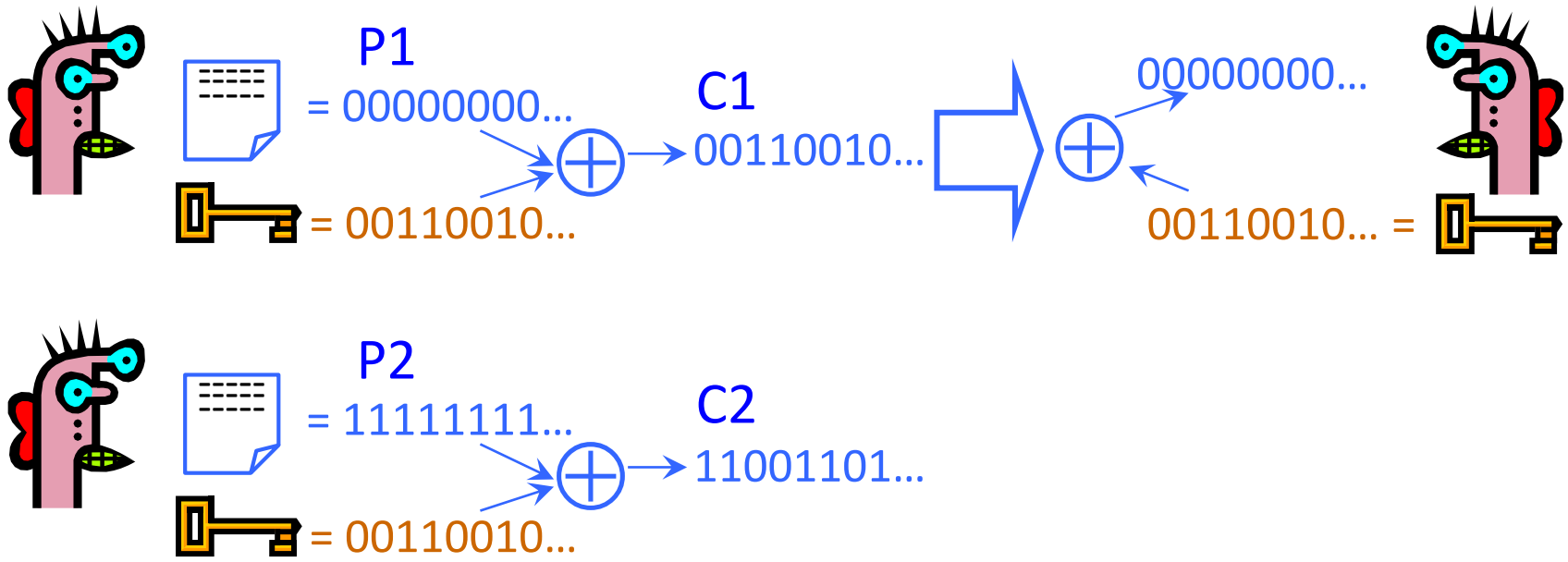
- Discuss and pollev
- What potential security problems do you see with the one-time pad?
- (Try not to look ahead and next slides)
- Recall two key goals of cryptography: confidentiality and integrity

One-Time Pad - Reminder



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)

Dangers of Reuse



Learn relationship between plaintexts

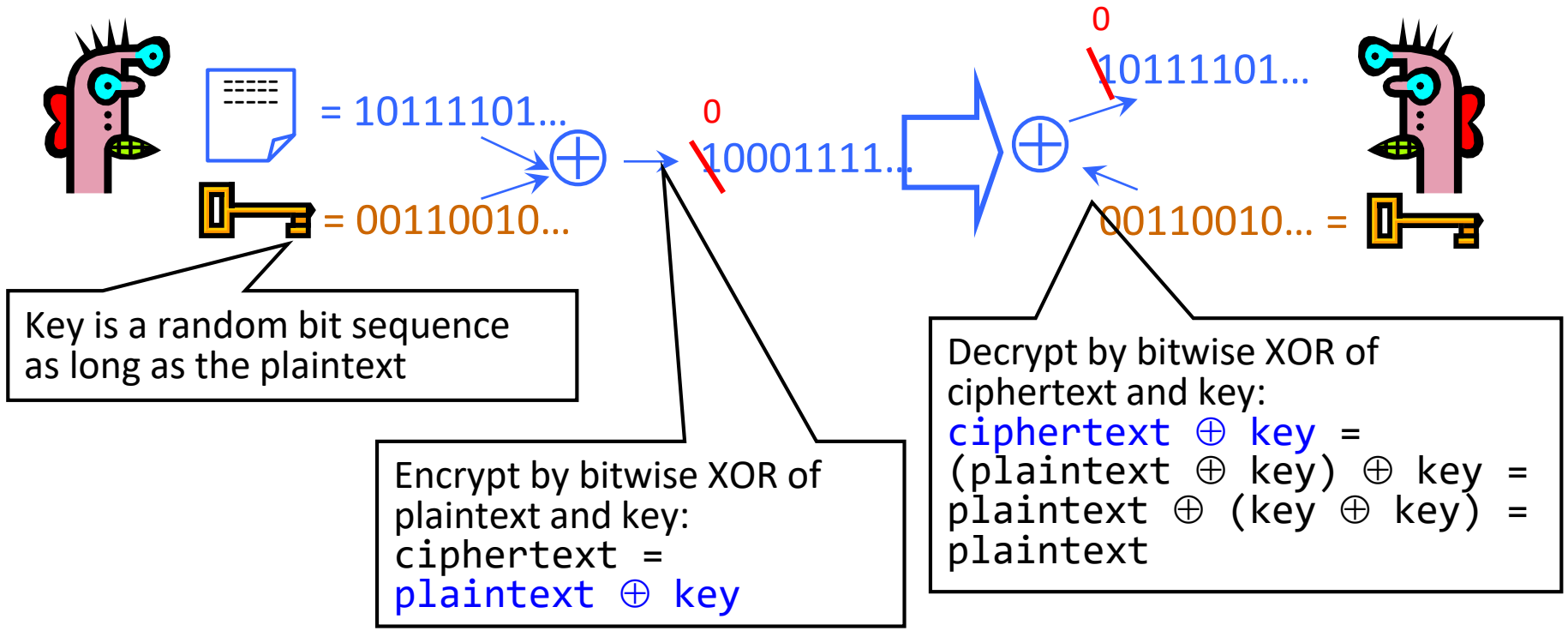
C1⊕C2 = (P1⊕K)⊕(P2⊕K) =

(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

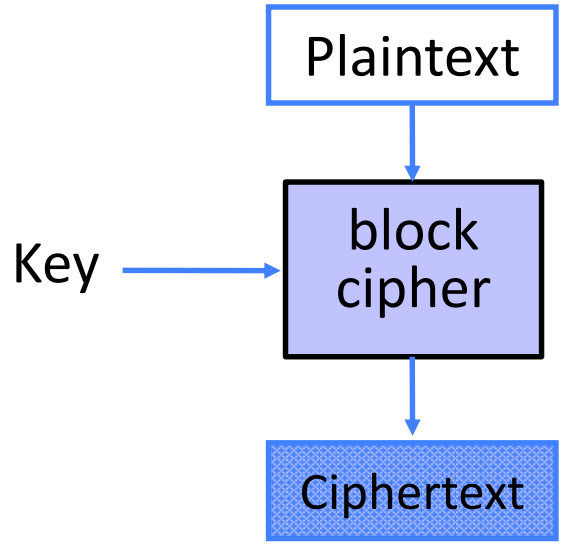
- What to do when it is infeasible to pre-share huge random keys?
  - When one-time pad is unrealistic…
- Use special cryptographic primitives: block ciphers, stream ciphers
  - Single key can be re-used (with some restrictions)
  - Not as theoretically secure as one-time pad

What if we try something simple?

- Alice and Bob synchronize their clocks perfectly, then generate OTPs
- Hash(time)
- Hash(time, key)

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

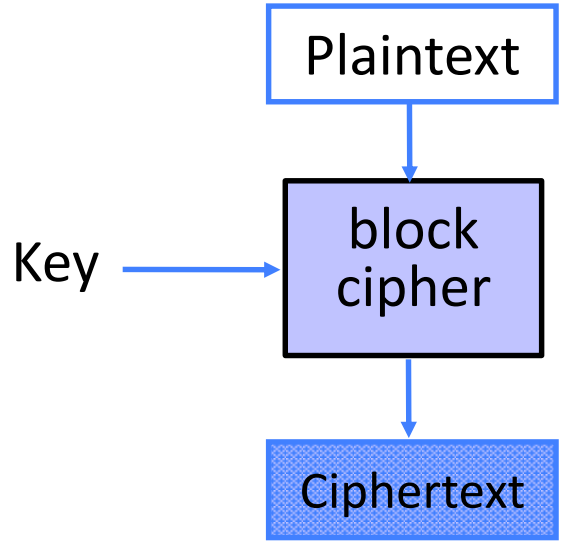
input	possible output	possible output	etc.
000	010	111	…
001	111	110	…
010	101	000	…
011	110	101	…
…	…		…
111	000	110	…

Key = 00
Key = 01

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

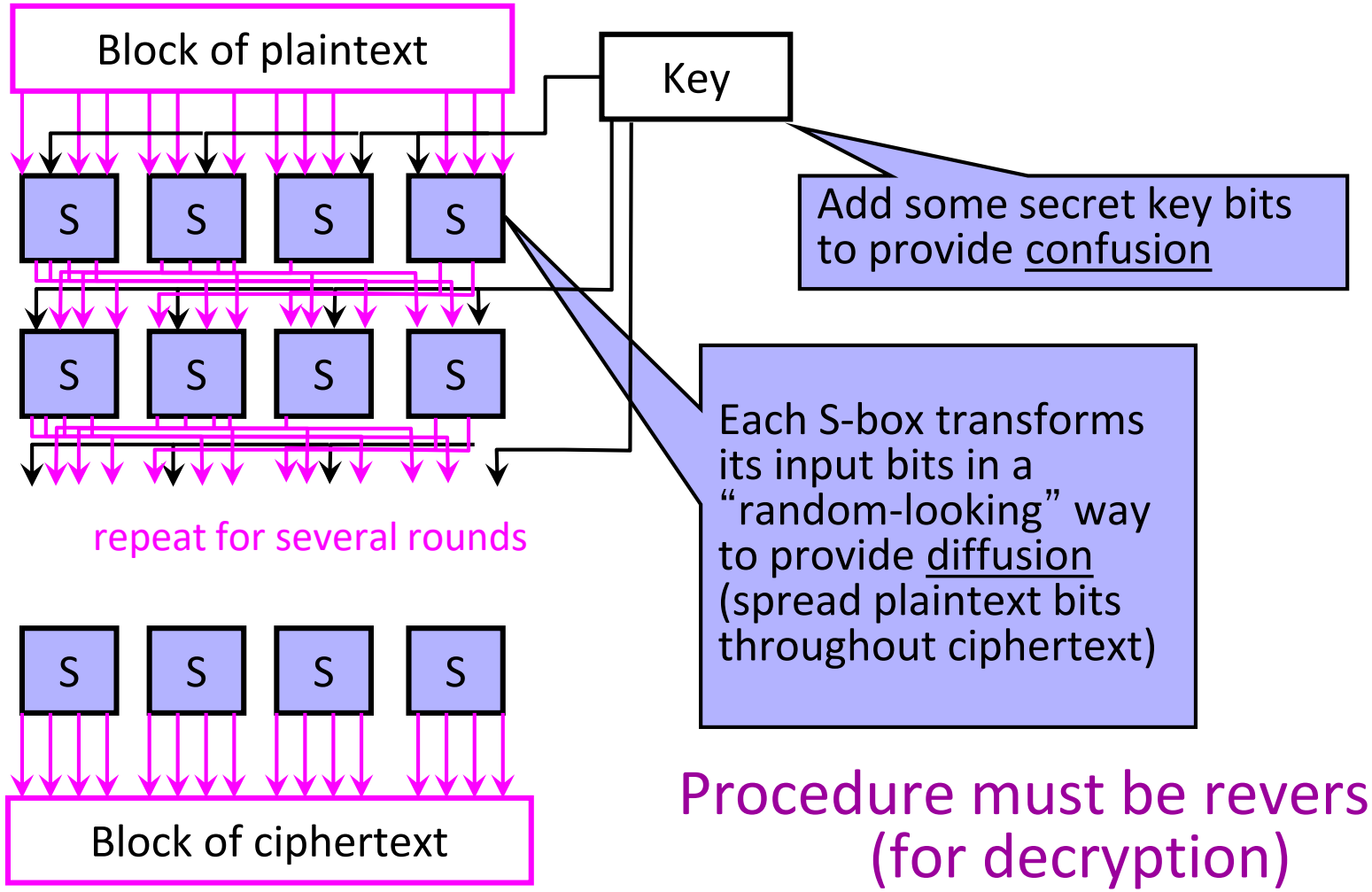


Block Cipher Security

- 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)



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

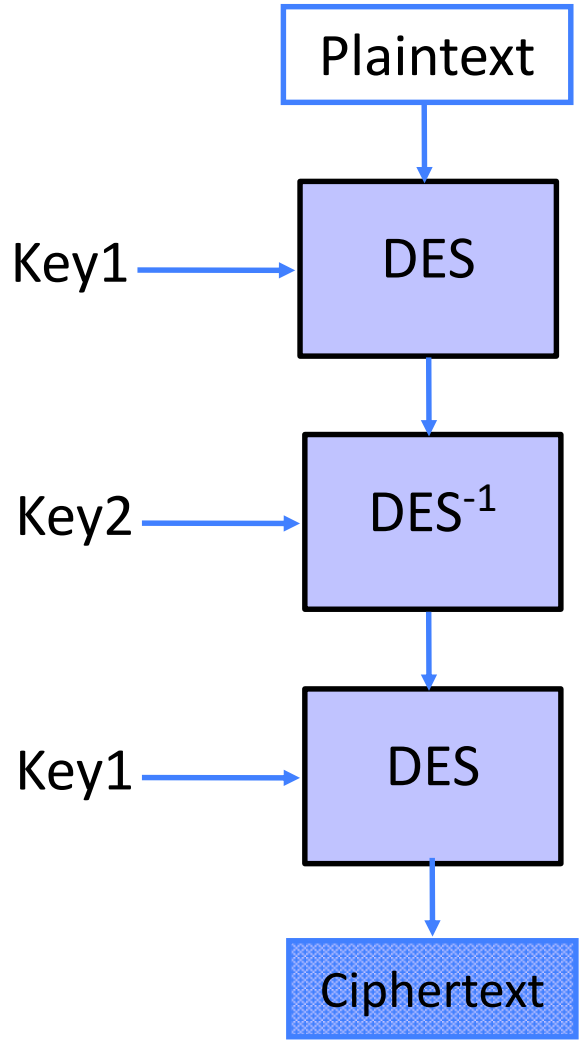
- 56 bit keys are quite short

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)

3DES

- Two-key 3DES increases security of DES by doubling the key length



But wait… what about *2DES*?

- Suppose you are given plaintext-ciphertext pairs (P1,C1), (P2,C2), (P3,C3)
- Suppose Key1 and Key2 are each 56-bits long
- Can you figure out Key1 and Key2 if you try all possible values for both (2^{112} possibilities) → Yes
- Can you figure out Key1 and Key2 more efficiently than that? → Discuss!