CSE 484 / CSE M 584 (Spring 2012)

# Intro to Cryptography

Tadayoshi Kohno

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

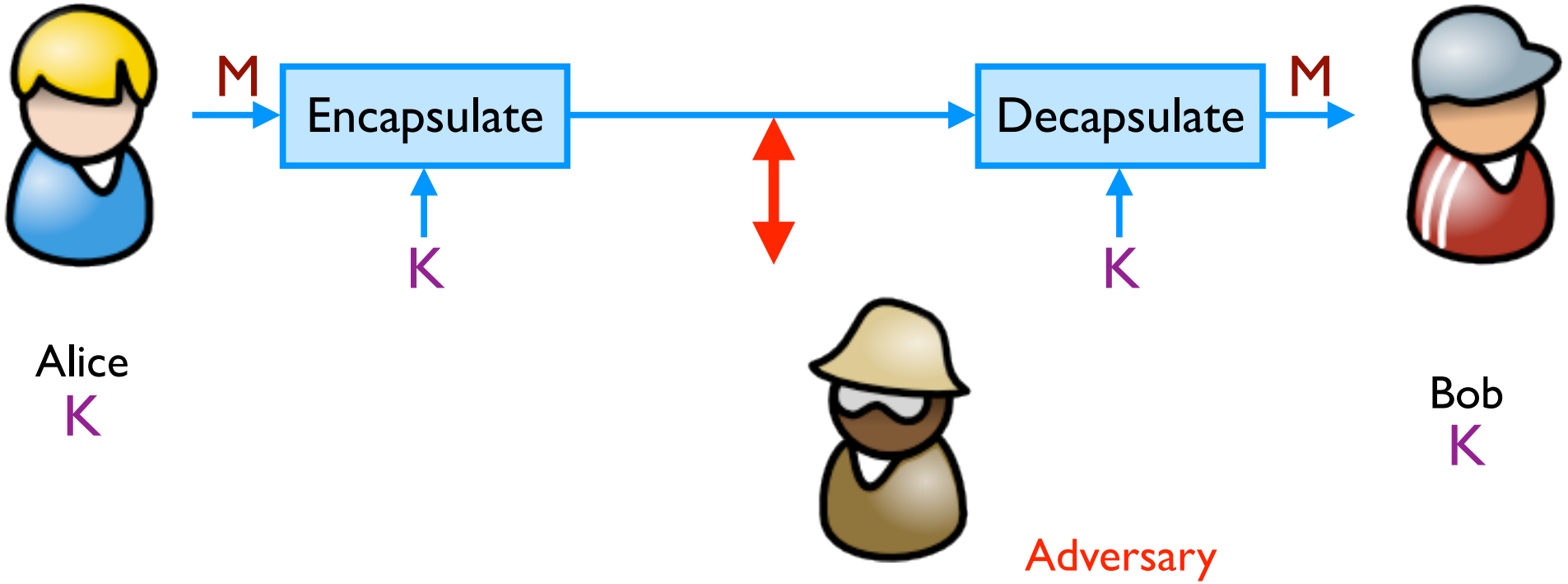
# Goals for Today

- Cryptography

- Also:  Lab part 1 due on Friday
  - Don’t all increase in complexity
  - Read recommended readings

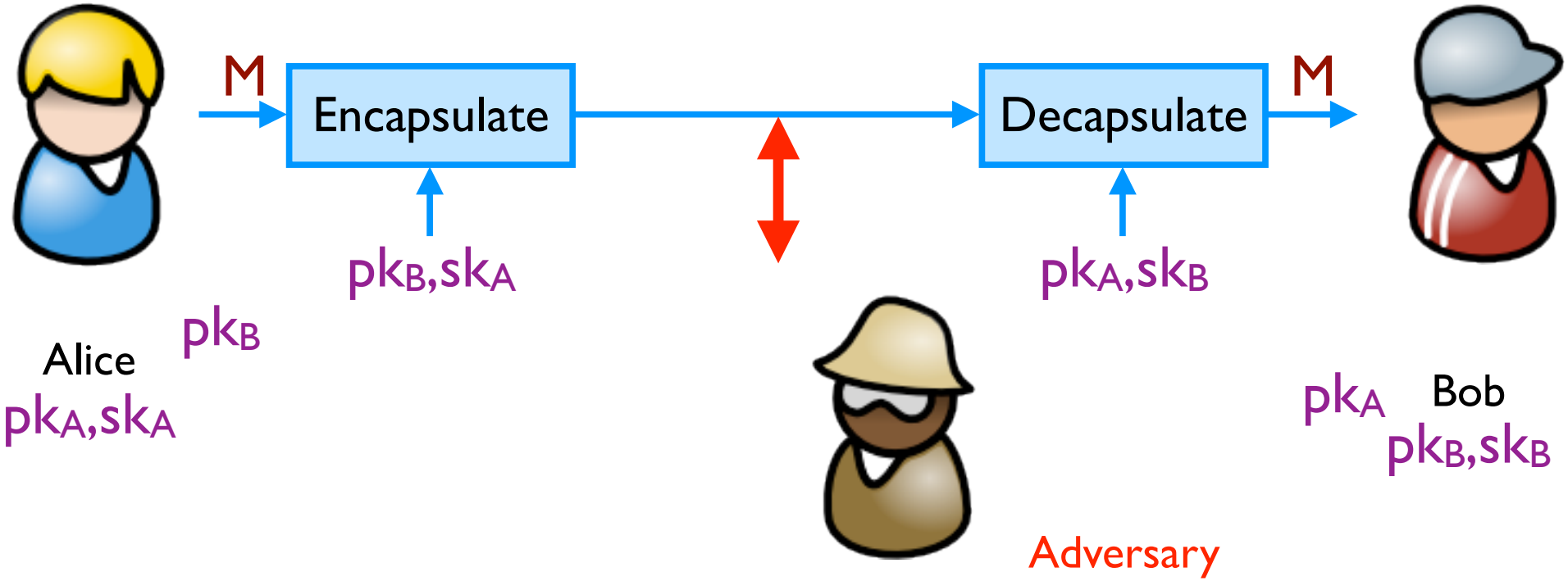
# Symmetric Setting

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



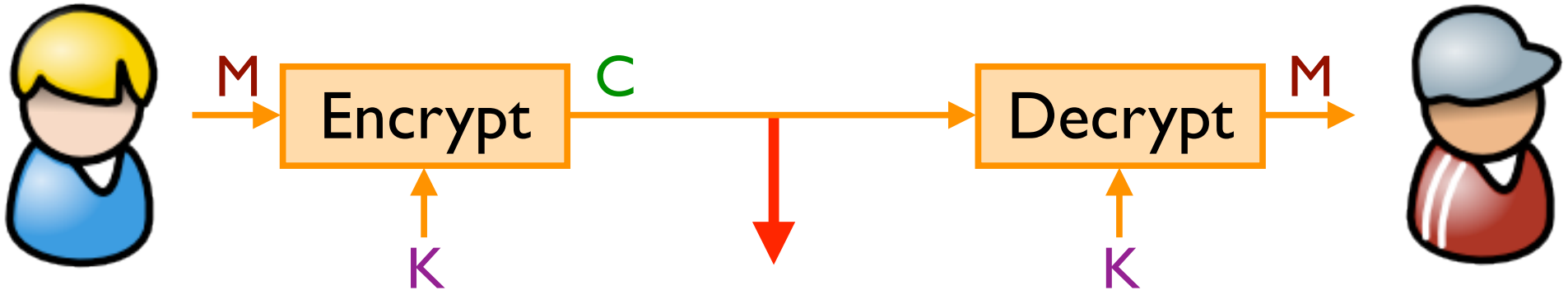
# Asymmetric Setting

Each party creates a public key pk and a secret key sk.



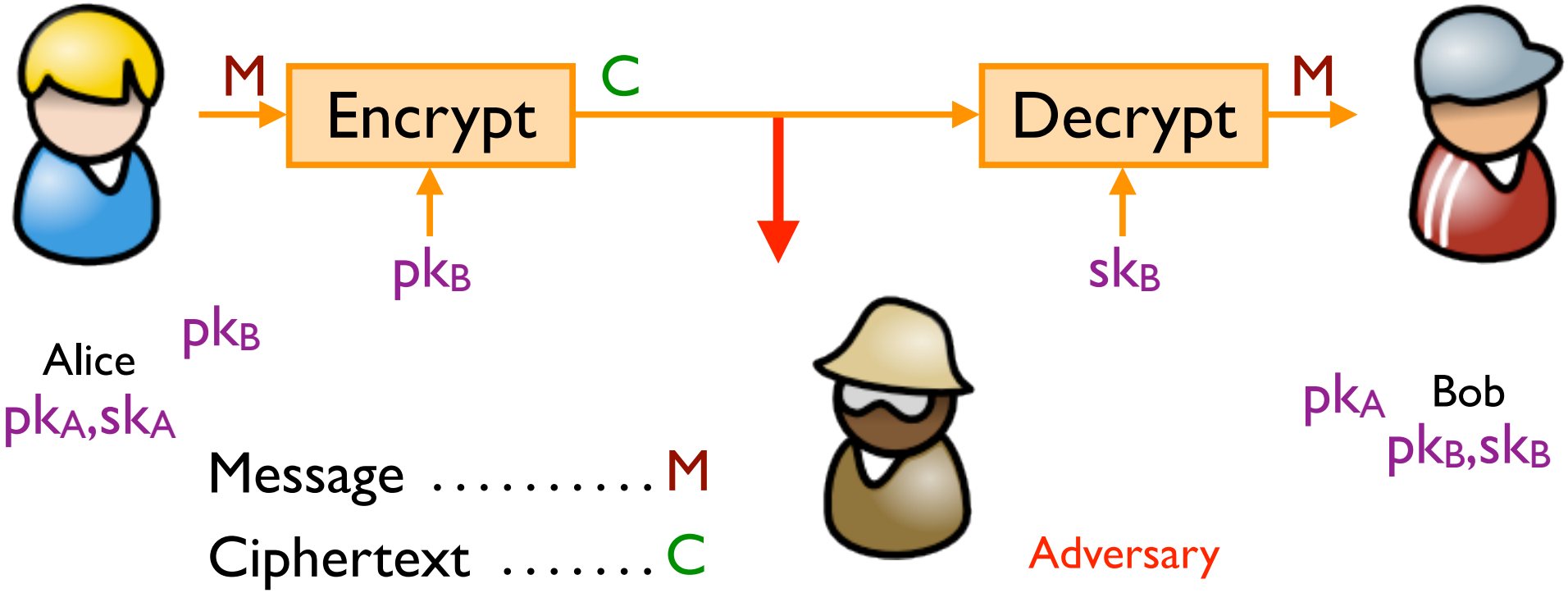
# Achieving Privacy (Symmetric)

Encryption schemes: A tool for protecting privacy.



# Achieving Privacy (Asymmetric)

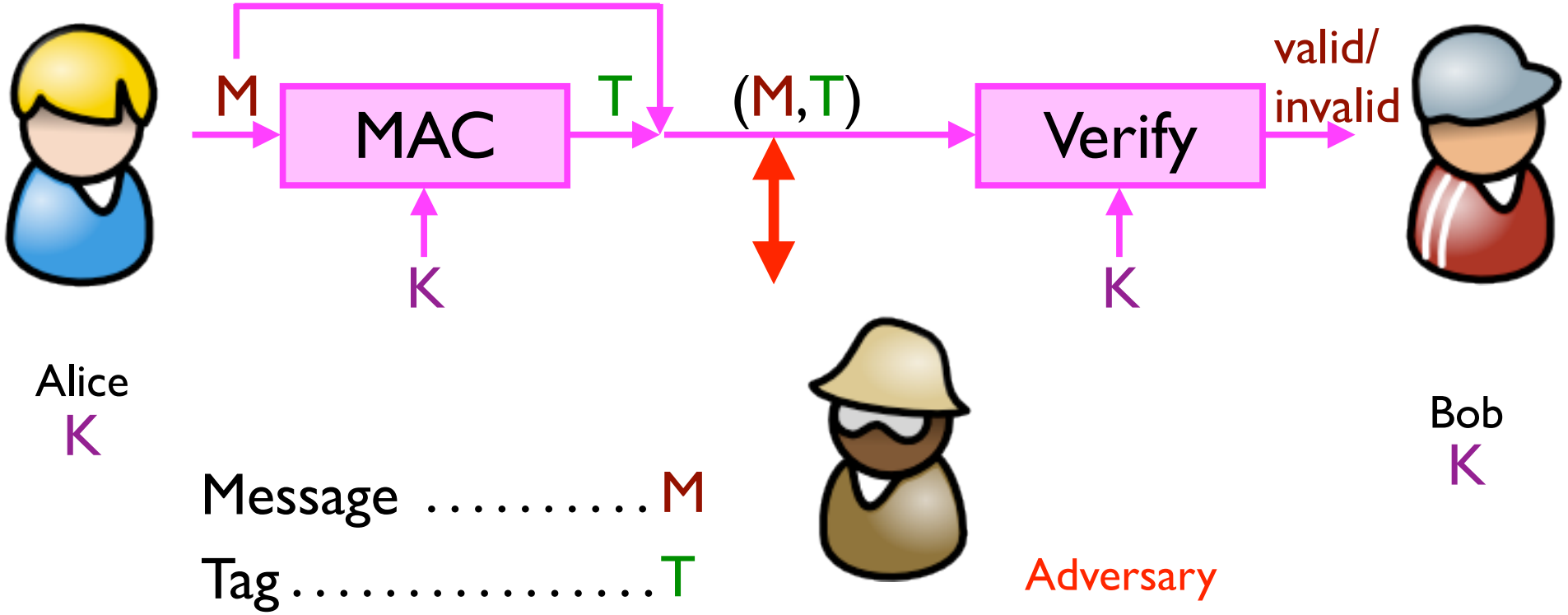
Encryption schemes: A tool for protecting privacy.



# Achieving Integrity (Symmetric)

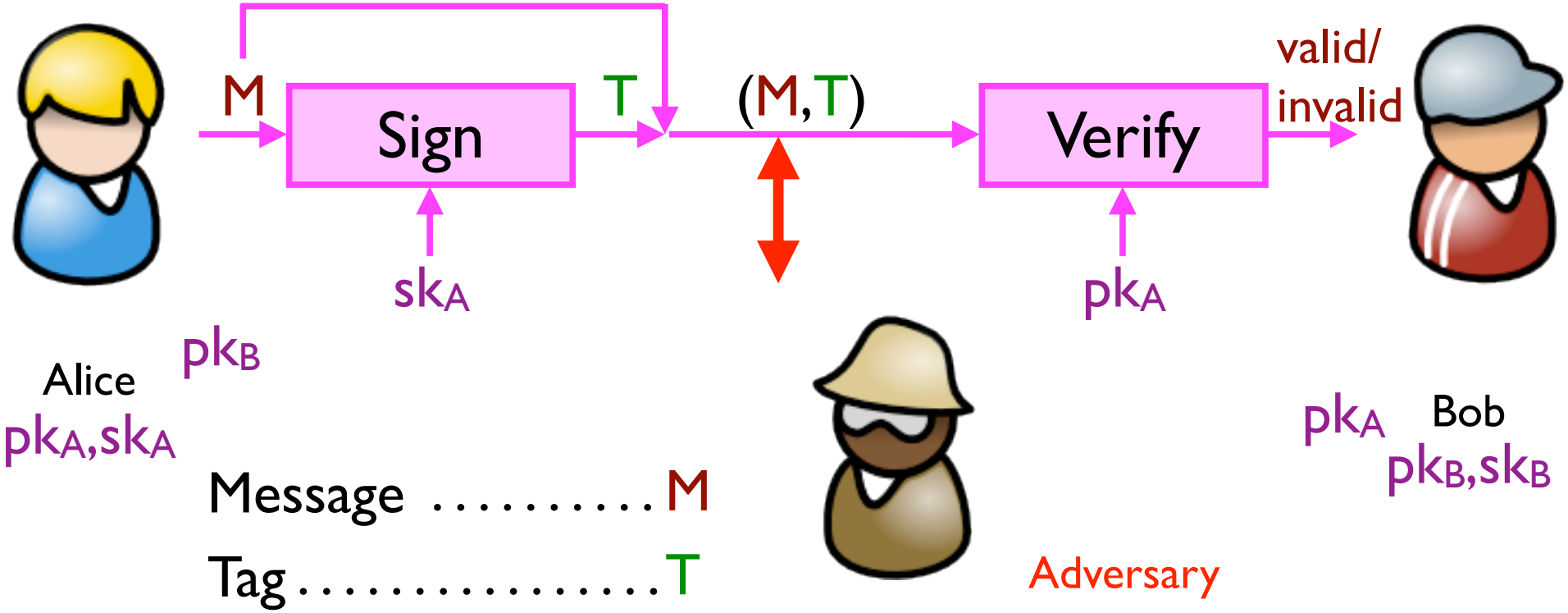
Message authentication schemes: A tool for protecting integrity.

(Also called message authentication codes or MACs.)



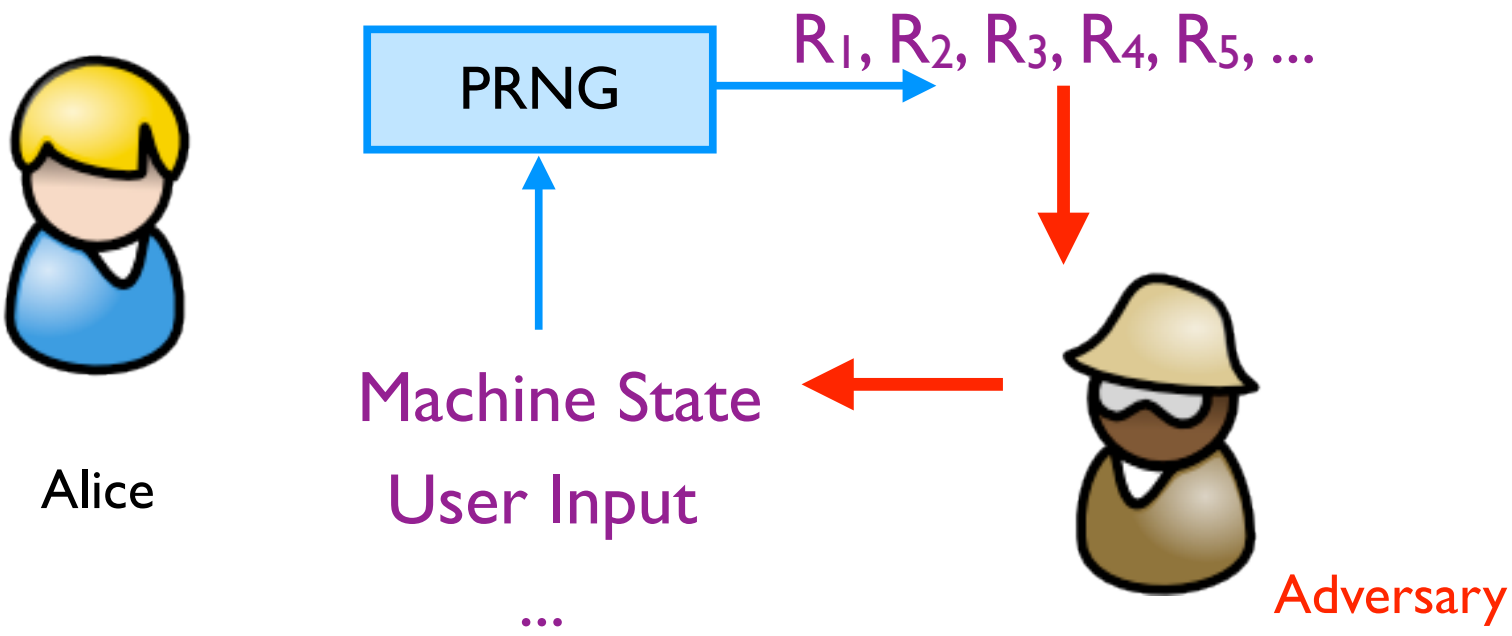
# Achieving Integrity (Asymmetric)

Digital signature schemes: A tool for protecting integrity and authenticity.



# “Random” Numbers

Pseudorandom Number Generators (PRNGs)



# Getting keys: PBKDF

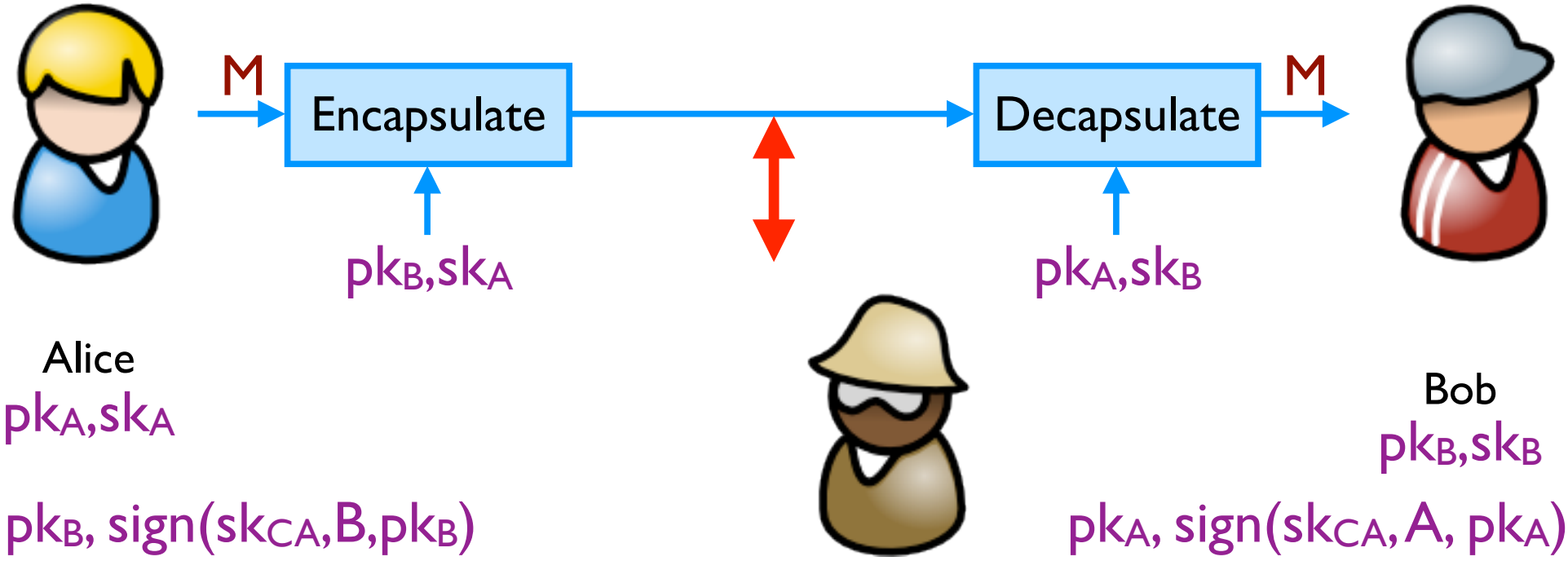
Password-based Key Derivation Functions


Alice

# Getting keys: CAs

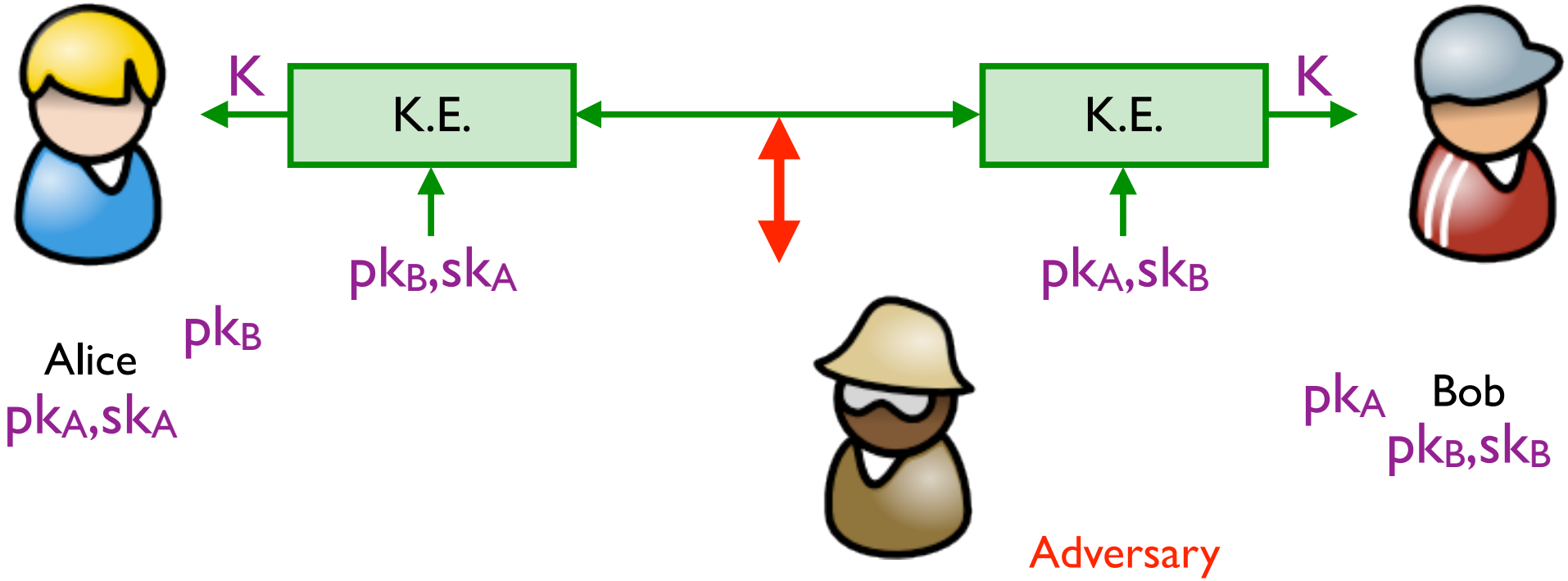
Each party creates a public key pk and a secret key sk.

(Public keys signed by a trusted third party: a certificate authority.)



# Getting keys: Key exchange

Key exchange protocols: A tool for establishing a shared symmetric key from public keys



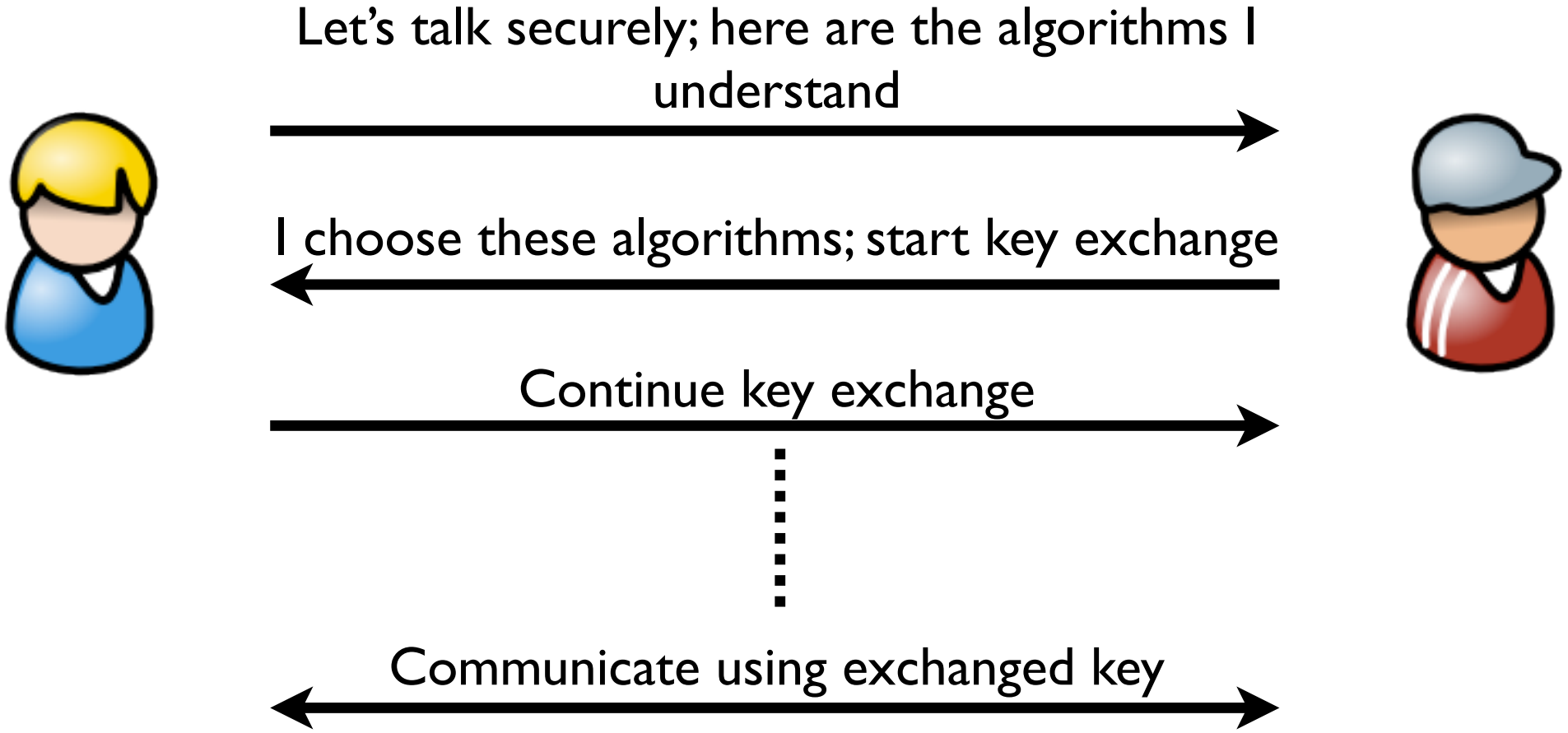
# One-way Communications

PGP is a good example

Message encrypted under Bob's public key

# Interactive Communications

In many cases, it's probably a good idea to just use a standard protocol/system like SSH, SSL/TLS, etc...



# Let's Dive a Bit Deeper

# One-way Communications

(*Informal* example; ignoring, e.g., signatures)

1. Alice gets Bob's public key; Alice *verifies* Bob's public key (e.g., via CA)
2. Alice generates random symmetric keys K1 and K2
3. Alice encrypts the message M the key K1; call result C
4. Alice authenticates (MACs) C with key K2; call the result T
5. Alice encrypts K1 and K2 with Bob's public key; call the result D

6. Send D, C, T

(Assume Bob's private key is encrypted on Bob's disk.)

7. Bob takes his password to derive key K3
8. Bob decrypts his private key with key K3
9. Bob uses private key to decrypt K1 and K2
10. Bob uses K2 to verify MAC tag T
11. Bob uses K1 to decrypt C

# Interactive Communications

(*Informal* example; details omitted)

1. Alice and Bob exchange public keys and certificates
2. Alice and Bob use CA's public keys to verify certificates and each other's public keys
3. Alice and Bob take their passwords and derive symmetric keys
4. Alice and Bob use those symmetric keys to decrypt and recover their asymmetric private keys.
5. Alice and Bob use their asymmetric private keys and a *key exchange* algorithm to derive a shared symmetric key

(They key exchange process will require Alice and Bob to generate new pseudorandom numbers)

6. Alice and Bob use shared symmetric key to encrypt and authenticate messages

(Last step will probably also use random numbers; will need to rekey regularly; may need to avoid replay attacks,...)

What cryptosystems have you heard of? (Past or present)

# History

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

- Recommended Reading: **The Codebreakers** by David Kahn and **The Code Book** by Simon Singh.
  - Military uses
  - Rumrunners
  - ....

# Classic Encryption

- Goal: To communicate a secret message
- Start with an *algorithm*
- Caesar cipher (substitution cipher):

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ

GHIJKLMNOPQRSTUVWXYZABCDEF
```

# Then add a secret key

- Both parties know that the secret word is "victory":

  ABCDEFGHIJKLMNOPQRSTUVWXYZ

  **VICTORY**ABDEFGHJKLMNPQSUWXZ

- "state of  the art" for thousands of years

# 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.

# Checkpoint

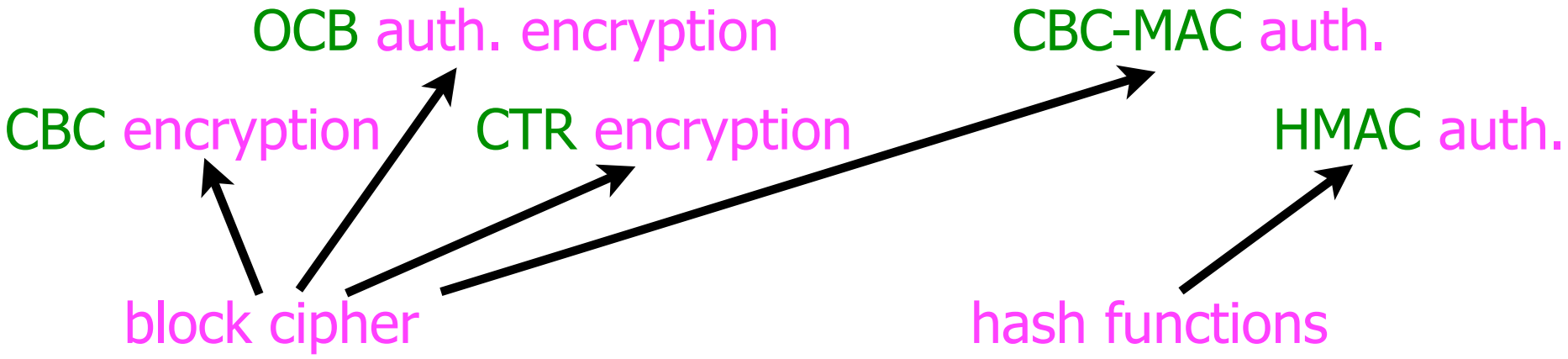
- **Symmetric cryptography**
  - Both sides know ***shared key***, no one else knows anything. Can ***encrypt, decrypt, sign/MAC, verify***
  - Computationally lightweight
  - ***Challenge:*** How do you privately share a key?
- **Asymmetric cryptography**
  - Everyone has a ***public*** *key* that everyone else knows; and a paired ***secret*** *key* that is private
  - Public key can ***encrypt***; only secret key can ***decrypt***
  - Secret key can ***sign/MAC***, public key can ***verify***
  - Computationally expensive
  - ***Challenge:*** How do you validate a public key?

# Checkpoint

- **Where are public keys from?**
  - One solution: keys for ***Certificate Authorities*** *a priori* known by browser, OS, etc.
- **Where are shared keys from?**
  - In person exchange, snail mail, etc.
  - If we have verifiable public/private keys: ***key exchange*** protocol generates a shared key for symmetric cryptography

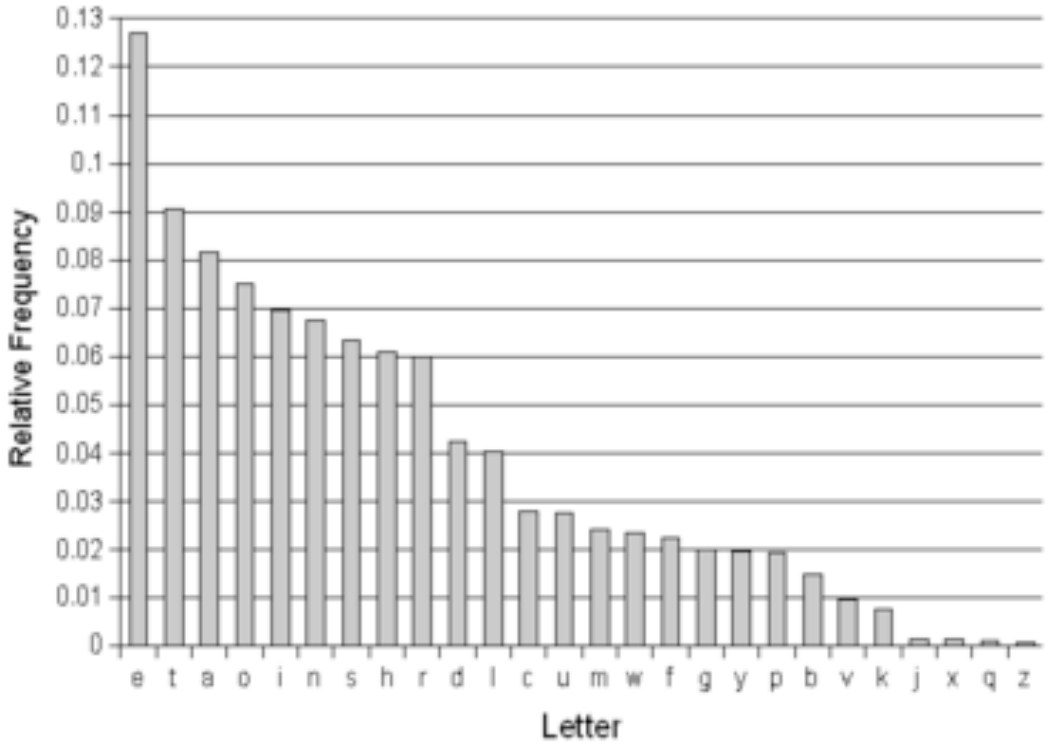
# How cryptosystems work today

- Layered approach:
  - Cryptographic primitives, like block ciphers, stream ciphers, hash functions, and one-way trapdoor permutations
  - Cryptographic protocols, like CBC mode encryption, CTR mode encryption, HMAC message authentication
- Public algorithms (Kerckhoff's Principle)
- Security proofs based on assumptions (not this course)



# “Old Days” Cryptanalysis and Probabilities

| Letter | Frequency |
|---|---|
| a | 8.167% |
| b | 1.492% |
| c | 2.782% |
| d | 4.253% |
| e | 12.702% |
| f | 2.228% |
| g | 2.015% |
| h | 6.094% |
| i | 6.966% |
| j | 0.153% |
| k | 0.772% |
| l | 4.025% |



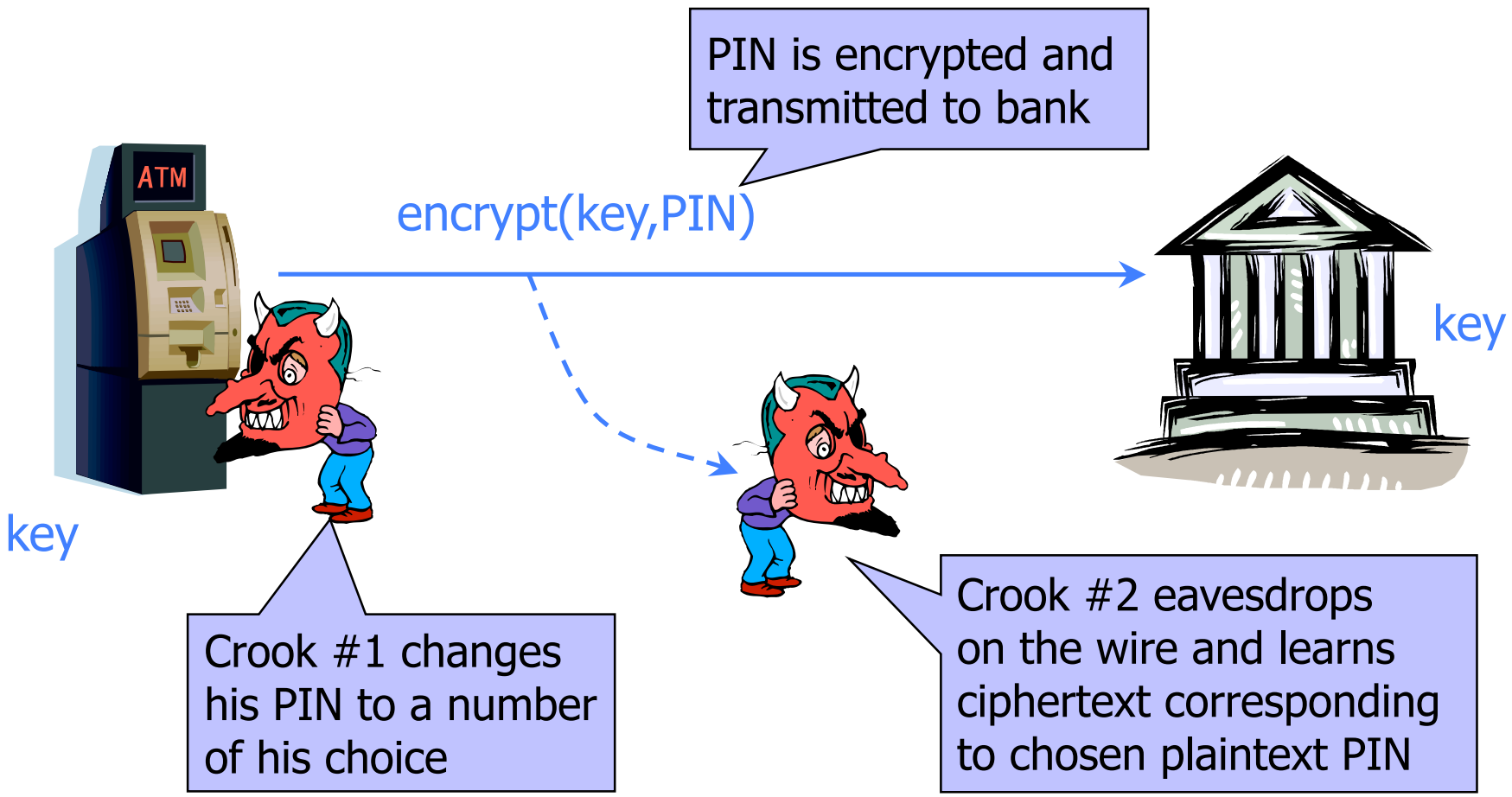

From http://en.wikipedia.org/wiki/Letter_frequencies

# Attack Scenarios for Encryption

- Ciphertext-Only
- Known Plaintext
- Chosen Plaintext
- Chosen Ciphertext (and Chosen Plaintext)

- (General advice:  Target strongest level of privacy possible -- even if not clear why -- for extra "safety")

# Chosen-Plaintext Attack

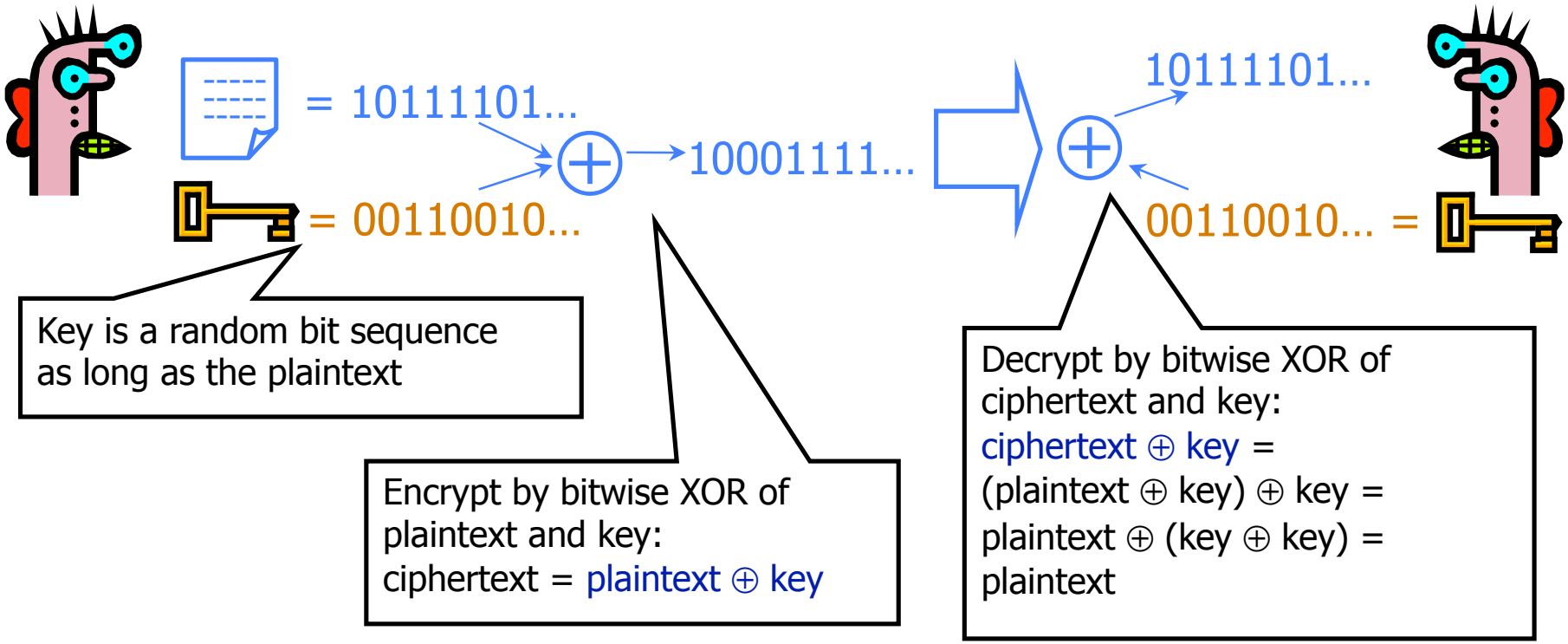



... repeat for any PIN value

# Attack Scenarios for Integrity

- What do you think these scenarios should be?

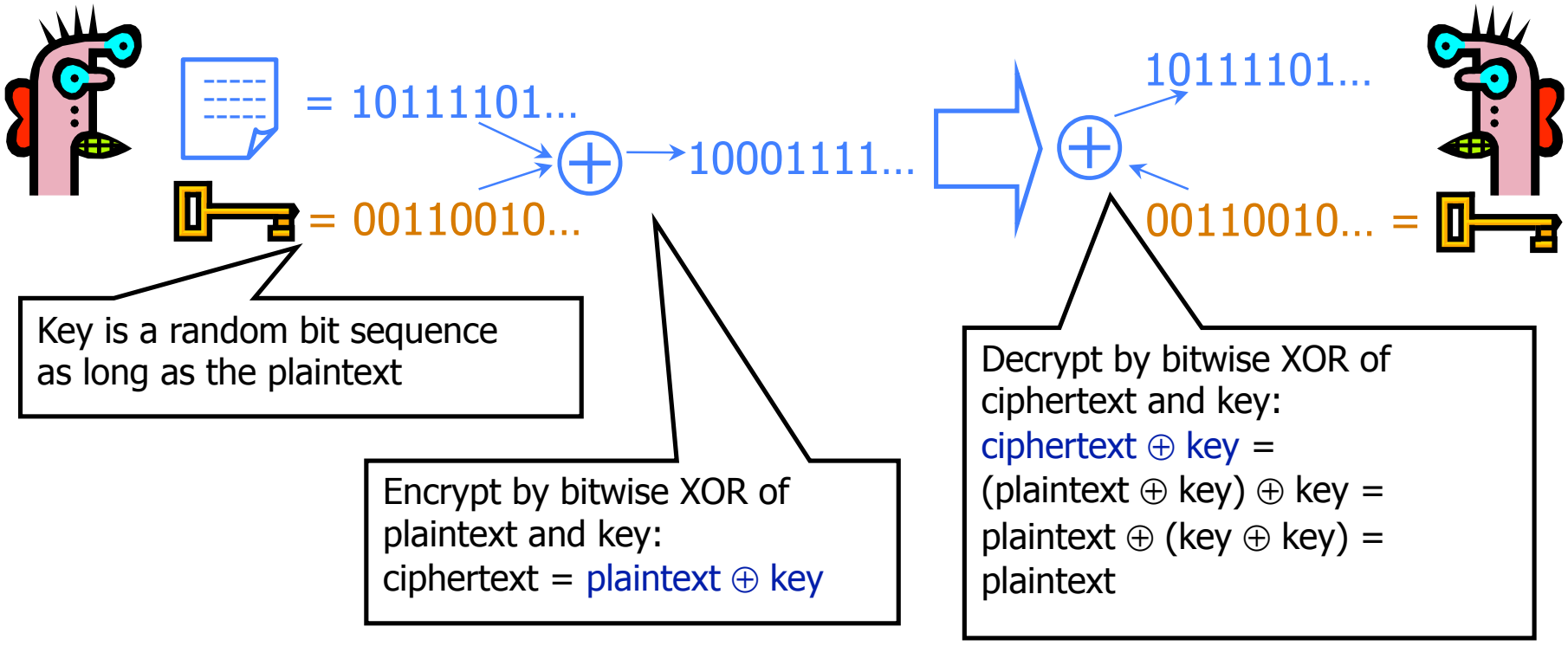
# One-Time Pad


= 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

# 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 the sender communicate the key to receiver?

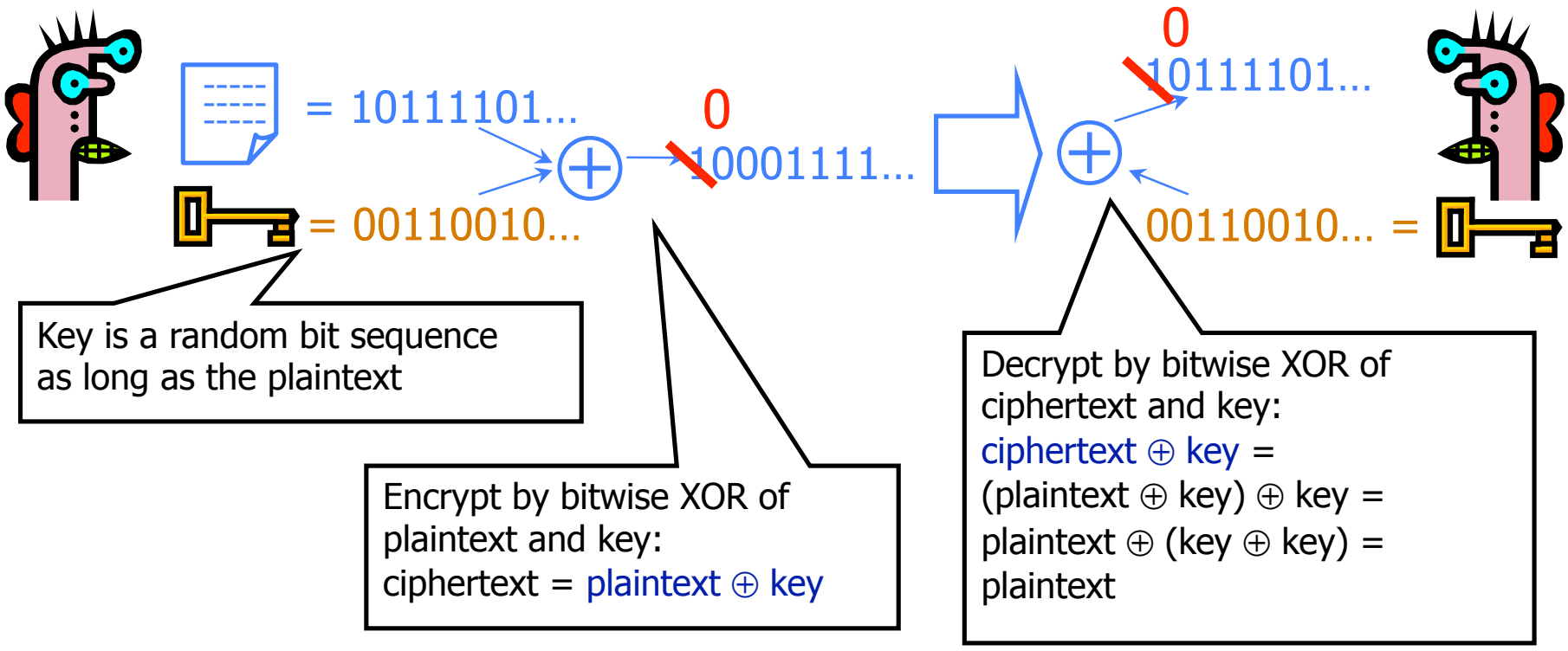
# Disadvantages




Disadvantage #1: Keys as long as messages.

- Impractical in most scenarios
- Still used by intelligence communities

# Disadvantages




Disadvantage #2: No integrity protection