Confidentiality Attacks and Key Exchange

EECS 388 F17

UNIVERSITY OF MICHIGAN

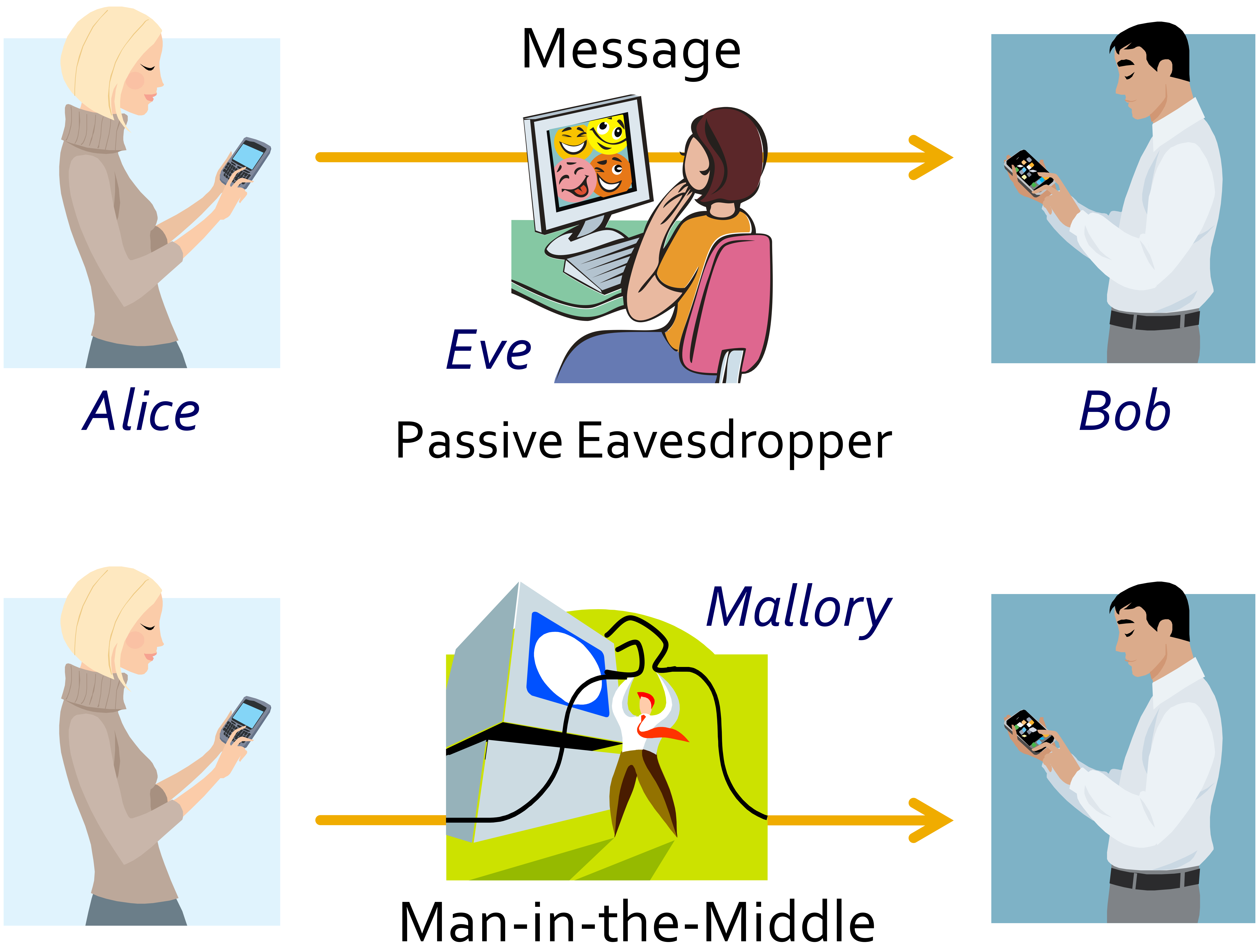
Review

Properties of a Secure Channel

Confidentiality

Integrity

Authentication (coming soon)


Message
Eve
Alice
Passive Eavesdropper
Bob
Mallory
Man-in-the-Middle

Integrity Review

1. Let f be a secure PRF.

2. In advance choose a random k known only to Alice and Bob.

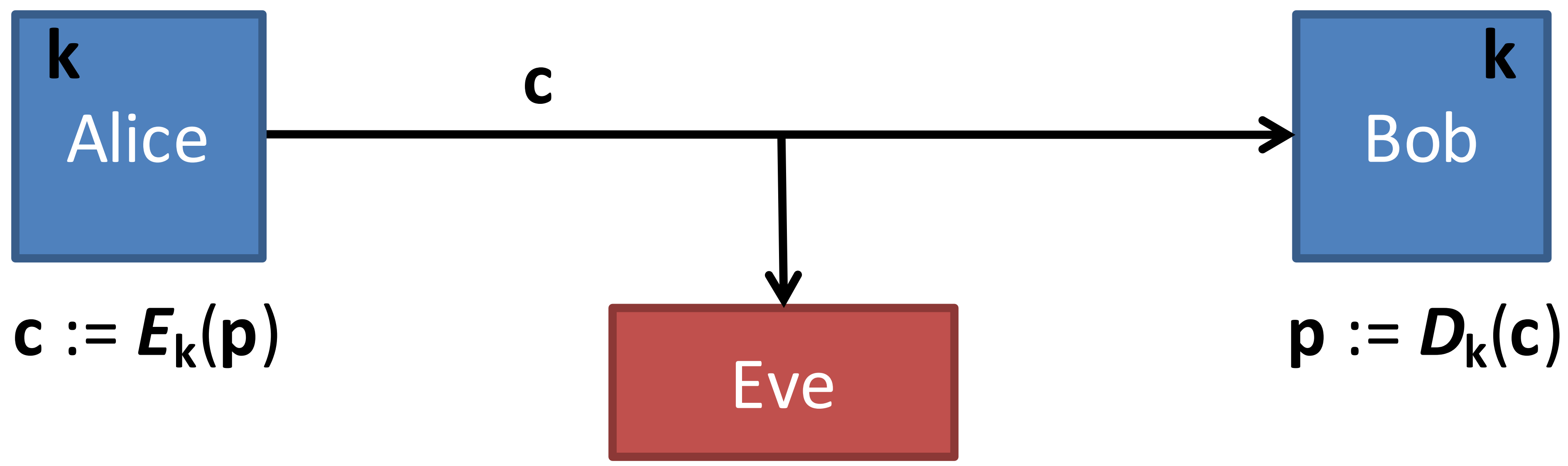
3. Alice computes **v** := $\boldsymbol{f}_{\mathbf{k}}(\mathbf{m})$.

4. 

5. Bob verifies **v′** = $\boldsymbol{f}_{\mathbf{k}}(\mathbf{m'})$, accepts if and only if this is true.

Confidentiality Review

Goal: Keep contents of message ***p*** secret from an eavesdropper



$c := E_k(p)$ $p := D_k(c)$

p	plaintext
c	ciphertext
m	message / plaintext
k	secret key
E	encryption function
D	decryption function

Encryption / Integrity Ordering

Encrypt, then MAC

Encrypt, then MAC

Encrypt, then MAC

Cryptographic Doom Principle: If you have to perform *any* cryptographic operation before verifying the MAC on a message you've received, it will inevitably lead to doom.

New Stuff

Padding Oracles

Must be able to distinguish between invalid MAC and invalid padding

Enough to learn plaintext

Vaudenay padding oracle attack

Recall: Cipher-block Chaining (CBC)

For each block P_i, do:

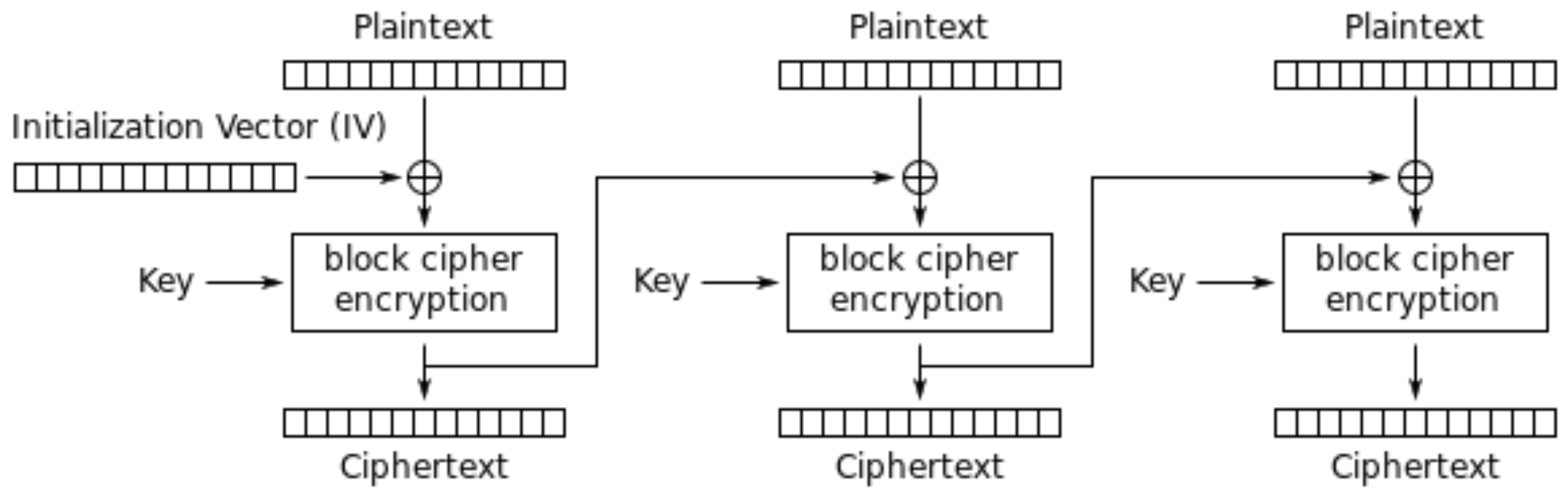
C_0 := *initialization vector*

$C_i := E_k(P_i \oplus C_{i-1})$

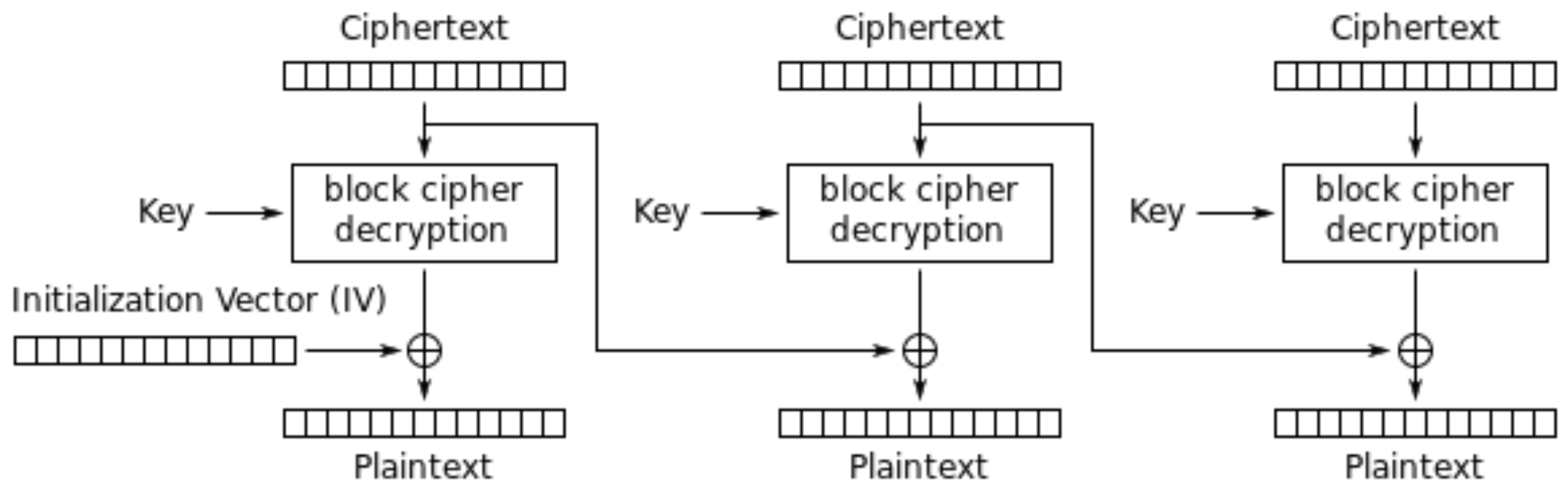
To decrypt C_i, do:

C_0 := *initialization vector*

$P_i := D_k(C_i) \oplus C_{i-1}$



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Padding Oracle Attack Example

YouTube Link

Another more “mathy" example is here:

https://en.wikipedia.org/wiki/Padding_oracle_attack#Example_of_the_attack_on_CBC_encryption

Padding Oracle Defenses

Don’t use separate errors for MAC vs padding

Use *constant-time code (i.e.* all code paths must be equal with limited branching) [why? (<u>side channel attack</u>)]

Always check **integrity** (i.e. ***encrypt, then MAC***)

Don’t use CBC mode!

AEAD (to the rescue)

Authenticated Encryption and Associated Data

```
ciphertext, auth_tag := Seal(key, plaintext, associated_data)

plaintext := Unseal(key, ciphertext, associated_data, auth_tag)
```

Combine integrity and encryption into a single primitive — woohoo!

Commonly used is AES-GCM (“Galois Counter Mode”), has hardware support on modern Intel processors.

ChaCha-Poly1305, Salsa20-Poly1305, common on mobile devices.

Key Size

How big should keys be?

<u>Moore's Law</u>: Computer's get twice as good for the same price every 18 months.

Current reasonable safe size: *128 bits*

Worried about quantum computers? *256 bits*

MACs/PRFs need to be 2x cipher key size [*why?*]

Review: Building a Secure Channel

What if you want <u>confidentiality</u> and <u>integrity</u> at the **same time**?

- Encrypt, *then* MAC. Better yet, use an AEAD!
- Use *separate keys* for confidentiality and integrity. Better yet, use an AEAD!
- Need two (or more) shared keys, but only have one? That’s what PRG’s are for!
- If there’s a reverse channel (Bob to Alice), use *separate keys* for that!

Key Exchange

Alice and Bob can have a **public** conversation to derive a **secret** key.

Diffie-Hellman

<u>1976</u>: Whitfield Diffie, Marty Hellman with ideas from Ralph Merkle

- Earlier, in secret, by Malcolm Williamson of British intelligence agency GCHQ

Relies on a mathematical hardness assumption called *discrete log problem* (a problem believed to be hard)

DH Protocol

Standard g (generator), and p (prime, or modulus)

Alice picks secret a

$g^a \bmod p$ →

← $g^b \bmod p$

$g^{ab} \bmod p = g^{ba} \bmod p$

Bob picks secret b

DH Protocol Example (from wikipedia)

Non-secret values in blue, and secret values in **red**:

1. Alice and Bob agree to use a modulus $p = 23$ and base $g = 5$.
2. Alice chooses a secret integer $\boldsymbol{a} = \mathbf{6}$, then sends Bob $A = g^{a} \bmod p$
   - $A = 5^{6} \bmod 23 = 8$
3. Bob chooses a secret integer $\boldsymbol{b} = \mathbf{15}$, then sends Alice $B = g^{b} \bmod p$
   - $B = 5^{15} \bmod 23 = 19$
4. Alice computes $\boldsymbol{s} = B^{a} \bmod p$
   - $\boldsymbol{s} = 19^{6} \bmod 23 = \mathbf{2}$
5. Bob computes $\boldsymbol{s} = A^{b} \bmod p$
   - $\boldsymbol{s} = 8^{15} \bmod 23 = \mathbf{2}$
6. Alice and Bob now share a secret (the number **2**).

$$A^b \bmod p = g^{ab} \bmod p = g^{ba} \bmod p = B^a \bmod p$$

More specifically,

$$(g^a \bmod p)^b \bmod p = (g^b \bmod p)^a \bmod p$$

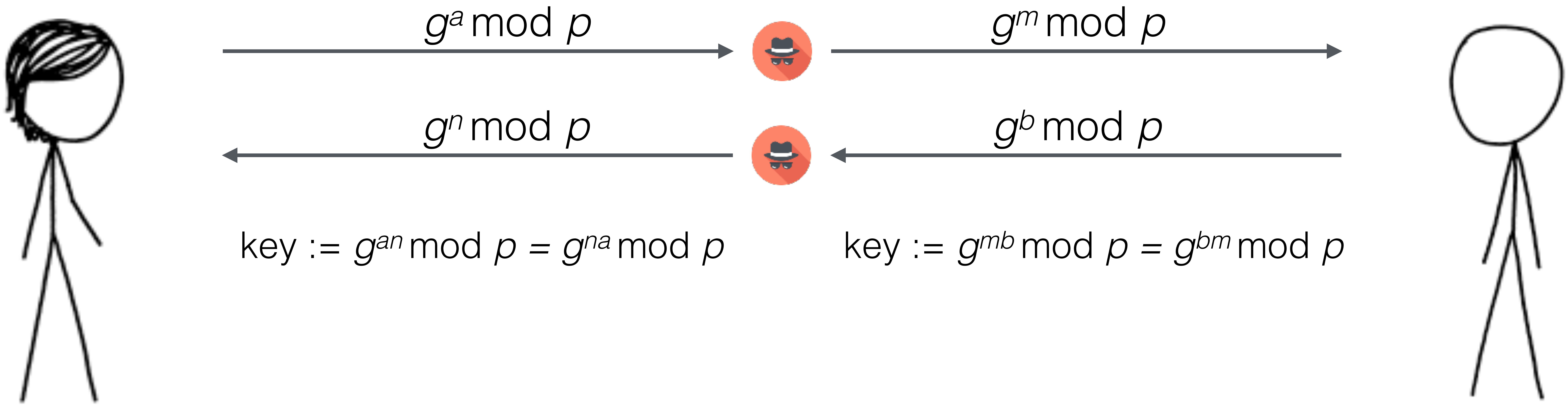
Another DH Key Exchange Example

Colors!

https://youtu.be/YEBfamv-_do?t=163

DH Protocol (MITM)

Standard g (generator), and p (prime, or modulus)



Defending against MITM

1) Cross your fingers and hope.

2) Rely on out-of-band communication between users.

3) Rely on physical contact to make sure there’s no MITM.

4) Use digital signatures. [More on this later]

Key Management

1. Key management is the hard part (protecting them, communicating them, etc…)

2. Each key should only have one purpose

3. Vulnerability of a key increases:
    i. The more you use it
    ii. The more places you store it
    iii. The longer you have it

4. Keep your keys far away from the attacker (don’t post them on the web!)

5. Protect yourself against old keys that have been compromised

6. Goal: *forward secrecy*

Forward Secrecy

Learning old key shouldn't help adversary learn new key

Compromising an individual session should not compromise future sessions

Compromising a long-term key should not enable decryption of recorded ciphertexts (more on this later)

So Far

Assuming no imposters

Next Lecture…

Authentication, RSA, digital signatures

Putting it all together