رمزنگاری، امنیت اطلاعات و حریم خصوصی

ارائه: دکتر سیدعلی لاجوردی

بخش سوم

Encrypting long messages?

- Recall that CPA-security $\Rightarrow$ security for the encryption of multiple messages

- So, can encrypt the message m1, …, mt as Enck(m1), Enck(m2), …, Enck(mt)
  - This is also CPA-secure!

Drawback

- The ciphertext in that case is twice the length of the plaintext
  - I.e., ciphertext expansion by a factor of two
- Can we do better?
- Modes of operation
  - Stream-cipher modes of operation
  - Block-cipher modes of operation

Stream ciphers

Stream ciphers

- As we defined them, PRGs are limited
  - They have fixed-length output
  - They produce output in “one shot”

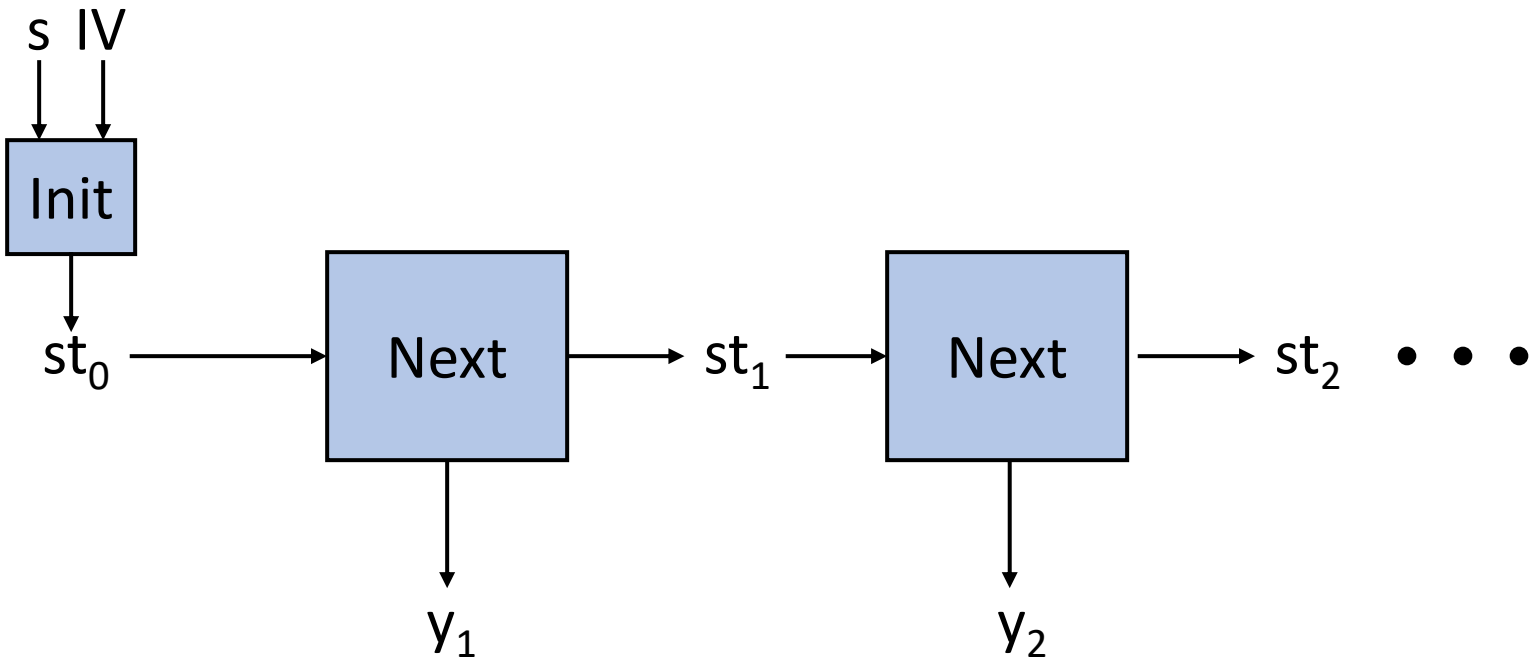
- In practice, stream ciphers are used
  - Can be viewed as producing an “infinite” stream of pseudorandom bits, on demand
  - More flexible, more efficient

Stream ciphers

- Pair of efficient, deterministic algorithms (Init, Next)
  - Init takes a seed s (and optional IV), and outputs initial state st
  - Next takes the current state st and outputs a bit y along with updated state st’
    - (In practice, y would be a block rather than a bit)

Stream ciphers

- Can use (Init, Next) to generate any desired number of output bits from an initial seed/IV



AES

- Advanced encryption standard (AES)
  - Key length = 128, 192, or 256 bits
  - Block length = 128 bits
- Will discuss details later in the course
- Available in standard crypto libraries
- No real reason to use anything else

Stream ciphers

- A stream cipher is secure if the output stream (from a uniform seed) is pseudorandom
  - I.e., regardless of how long the output stream is (as long as it is polynomial)
  - See book for formal definition

- Easy to construct from a block cipher (see book)

Modes of operation

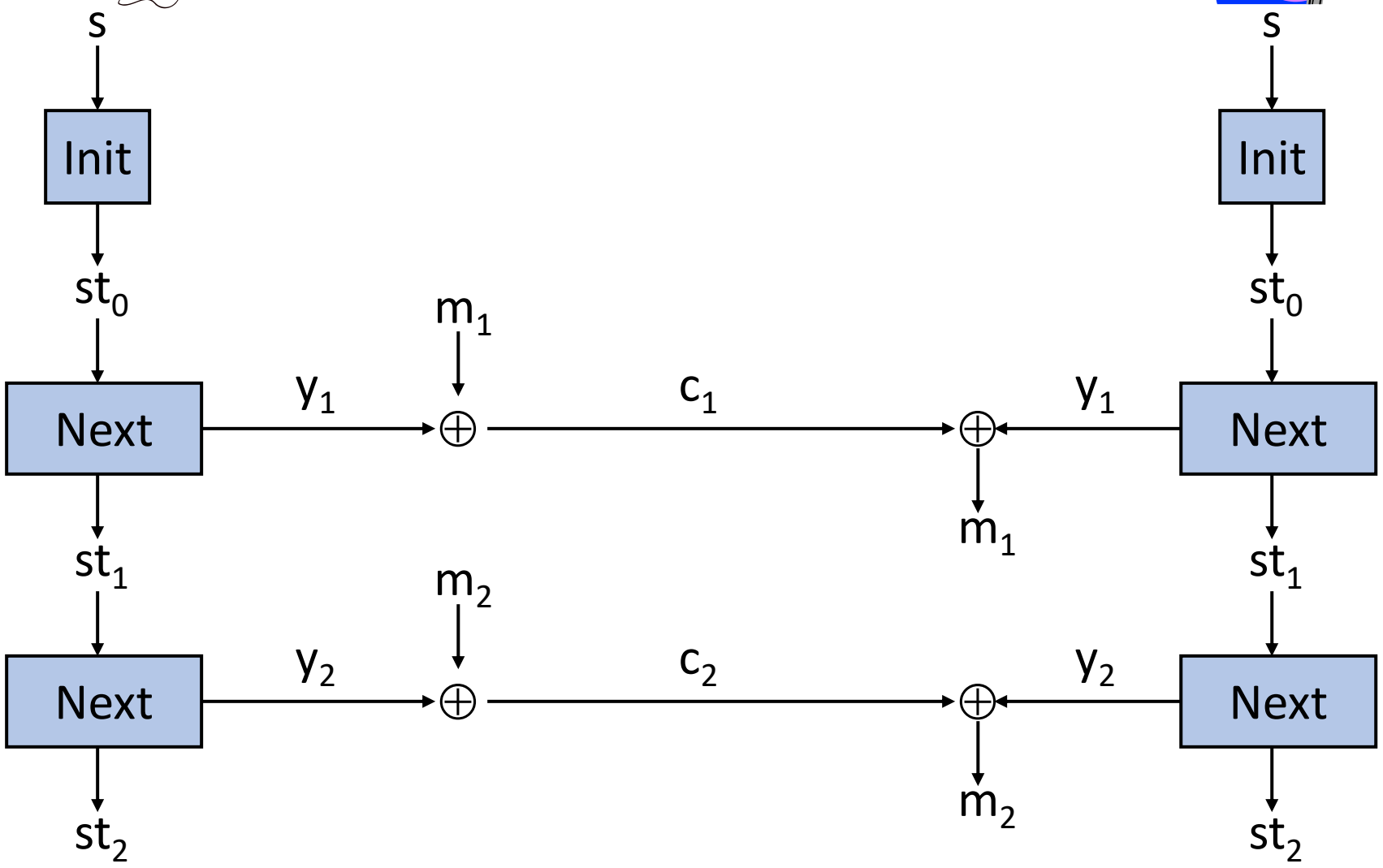
- Stream-cipher modes of operation
  - Synchronized
  - Unsynchronized

Synchronized mode

- Sender and receiver maintain state (i.e., they are stateful), and must be synchronized

- Makes sense in the context of a limited-time communication session where both parties are online and messages are received in order, without being dropped

Synchronized mode



Synchronized mode

- Advantages
  - Stream cipher does not need to support an IV
  - No ciphertext expansion

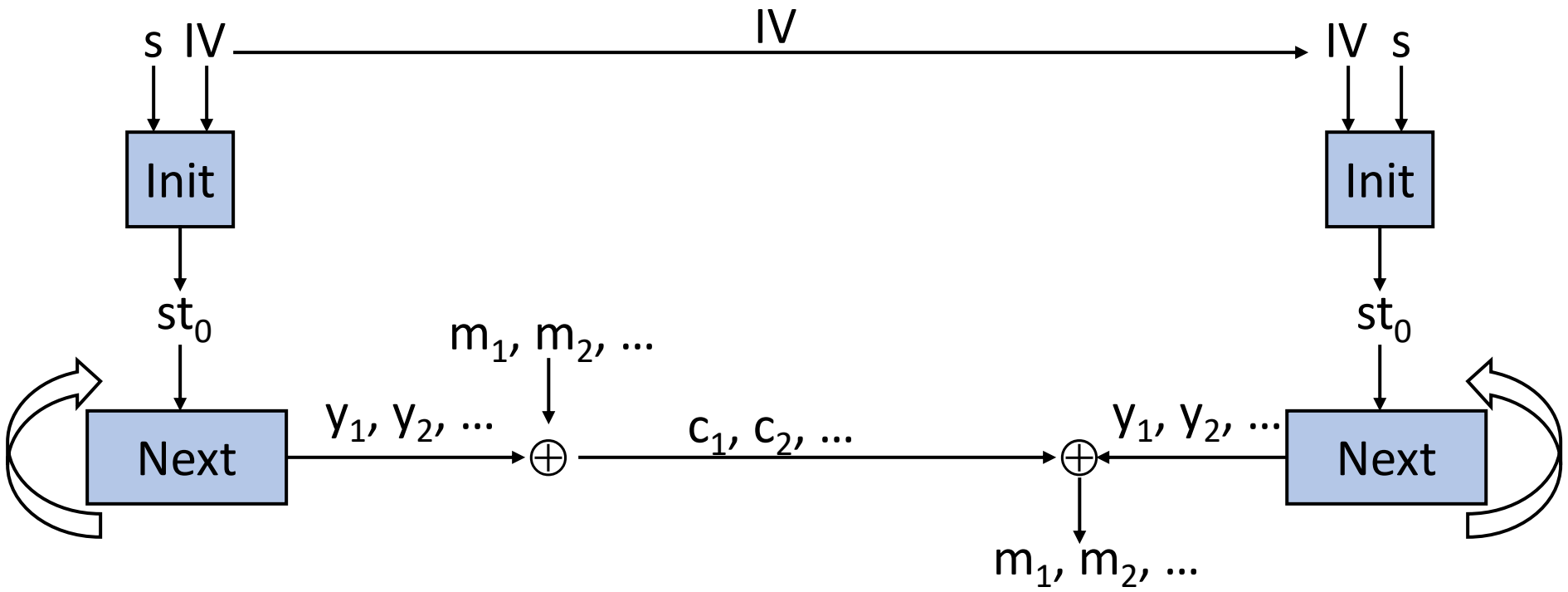
- Disadvantages
  - Stateful
  - Assumes messages arrive in order; never dropped

Unsynchronized mode

- Choose random IV to encrypt next message

- Similar to the first CPA-secure scheme we saw
  - But “natively” handles arbitrary-length messages with better ciphertext expansion

Unsynchronized mode



Block-cipher modes of operation

ECB mode

- Enck(m1, …, mt) = Fk(m1), …, Fk(mt)

- Deterministic
  - Not CPA-secure!

- Can tell from the ciphertext whether mi = mj
  - Not even EAV-secure!

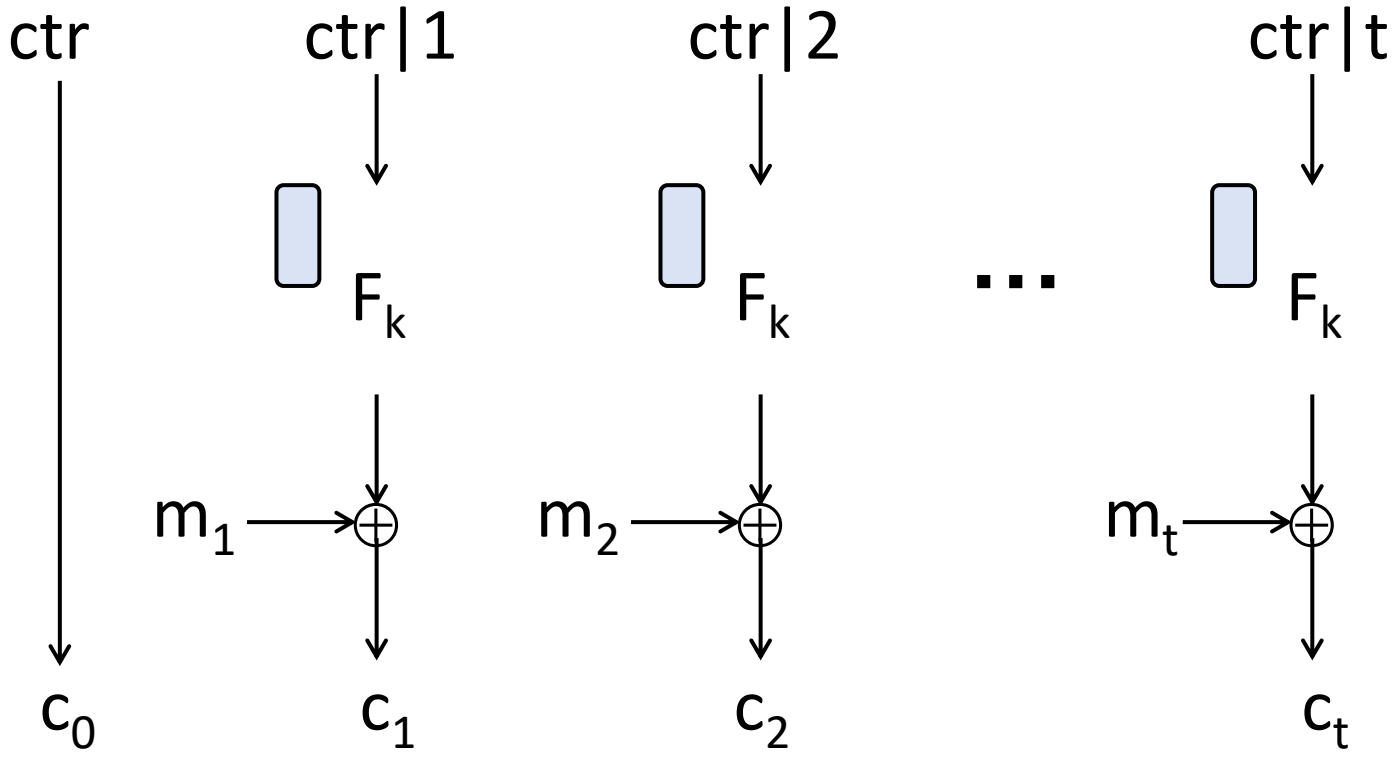
CTR mode

- Enck(m1, …, mt) // note: t is arbitrary
  - Choose ctr $\leftarrow$ {0,1}3n/4, set c0 = ctr
  - For i=1 to t:
    - ci = mi $\oplus$ Fk(ctr | i)
  - Output c0, c1, …, ct

- Decryption?
  - Note that F need not be invertible

- Ciphertext expansion is <1 block

CTR mode



CTR mode

- Theorem: If F is a pseudorandom function, then CTR mode is CPA-secure

- Proof sketch:
- The sequence Fk(ctr | 1), …, Fk(ctr | t) used for the challenge ciphertext is pseudorandom
  - Moreover, it is independent of every other such sequence unless ctr | j = ctri' | j' for some i', j'
    - Just need to bound the probability of that event

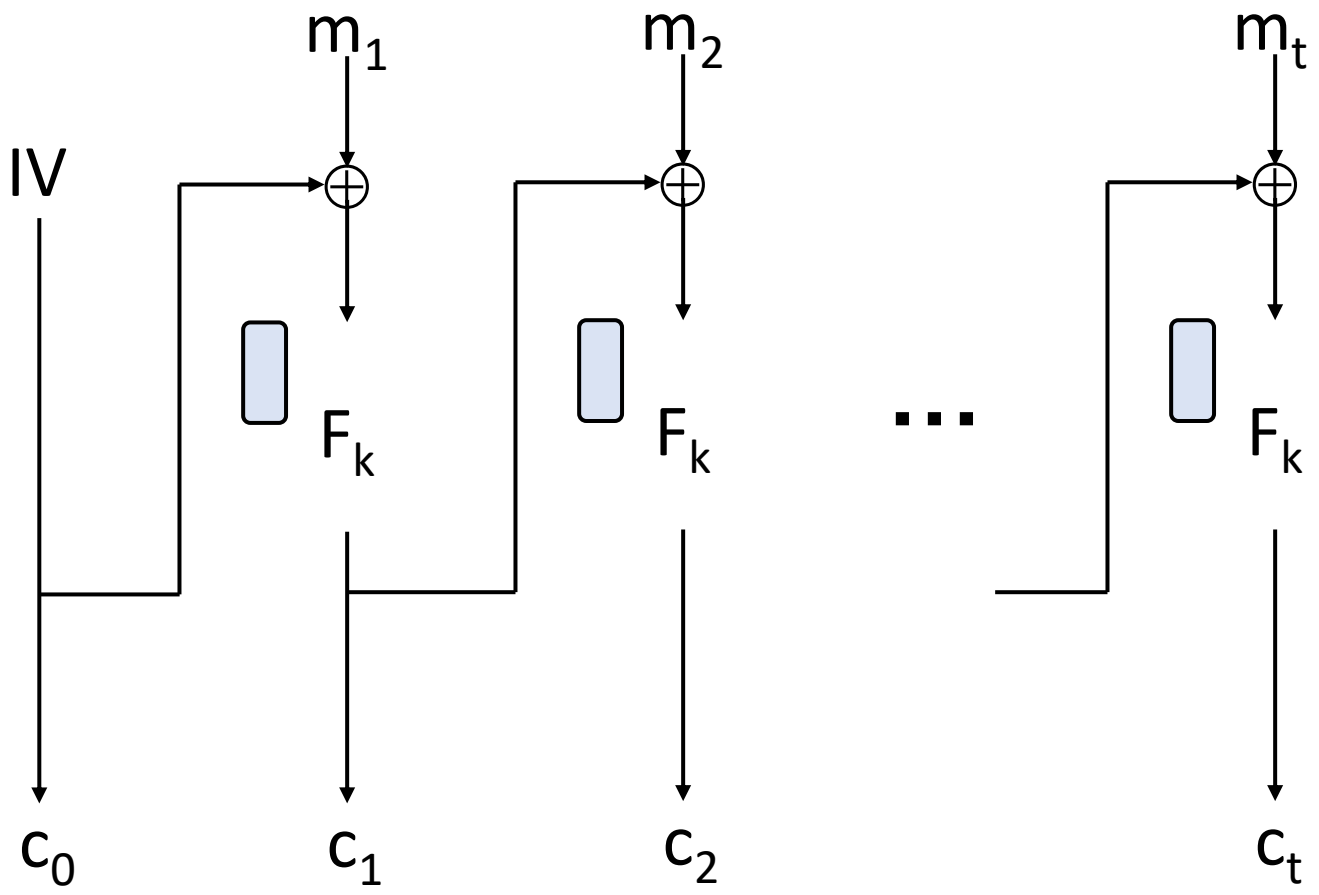
CBC mode

- Enck(m1, …, mt) // note: t is arbitrary
  - Choose random c0 $\leftarrow$ {0,1}n (also called the IV)
  - For i=1 to t:
    - ci = Fk(mi $\oplus$ ci-1)
  - Output c0, c1, …, ct

- Decryption?
  - Requires F to be invertible, i.e., a permutation

- Ciphertext expansion is just 1 block

CBC-mode encryption



CBC mode

- Theorem: If F is a pseudorandom permutation, then CBC mode is CPA-secure

- Proof is more complicated than for CTR mode

Message integrity

Secrecy vs. integrity

- So far we have been concerned with ensuring secrecy of communication

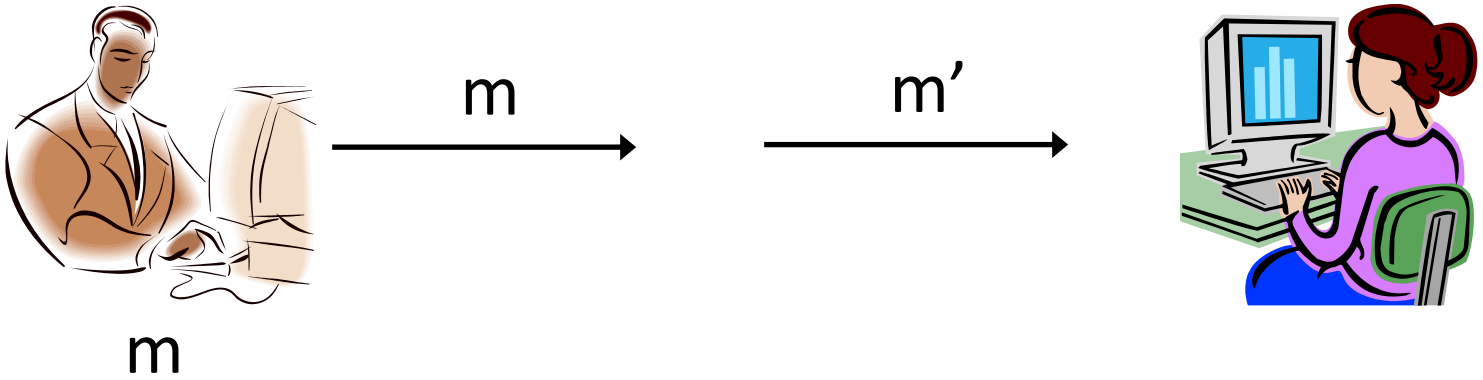
- What about integrity?
  - I.e., ensuring that a received message originated from the intended party, and was not modified

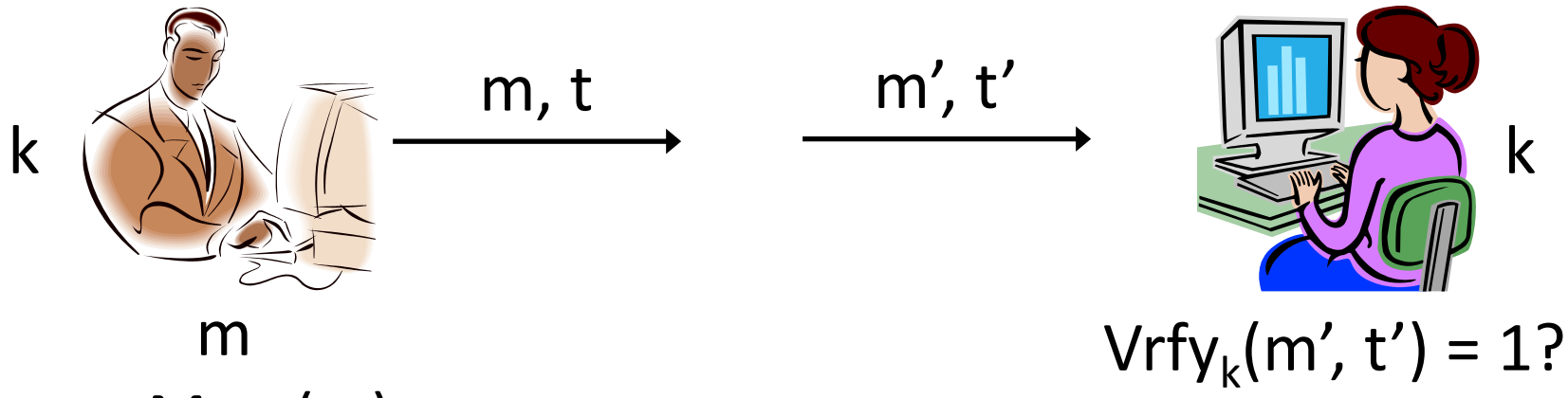
- Standard error-correction not enough!
  - The right tool is a message authentication code

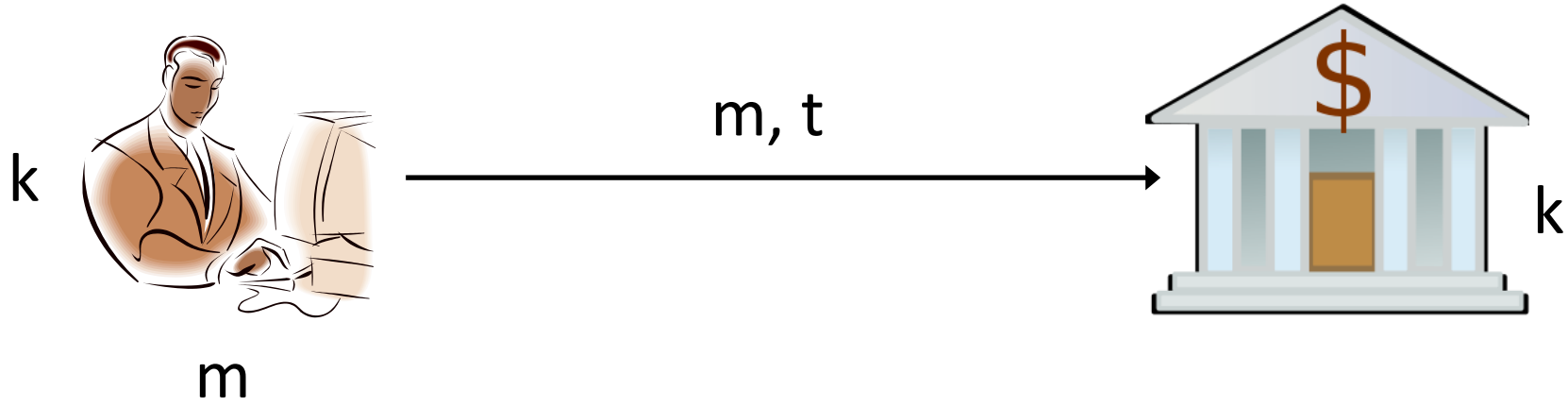
Passive attacks vs. active attacks

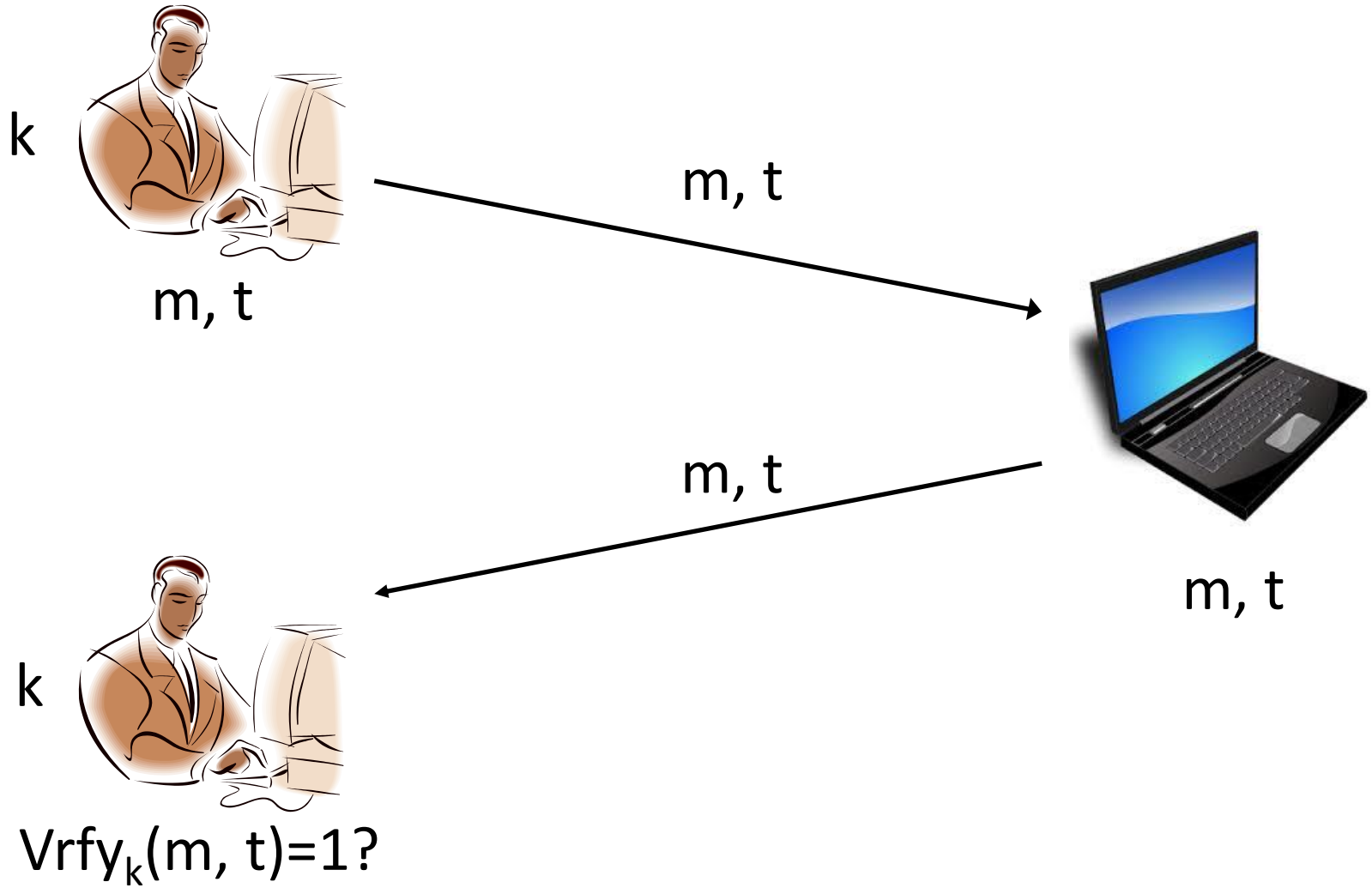
- So far we have been considered only passive (i.e., eavesdropping) attacks
  - Attacker simply observes the channel (even if it might also carry out a chosen-plaintext attack)

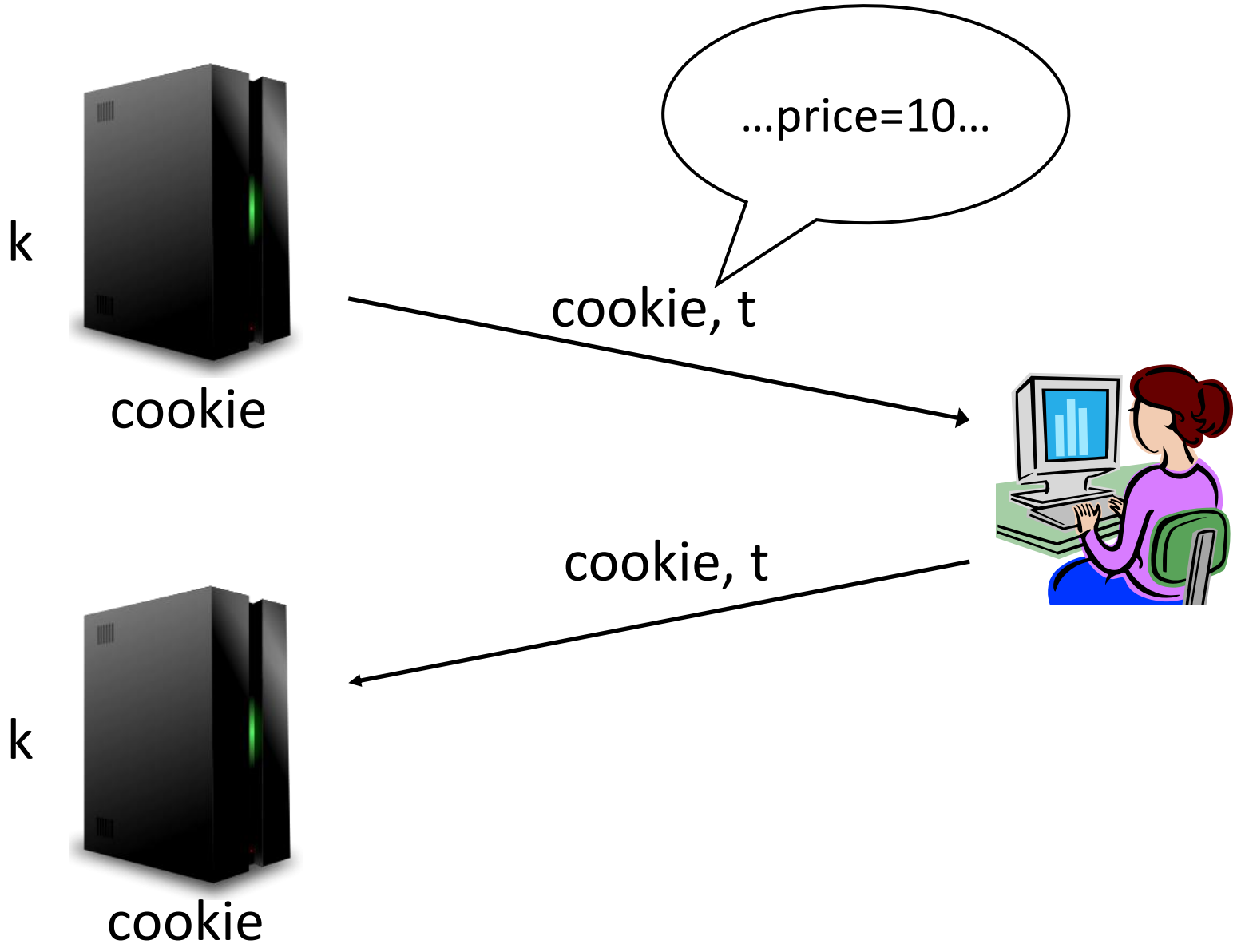
- In the setting of integrity, we explicitly consider active attacks
  - Attacker has full control over the channel


m
m’
m


k
m, t
m’, t’
k
m
t = Mac_k(m)
Vrfy_k(m’, t’) = 1?


k
m
m, t
$
k


k
m, t
m, t
m, t
m, t
k
Vrfy$_k$(m, t)=1?


k
cookie
…price=10…
cookie, t
cookie, t
k
cookie

Message authentication code (MAC)

- A message authentication code is defined by three PPT algorithms (Gen, Mac, Vrfy):
  - Gen: takes as input 1^n; outputs k. (Assume $|k| \geq n$.)
  - Mac: takes as input key k and message; outputs a tag t

    $t \leftarrow Mac_k(m)$
  - Vrfy: takes key k, message m, and tag t as input; outputs 1 ("accept") or 0 ("reject")

For all m and all k output by Gen,
$Vrfy_k(m, Mac_k(m)) = 1$

Security?

- Only one standard definition
- Threat model
  - “Adaptive chosen-message attack”
  - Assume the attacker can induce the sender to authenticate messages of the attacker’s choice
- Security goal
  - “Existential unforgeability”
  - Attacker should be unable to forge a valid tag on any message not previously authenticated by the sender

k

$t_1 := Mac_k(m_1)$

$t_2 := Mac_k(m_2)$

…

$t_i := Mac_k(m_i)$

m_1, t_1 →

m_2, t_2 →

⋮

m_i, t_i →

m, t →

k

$Vrfy_k(m, t)$??

Security?

- Is the definition too strong?
  - We don’t want to make any assumptions about what the sender might authenticate
  - We don’t want to make any assumptions about what forgeries are “meaningful”
- A MAC satisfying this definition can be used anywhere integrity is needed

Replay attacks

- Note that replay attacks are not prevented
  - No stateless mechanism can prevent them
- Replay attacks are often a significant real-world concern
- Need to protect against replay attacks at a higher level
  - Decision about what to do with a replayed message is application-dependent

A fixed-length MAC

Intuition?

- We need a keyed function Mac such that:
  - Given Mack(m1), Mack(m2), …,
  - …it is infeasible to predict the value Mack(m) for any $m \notin \{m1, \ldots, \}$

- Let Mac be a pseudorandom function!

Construction

- Let F be a length-preserving, keyed function

- Construct the following MAC Π:
  - Gen: choose a uniform key k for F
  - Mack(m): output Fk(m)
  - Vrfyk(m, t): output 1 iff Fk(m)=t

- Theorem: If F is a pseudorandom function, then Π is a secure MAC

Suggestions?

- Can you construct a secure MAC for variable-length messages from a MAC for fixed-length messages?

- One natural idea:
  - Mac'k(m1, …, ml) = Mack(m1), …, Mack(ml)
  - Vrfy'k(m1, …, ml, t1, …, tl) = 1 iff
    Vrfyk(mi, ti) = 1 for all i
  - Is this secure?

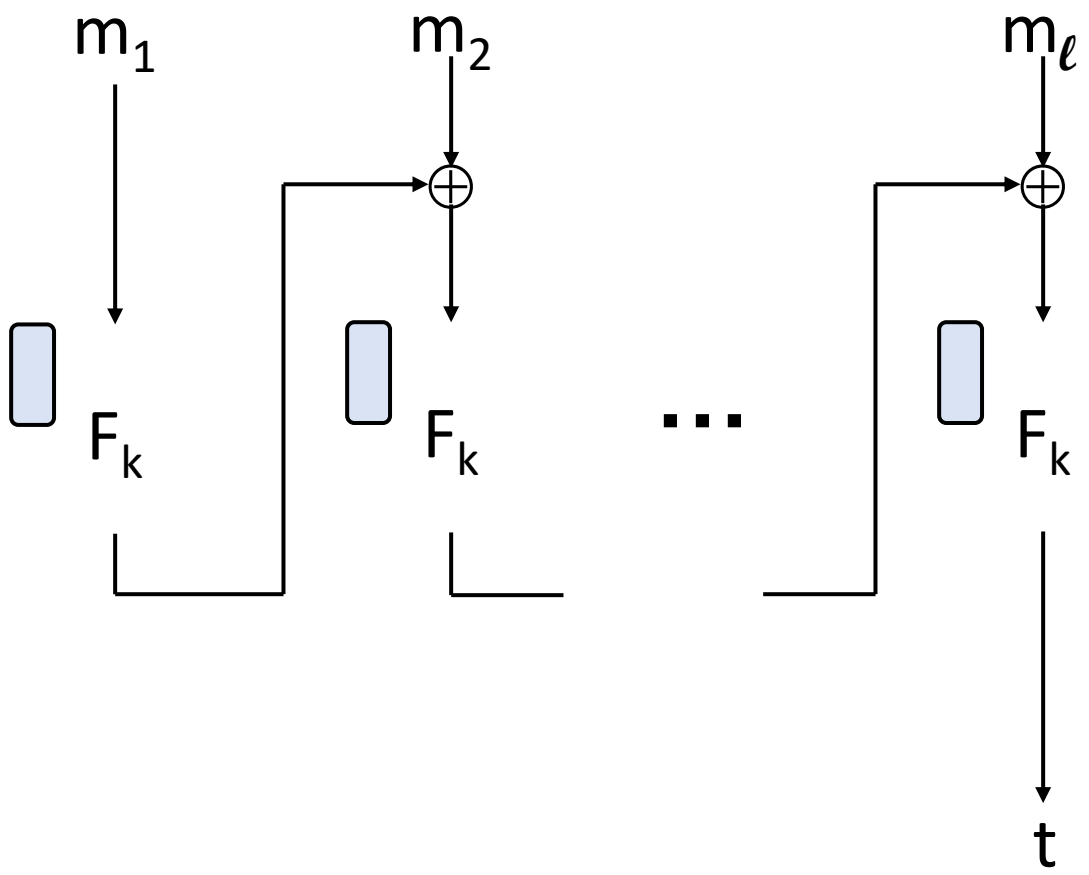
- Other suggestions?

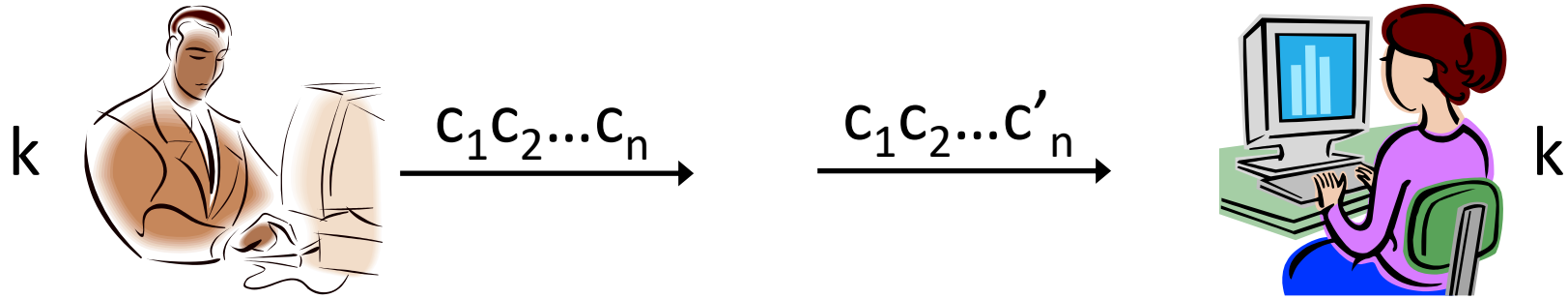
A construction

- Need to prevent (at least)
  - Block reordering
  - “Mixing-and-matching” blocks from multiple messages
  - Truncation

- One solution:
  - Mac’k(m1, …, ml) =
    r, Mack(r | l | 1 | m1), Mack(r | l | 2 | m2), …
  - Not very efficient – can we do better?

(Basic) CBC-MAC




k
$c_1c_2...c_n$
$c_1c_2...c'_n$
k
$c := (m_1m_2...m_n) \oplus k$
$m_1m_2...m'_n := (c_1c_2...c'_n) \oplus k$

CBC-MAC vs. CBC-mode

- CBC-MAC is deterministic (no IV)
  - MACs do not need to be randomized to be secure
  - Verification is done by re-computing the result
- In CBC-MAC, only the final value is output
- Both are essential for security
  - Exercise: show attacks on variants

CBC-MAC extensions

- Several ways to handle variable-length messages

- One of the simplest: prepend the message length before applying (basic) CBC-MAC
  - Can also be adapted to handle messages whose length is not a multiple of the block length