Applied Cryptography and Network Security
CS 1653

Summer 2023

Sherif Khattab

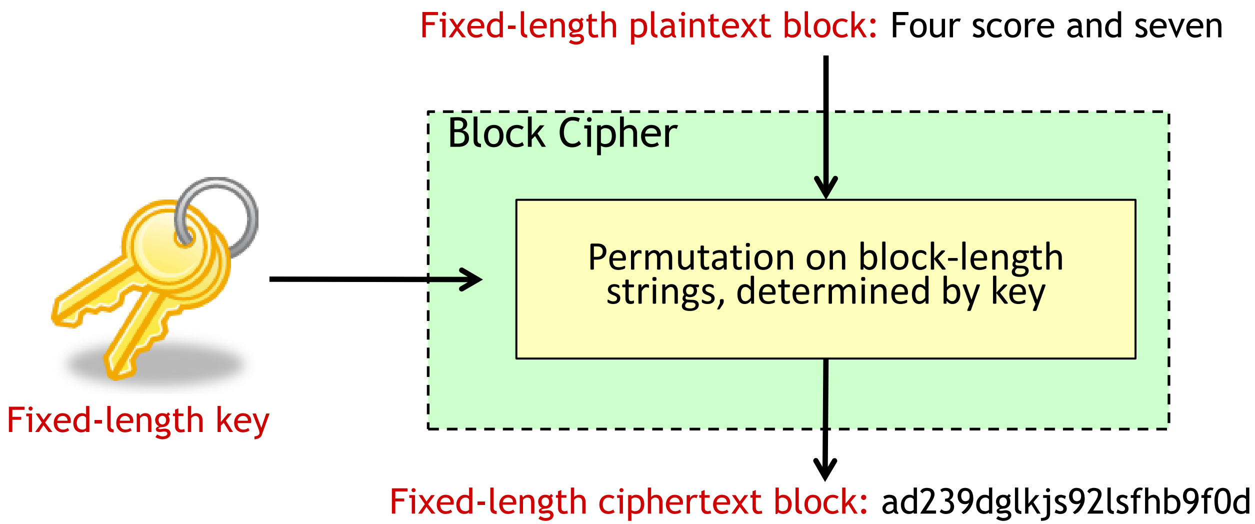
ksm73@pitt.edu

(Slides are adapted from Prof. Adam Lee's CS1653 slides.)

Announcements

- Homework 3 due this Friday @ 11:59 pm
  - will be posted tonight
- Phase 1 of Project due tomorrow @ 11:59 pm

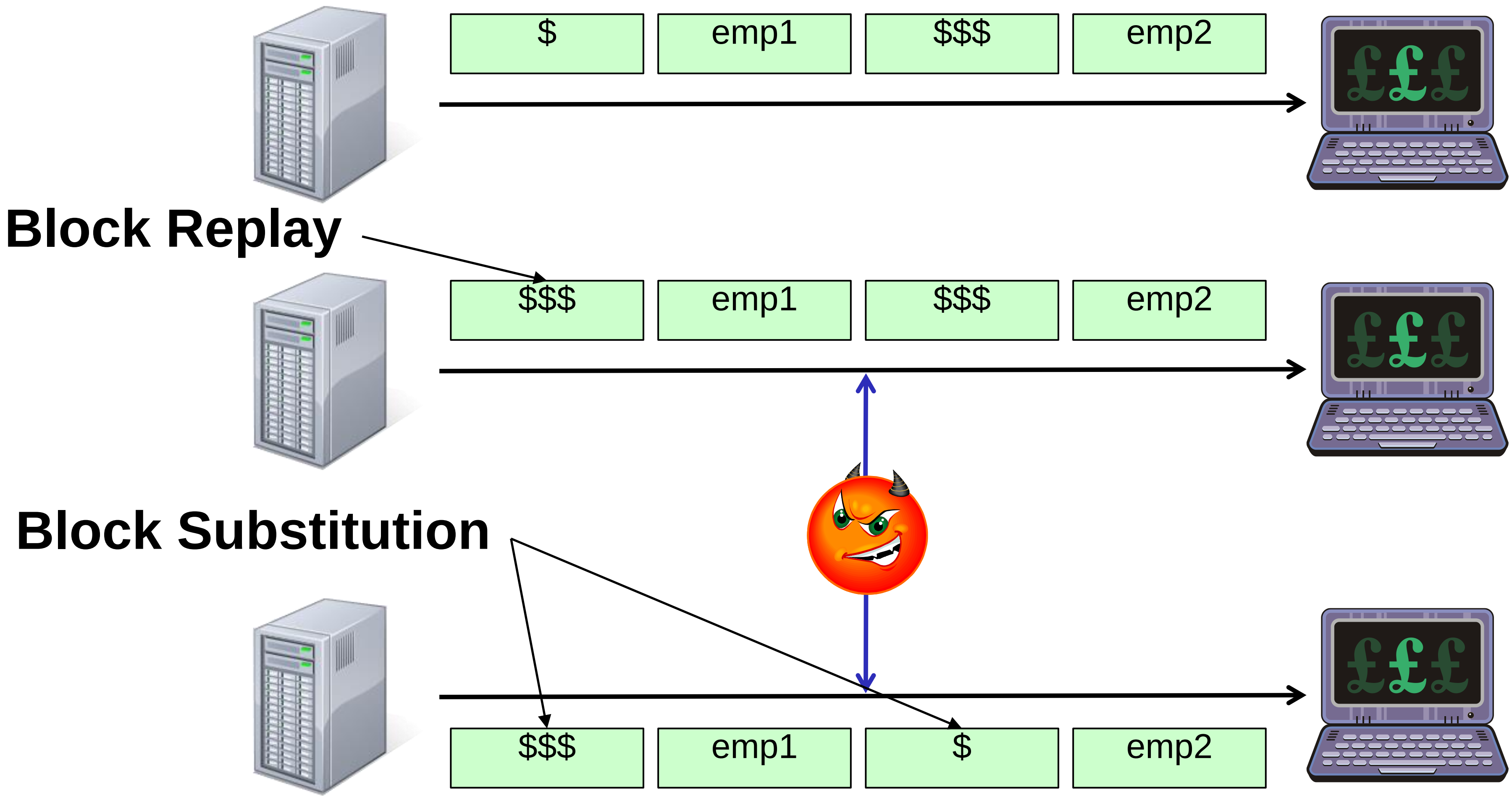
Block Cipher Modes of Operation



Question: What happens if we need to encrypt more than one block of plaintext?

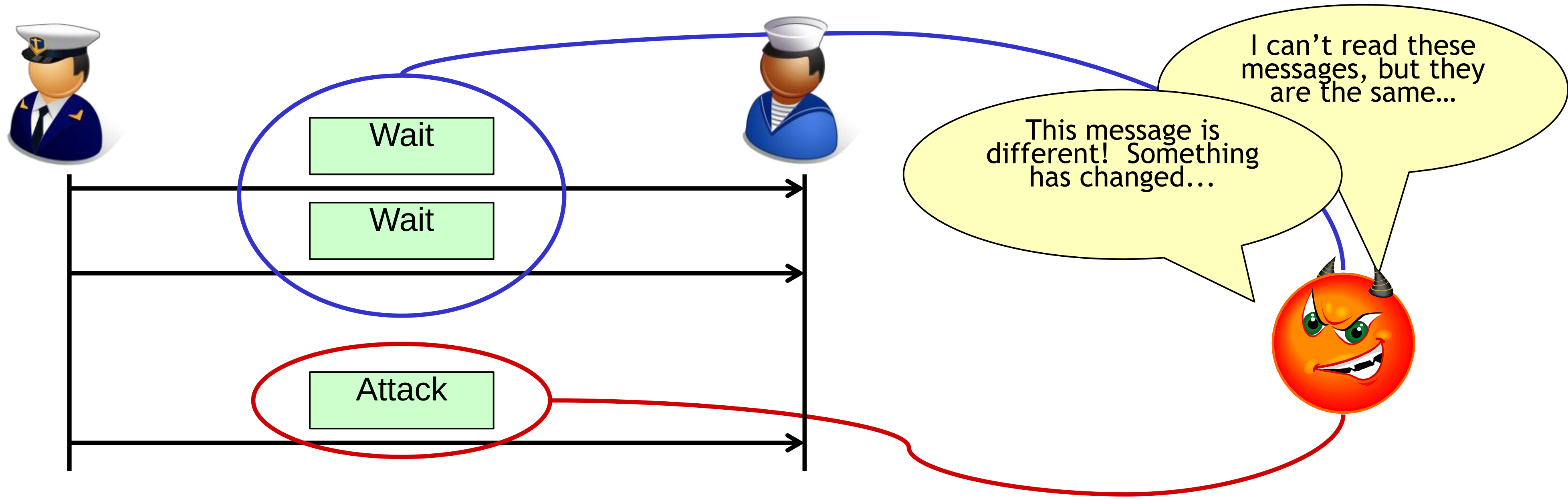
ECB mode can lead to block replay or substitution attacks

Example: Salary data transmitted using ECB



Why is the ability to build a codebook dangerous?

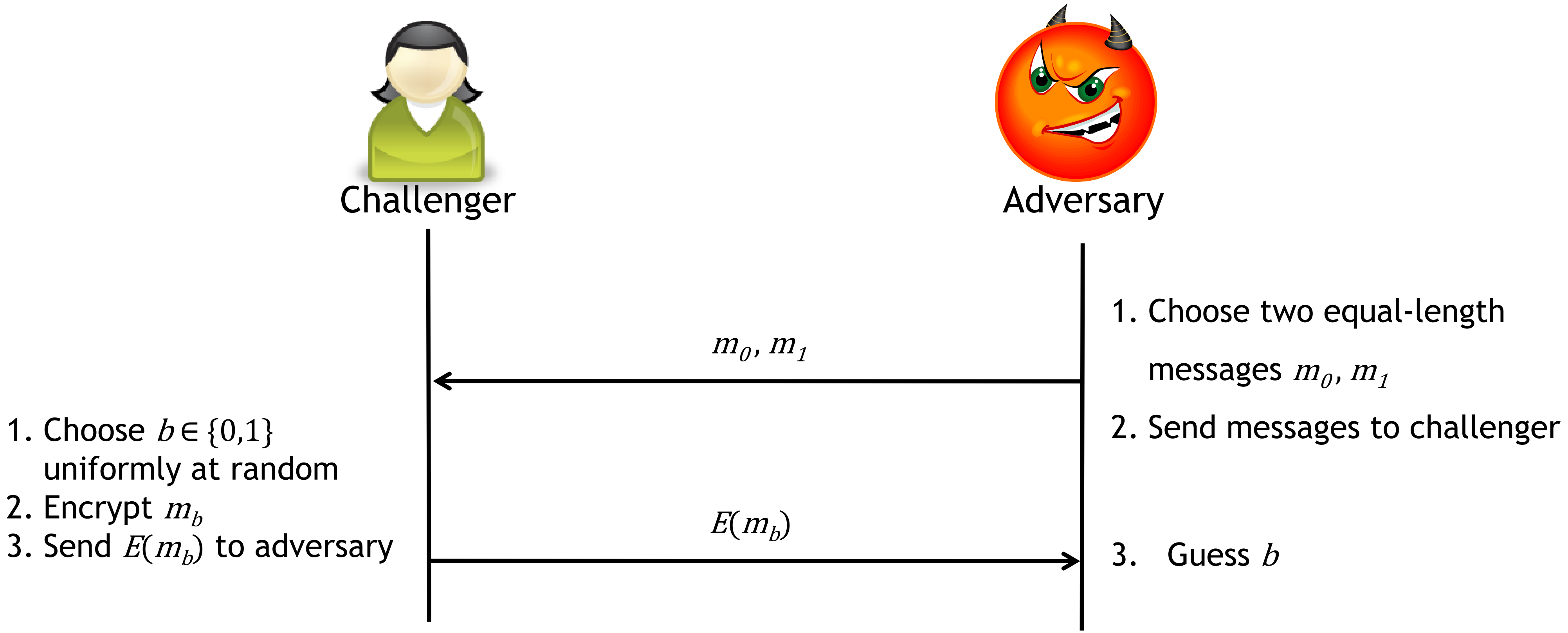
Observation: In ECB, the same block will always be encrypted the same way



To protect against this type of **guessing attack**, we need our cryptosystem to provide us with semantic security.

Semantic Security

The semantic (in)security of a cipher can be established as follows:



The adversary wins if he has a non-negligible advantage in guessing b. More concretely, he wins if $P[b' = b] > ½ + \varepsilon$.

If the adversary does not have an advantage, the cipher is said to be semantically secure.

Block ciphers in ECB mode are not semantically secure!

Question: Can you demonstrate this?

Challenger

Adversary

Adversary:

1. m_0 = "HelloHello"
   m_1 = "HelloWorld"
2. Send messages to challenger

Adversary → Challenger: m_0, m_1

Challenger:

1. Choose $b \in \{0, 1\}$ uniformly at random
2. Encrypt m_b using ECB
3. Send $E(m_b)$ to adversary

Challenger → Adversary: $E(m_b)$

Adversary:

3. If first block = second block, then $b = 0$. Otherwise, $b = 1$.

$P[b' = b] = 1$

This can also be thought of as a "**covert channel**" attack

Cipher Block Chaining (CBC) addresses problems in ECB

In CBC mode, each plaintext block is XORed with the previous ciphertext block prior to encryption

$$C_i = \mathrm{E}_k(P_i \oplus C_{i-1})$$

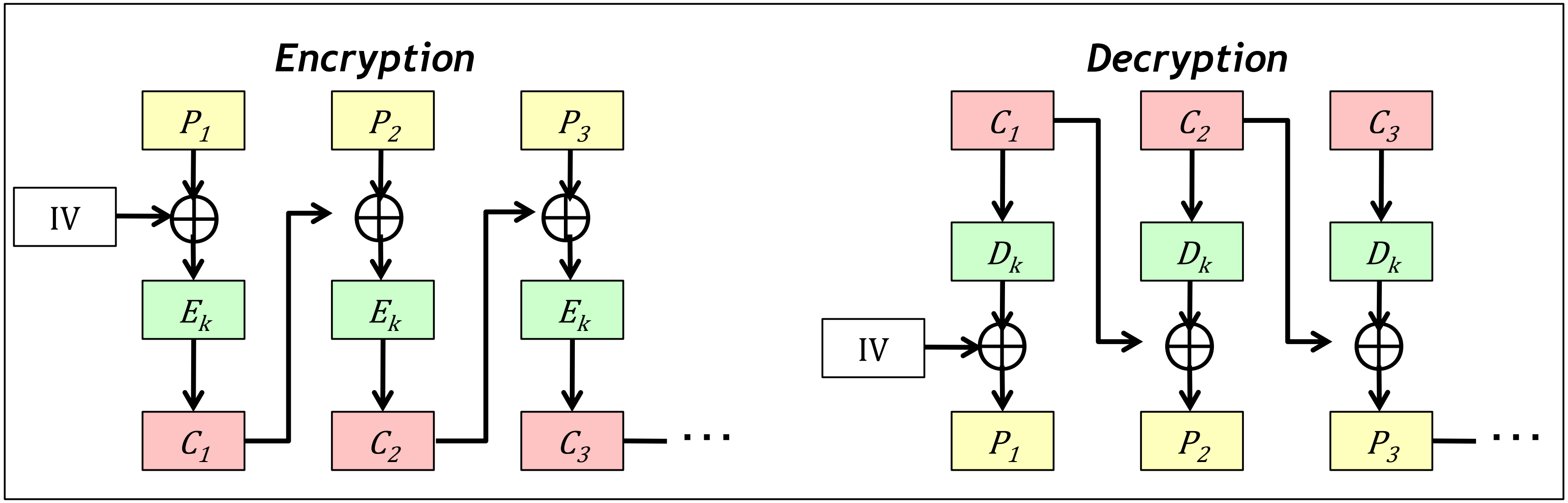
$$P_i = C_{i-1} \oplus \mathrm{D}_k(C_i)$$

Need to encrypt a random block to get things started

- This initialization vector needs to be random, but not secret (Why?)

CBC eliminates block replay attacks

- Each ciphertext block depends on previous block



Semantic security, redux

Note that the adversary's "trick" does not work anymore (Why?)

$$c_{01} = \mathrm{E}(\mathrm{IV} \oplus m_{01})$$

$$c_{02} = \mathrm{E}(c_{01} \oplus m_{02})$$

Essentially, the IV randomizes the output of the game, even if it is played over multiple rounds

Challenger

Adversary

Adversary:

1. m_0 = "HelloHello"
   m_1 = "HelloWorld"
2. Send messages to challenger

Adversary → Challenger: m_0, m_1

Challenger:

1. Choose $b \in \{0,1\}$ uniformly at random
2. Generate a random IV
3. Encrypt m_b using CBC
4. Send $E(m_b)$ to adversary

Challenger → Adversary: $E(m_b)$

Adversary:

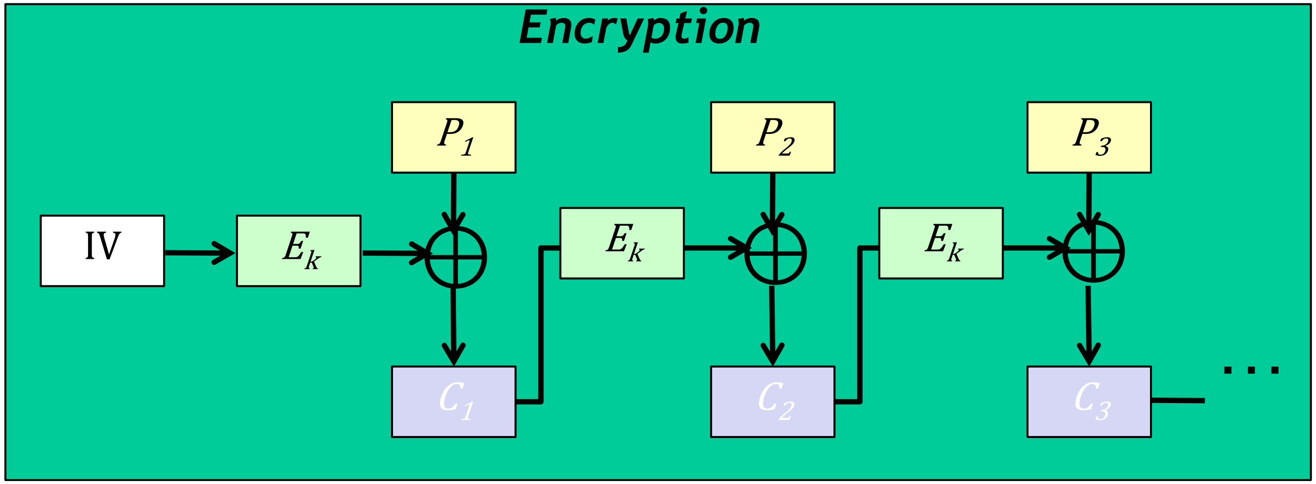
3. If first block = second block, then $b = 0$. Otherwise, $b = 1$.

Cipher Feedback Mode (CFB)

CFB mode:

$$C_i = P_i \oplus \mathrm{E}_k(C_{i-1})$$

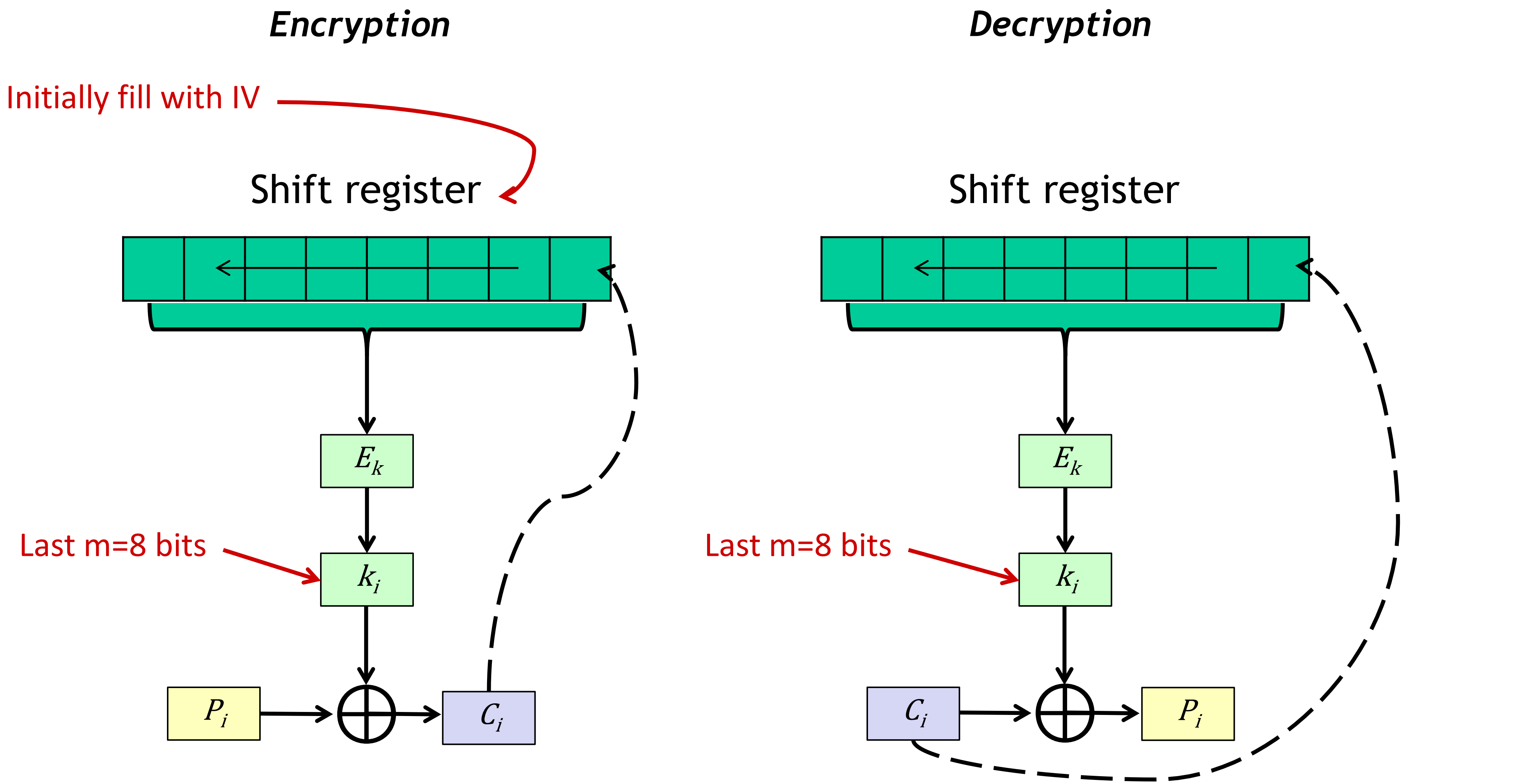
$$P_i = C_i \oplus \mathrm{E}_k(C_{i-1})$$



- CFB can be used to develop an ***m*-bit cipher** based upon an n-bit block cipher
    - $m \leq n$
    - using a **shift-register approach**
- This is great, since we don't need to wait for n bits of plaintext to encrypt!
    - ***Example:*** Typing at a terminal

Using an n-bit cipher to get an m-bit cipher ($m < n$)

Cipher Feedback Mode (CFB) can be used to construct a self-synchronizing stream cipher from a block cipher

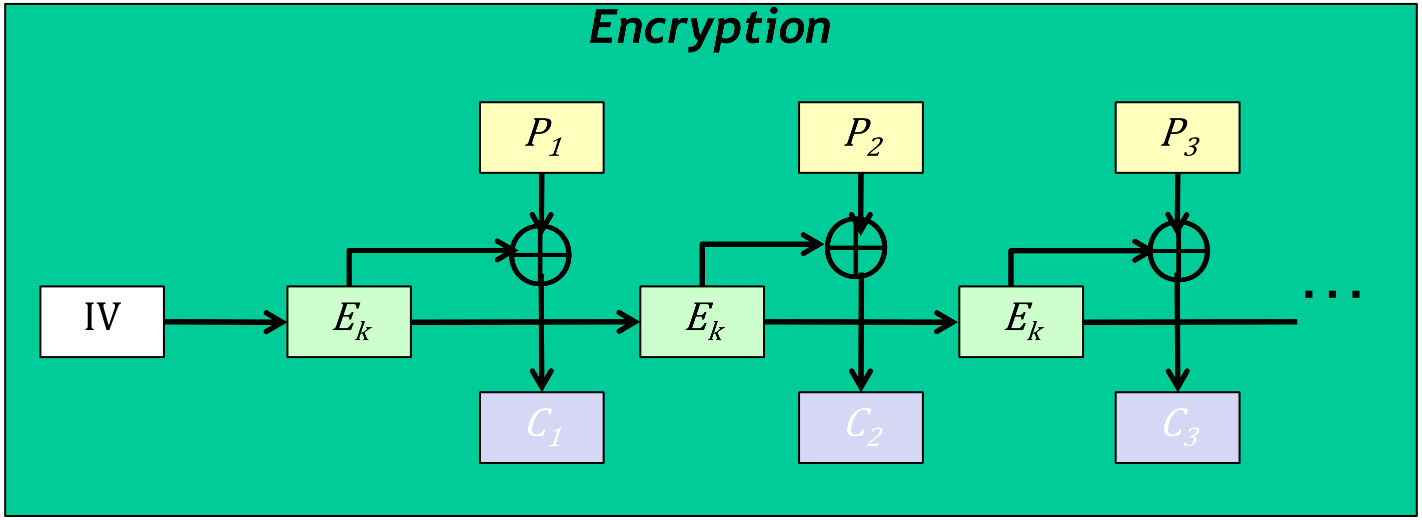


Output Feedback Mode (OFB)

How does OFB work?

$$C_i = P_i \oplus S_i,\ \ S_i = \mathrm{E}_k(S_{i-1})$$

$$P_i = C_i \oplus S_i,\ \ S_i = \mathrm{E}_k(S_{i-1})$$



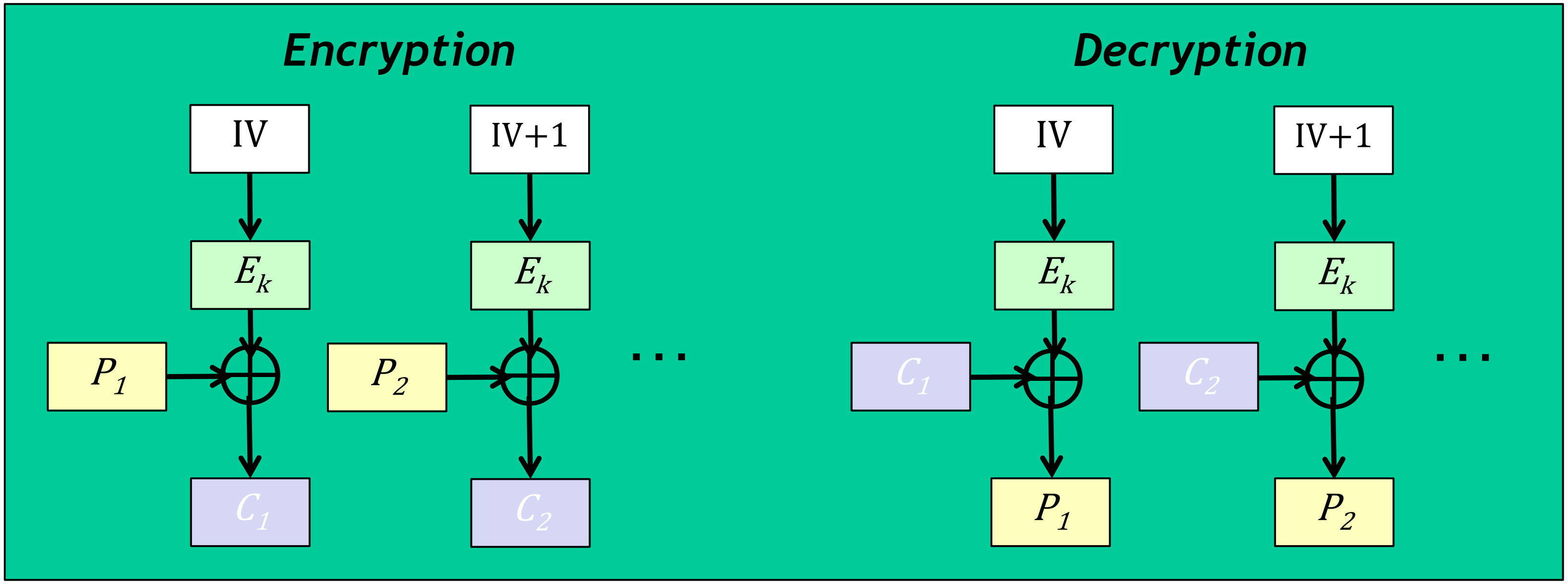
Key Stream generated independently of plaintext

Benefit: Key stream generation can occur offline

Can be used to construct a synchronous stream cipher from a block cipher

Pitfall: Loss of synchronization is a killer…

Counter mode (CTR)



CTR mode generates a key stream independently of the data

Pros:

We can do the expensive cryptographic operations offline

Encryption/decryption is just an XOR

It is possible to encrypt/decrypt starting anywhere in the message

Cons:

Don’t use the same (key, IV) for different files (Why?)

CTR mode has some interesting applications

Example: Accessing a large file or database

Operation: Read block number n of the file

CTR: One encryption operation is needed

$p_n = c_n \oplus \mathrm{E}(\mathrm{IV} + n)$

CBC: One decryption operation is needed

$p_n = c_{n-1} \oplus \mathrm{D}(c_n)$

In most symmetric key ciphers encryption and decryption have the same complexity

Operation: Update block k of n

If n is large, this is problematic…

CTR: One encryption operation is needed

$c_k = p_k \oplus \mathrm{E}(\mathrm{IV} + k)$

What about CBC?

First, we need to decrypt all blocks after k ($n - k$ decryptions)

Then, we need to encrypt blocks k through n ($n - k + 1$ encryptions)

Operation: Encrypt all n blocks of a file on a machine with c cores

CTR: $O(n / c)$ time required, as cores can operate in parallel

CBC: $O(n)$ time required on one core…

So… Which mode of operation should I use?

Do not use ECB!

Unless you are encrypting short, random data (e.g., a cryptographic key)

Encrypting streams of characters entered at, e.g., a text terminal?

CFB (usually 8-bit CFB) is the best choice

Error prone environments? (high chance if bit errors)

OFB or CTR is probably your best choice

Use CBC if either:

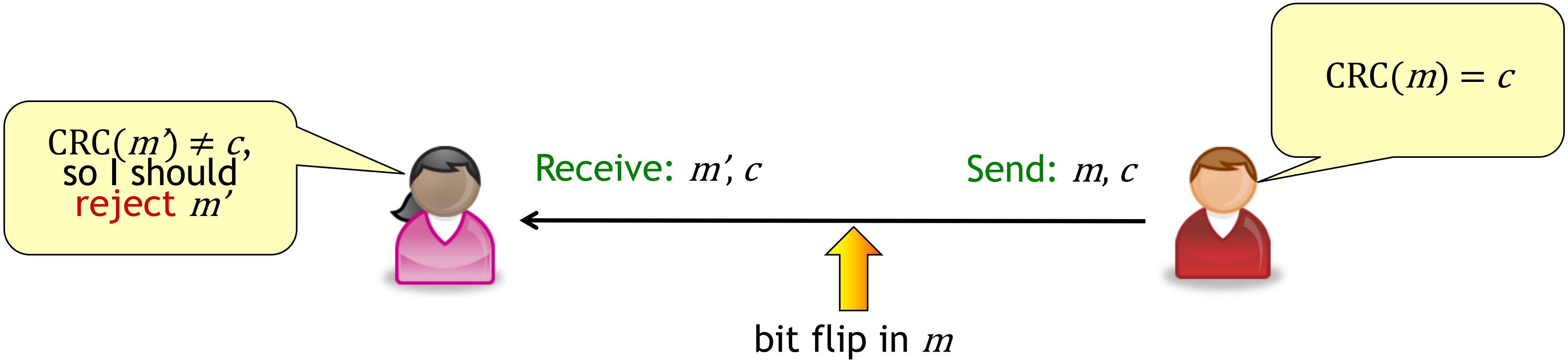
You are encrypting files, since there are rarely errors on storage devices

You are dealing with a software implementation

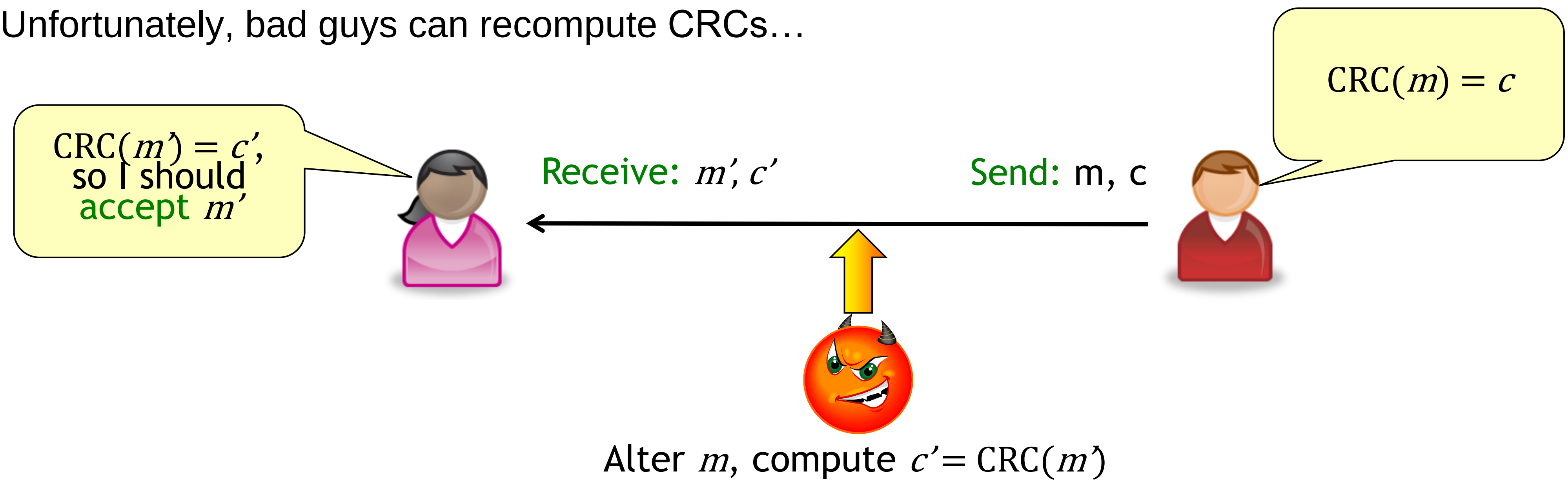
Want more information? See chapter 9 of *Applied Cryptography.*

Encryption does not guarantee integrity/authenticity

(Cyclic Redundancy Check) CRC can be used to detect random errors in a message



Unfortunately, bad guys can recompute CRCs…



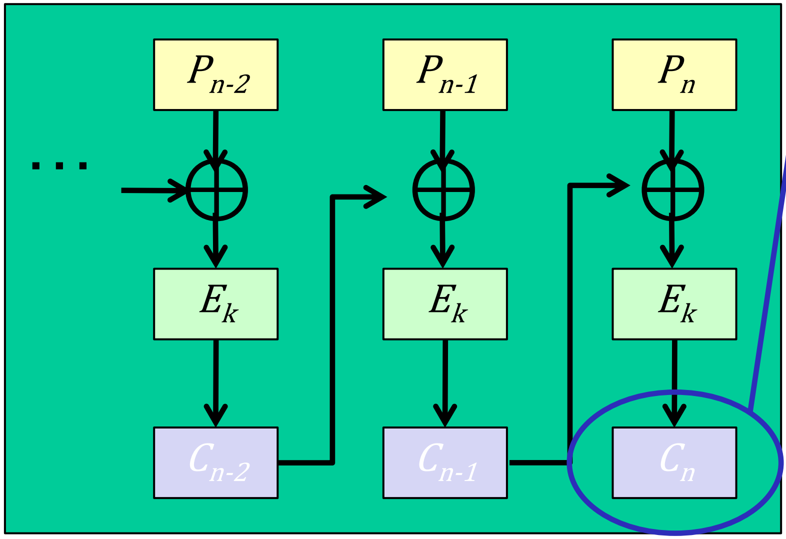
Solution: Cryptographic message authentication codes (MACs)

The CBC residue of an encrypted message can be used as a cryptographic MAC

How does this work?

- Use a block cipher in CBC mode to encrypt m using the shared key k
- Save the CBC residue r
- Transmit m and r to the remote party
- The remote party recomputes and verifies the CBC residue of m

The last block of a CBC encryption is called the CBC residue



Why does this work?

- Malicious parties can still manipulate m in transit
- However, without k, they cannot compute the corresponding CBC residue!

The bad news: Encrypting the whole message is expensive!

How can we guarantee confidentiality **and** integrity?

Does this mean using CBC encryption gives us confidentiality and integrity at the same time?

Unfortunately, it does not 😟

Truncation attack is possible if same key used for encryption and integrity!

To use CBC for confidentiality and integrity, we need two keys

Encrypt the message M using k_1 to get ciphertext $C_1 = \{c_{11}, ..., c_{1n}\}$

Encrypt M using k_2 to get $C_2 = \{c_{21}, ..., c_{2n}\}$

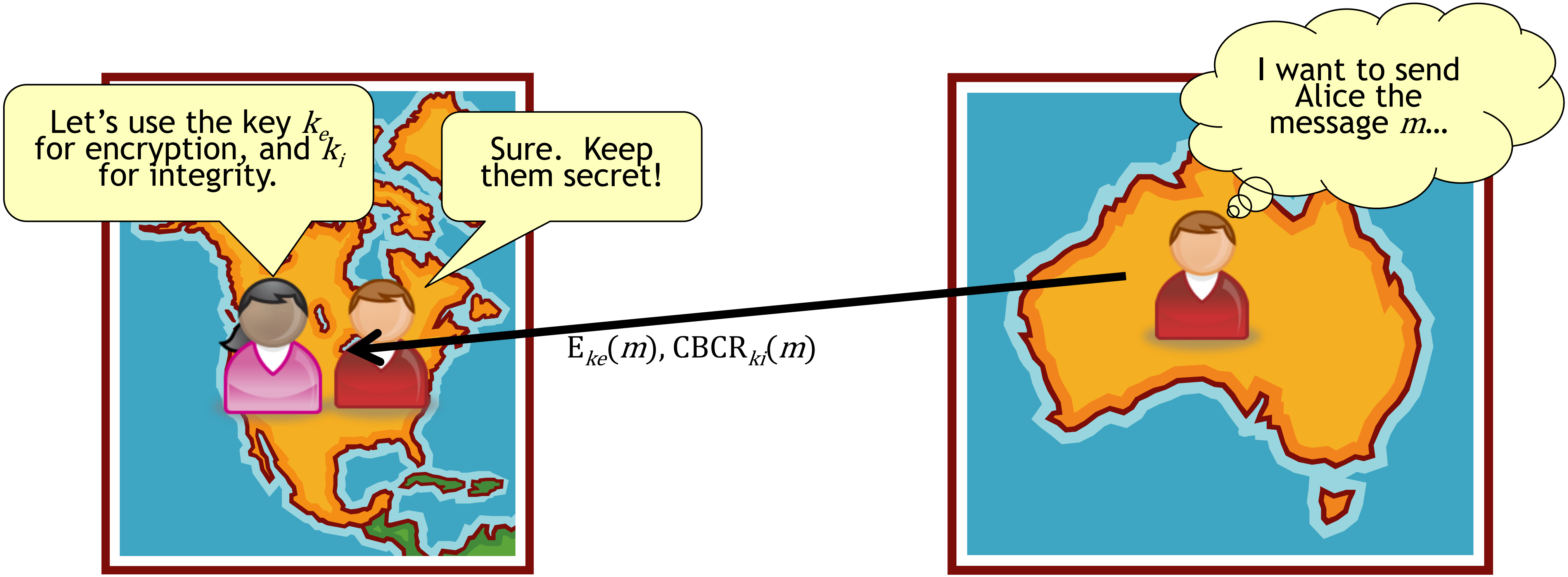
Transmit $\langle C_1, c_{2n} \rangle$

But wait, isn't that expensive?

Fix #1: Exploit parallelism if there is access to multiple cores

Fix #2: Faster hash-based MACs (next!)

Putting it all together…



All is well?

Ok, so symmetric-key cryptography can protect the confidentiality and integrity of our communications

So, the security problem is solved, right?

Unfortunately, symmetric key cryptography doesn't solve everything…

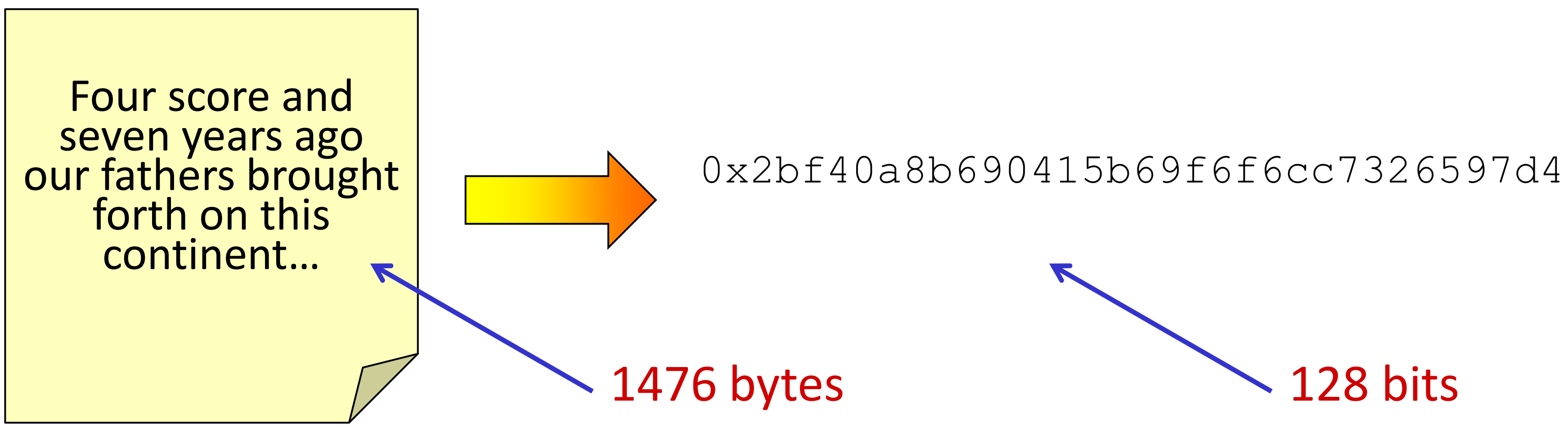
1. How do we get secret keys for everyone that we want to talk to?
2. How can we update these keys over time?

Later: Public key cryptography will help us solve problem 1

Even later in the semester, we'll look at key exchange protocols that help with problem 2

What is a hash function?

Definition: A hash function is a function that maps a variable-length input to a fixed-length code



Hash functions are sometimes called message digest functions

- SHA (e.g., SHA-1, SHA-256, SHA-3) stands for the secure hash algorithm
- MD5 stands for message digest algorithm (version 5)

To be useful cryptographically, a hash function needs to have a “randomized” output

For example:

- Given a large number of inputs, any given bit in the corresponding outputs should be set about half of the time
- Any given output should have half of its bits set on average
- Given two messages m and m' that are very closely related, $\mathrm{H}(m)$ and $\mathrm{H}(m')$ should appear completely uncorrelated

Informally: The output of an m-bit hash function should appear as if it was created by flipping m unbiased coins

Theoretical cryptographers sometimes use a more formalized notion of random oracles to replace hash functions when analyzing security protocols

More formally, cryptographic hash functions should have the following three properties

Assume that we have a hash function $H : \{0,1\}^* \rightarrow \{0,1\}^m$

What does infeasible mean?

1. Preimage resistance: Given a hash output value z, it should be infeasible to calculate a message x such that $H(x) = z$

   i.e., H is a one-way function

   Ideally, computing x from z should take $O(2^m)$ time

2. Second preimage resistance: Given a message x, it is infeasible to calculate a second message y such that $H(x) = H(y)$

   Note that this attack is always possible given infinite time (Why?)

   Ideally, this attack should take $O(2^m)$ time

3. Collision resistance: It is infeasible to find two messages x and y such that $H(x) = H(y)$

   Ideally, this attack should take $O(2^{m/2})$ time

   Why only $O(2^{m/2})$?

The Birthday Paradox!

The gist: If there are more than 23 people in a room, there is a better than 50% chance that two people have the same birthday

Wait, what?

366 possible birthdays

To solve: Find probability p_n that n people all have *different* birthdays,

then compute $1 - p_n$

$$p_n = \frac{365}{366}\frac{364}{366}\frac{363}{366}\cdots\frac{367-n}{366}$$

If $\mathrm{n} = 22$, $1 - p_n \approx 0.475$

If $\mathrm{n} = 23$, $1 - p_n \approx 0.506$

Note: The value of n can be approximated as $1.1774 \times \sqrt{N} = 1.1774 \, x \, \sqrt{366} \approx 22.525$

What does this have to do with hash functions?!

Note that "birthday" is just a function b : person → date

Goal: How many inputs x to the function b do we need to consider to find x_i, x_j such that $\mathrm{b}(x_i) = \mathrm{b}(x_j)$?

We're looking for collisions in the birthday function!

Now, a hash is a function $\mathrm{H} : \{0, 1\}^* \rightarrow \{0, 1\}^{\mathrm{m}}$

Note: H has 2^{m} possible outputs

So, using our approximation from the last slide, we'd need to examine about $1.1774 \times \sqrt{2^m} = 1.1774 \times 2^{\frac{m}{2}} = O(2^{\frac{m}{2}})$ inputs to find a collision!