Intro to Crypto

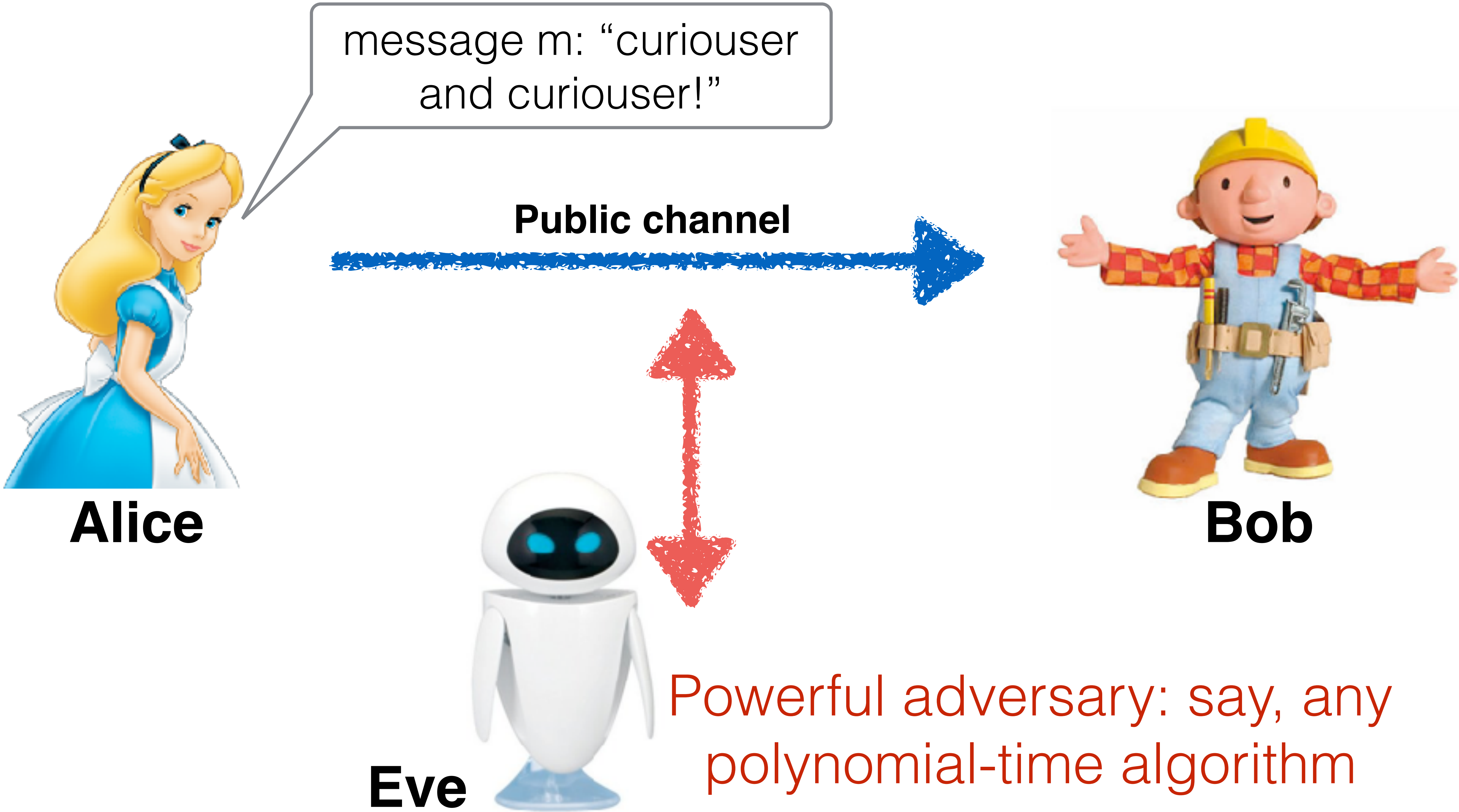
With material from: Michelle Mazurek, David Brumley, Dan Boneh

https://en.wikipedia.org/wiki/File:Bletchley_Park_Bombe4.jpg

Crypto is everywhere

- Secure comms:
  - Web traffic (HTTPS)
  - Wireless traffic (802.11, WPA2, GSM, Bluetooth)
- Files on disk: Bitlocker, FileVault
- User authentication: Kerberos
- … and much more

Overall goal: Protect communication

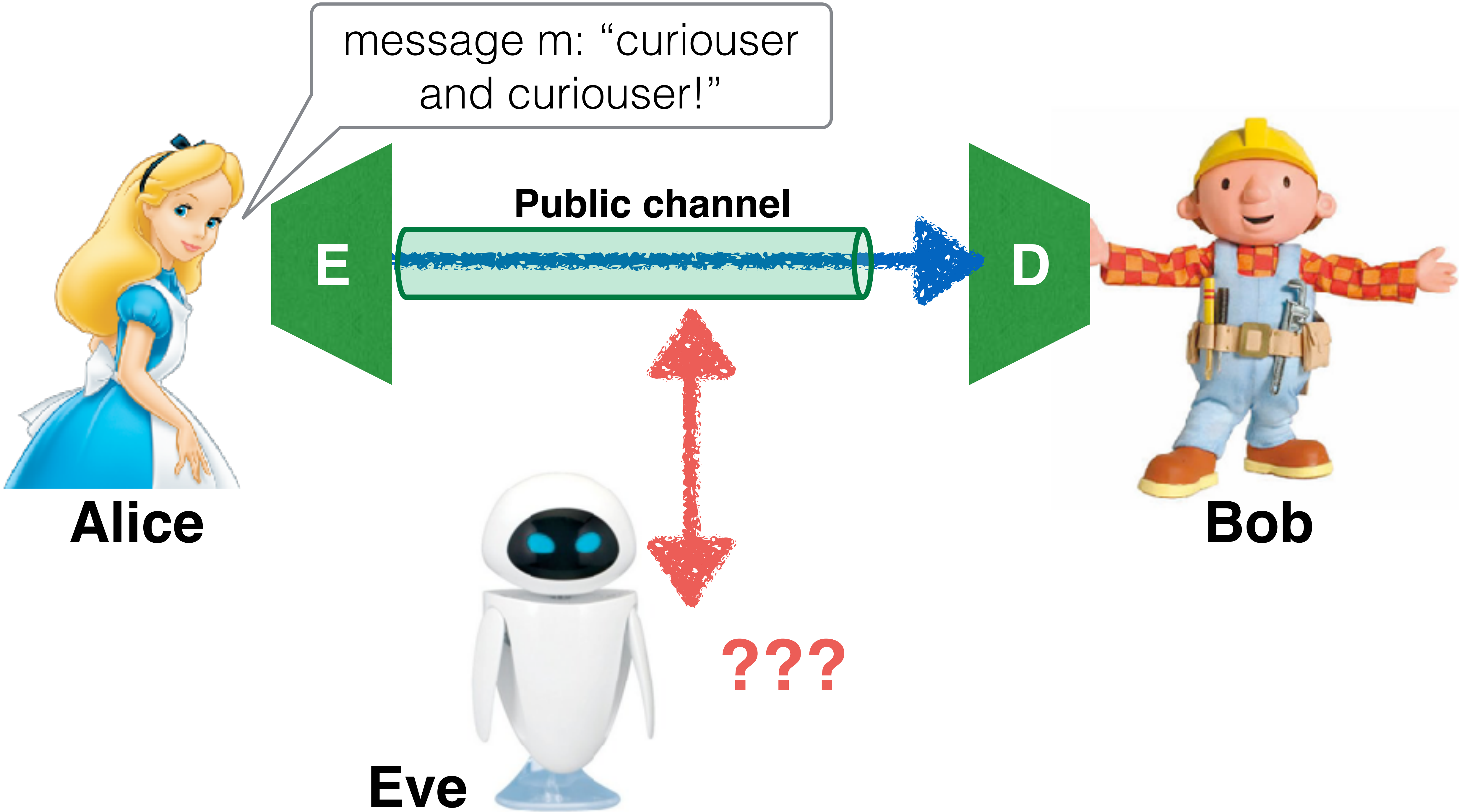


Security goals

- Privacy
- Integrity
- Authentication

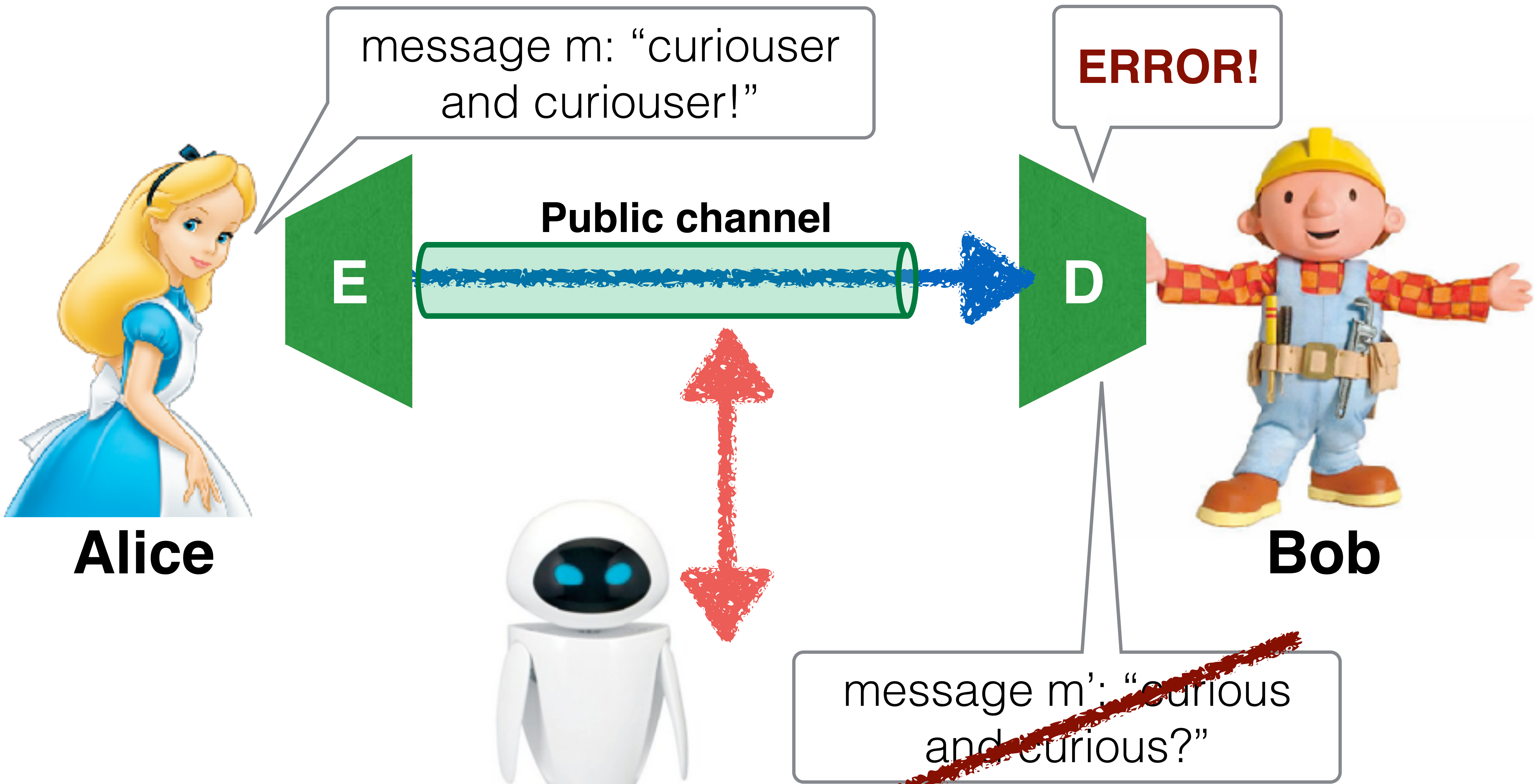
Goal: Privacy

Eve should not be able to learn m. Not even one bit!



Goal: Integrity

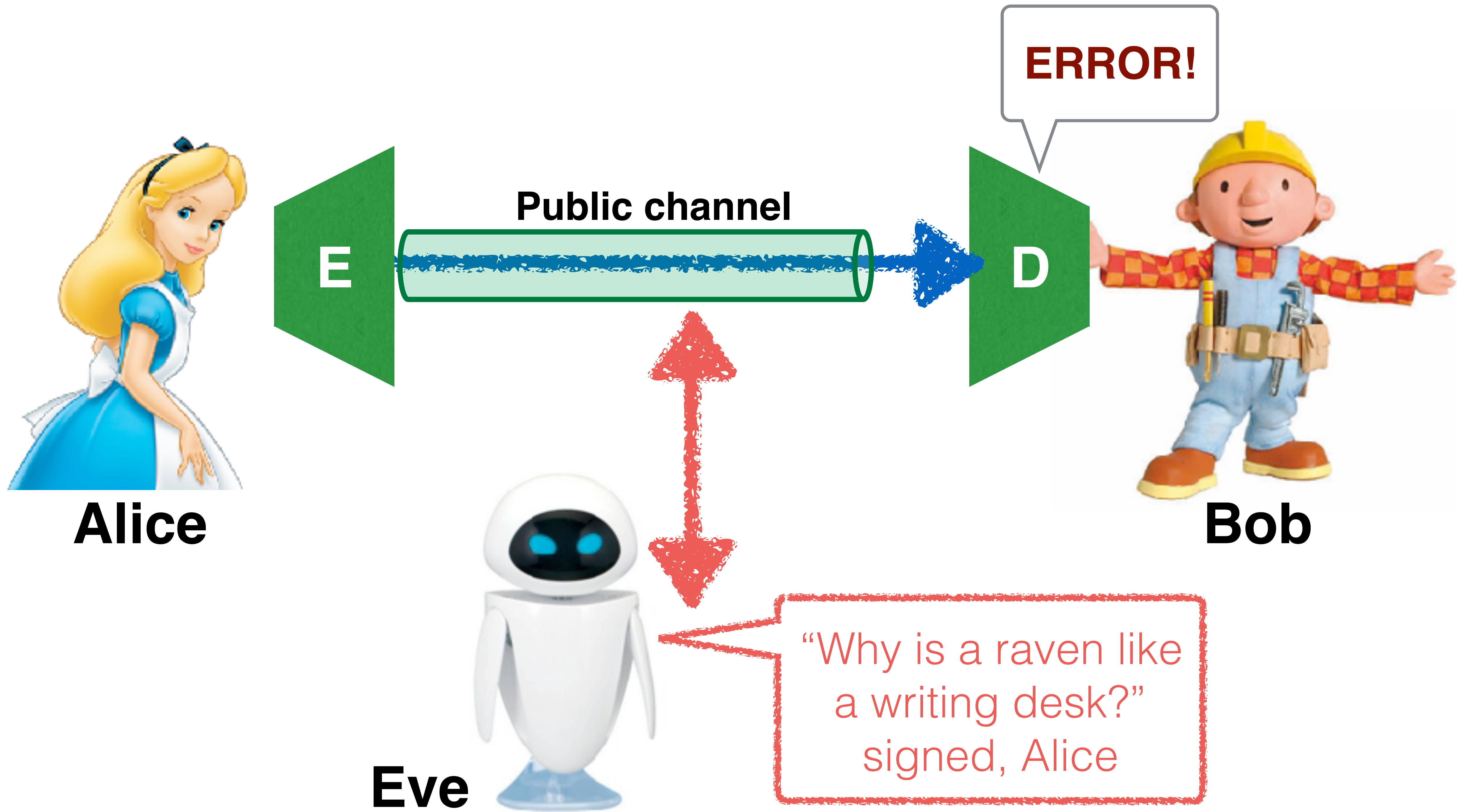
Eve should not be able to alter *m* without detection.



Works regardless of whether Eve knows the contents of m!

Goal: Authenticity

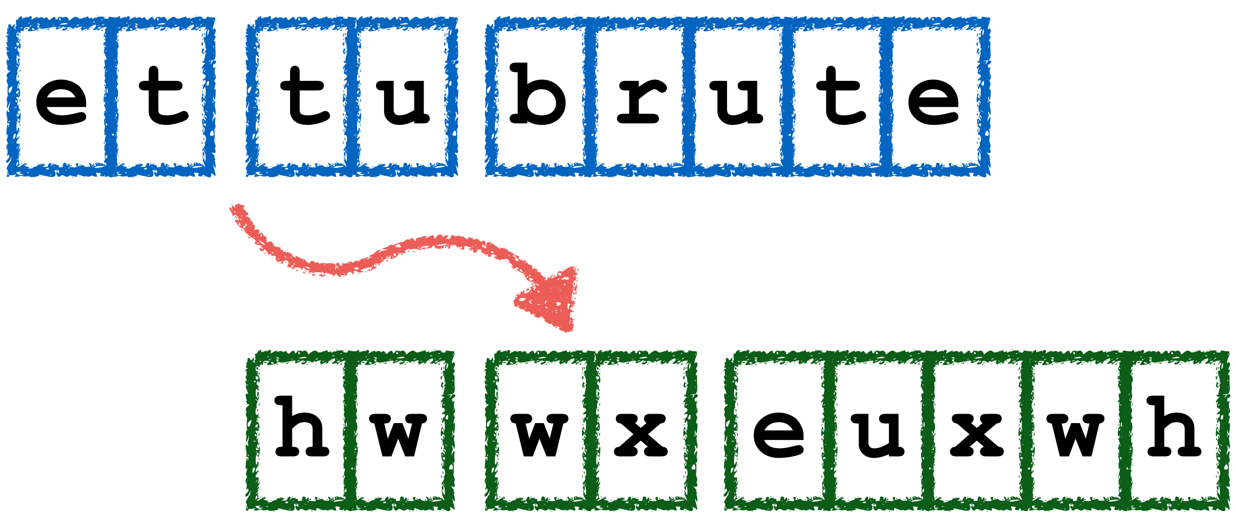
Eve should not be able to forge messages as Alice



History of Cryptography

Caesar cipher

- Also called shift or substitution cipher
- Classic: m + 3
    - Others: ROT13, etc.

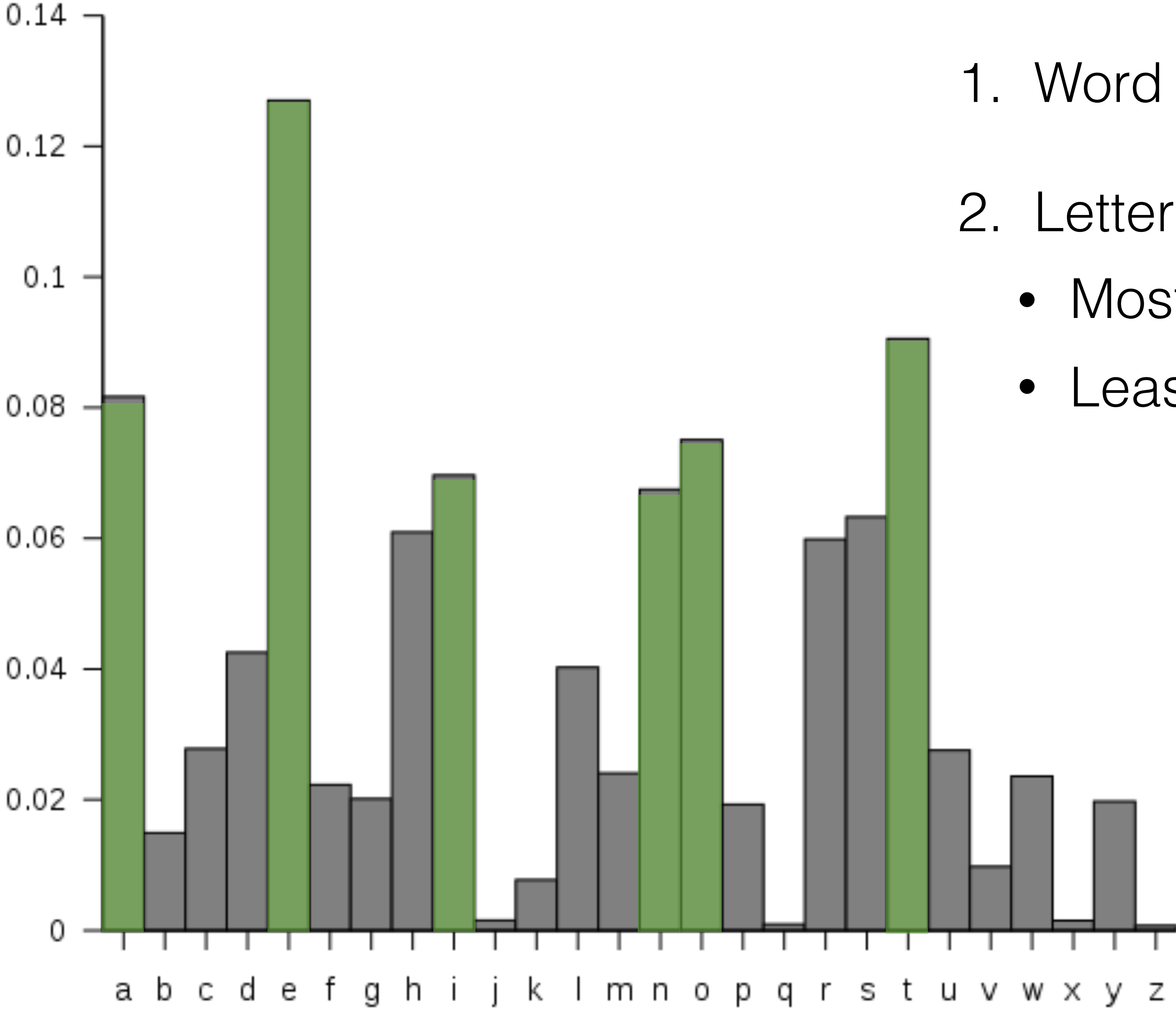


Julius Caesar
100 BC- 44 BC

How would you **attack** this cipher?

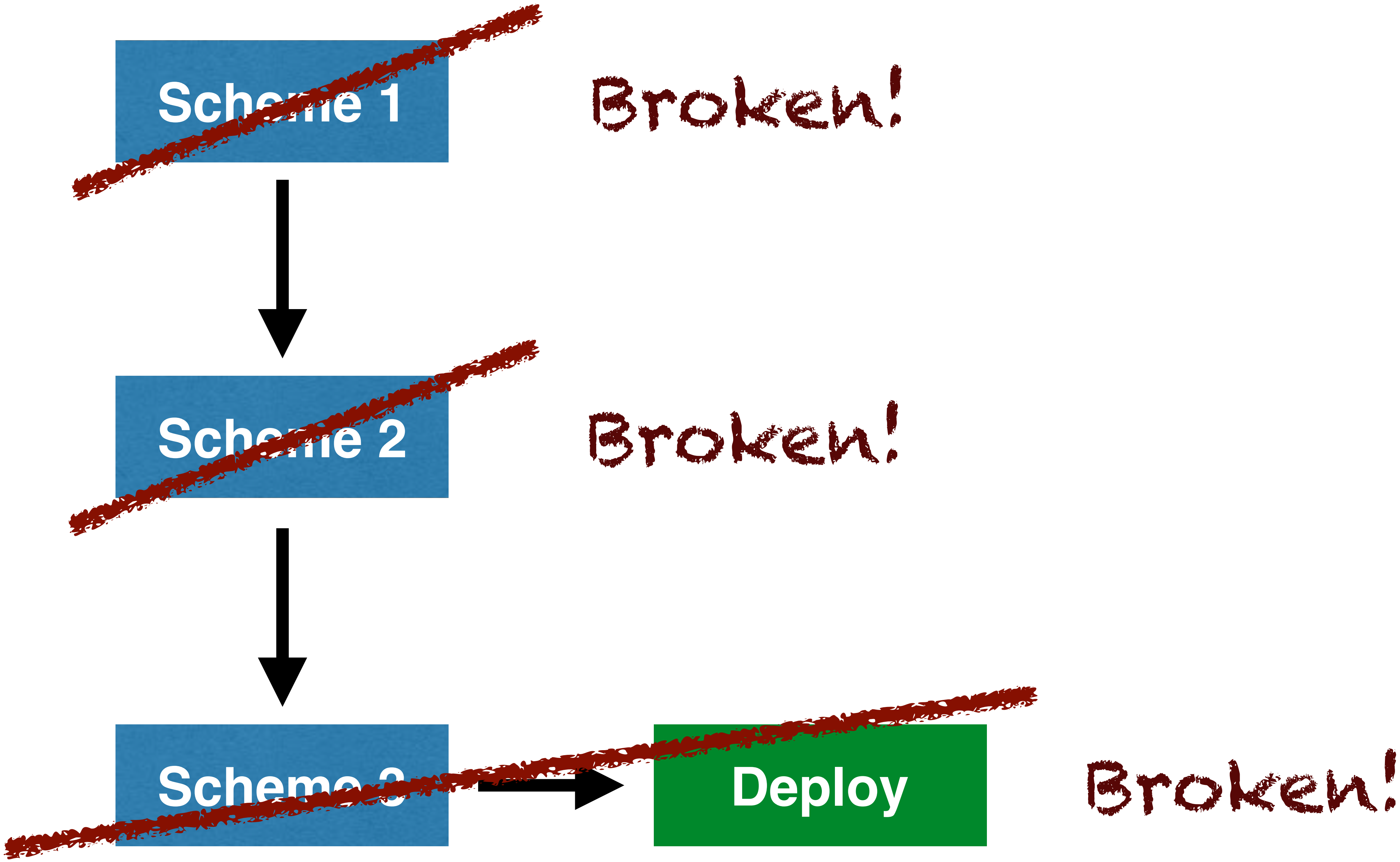
Jvl mlwclk yr jvl owmwez twp yusl w zyduo pjdcluj mqil zydkplmr. Hdj jvlz tykilc vwkc jy mlwku jvl wkj yr vwsiquo, tvqsv vlmflc mlwc jvlg jy oklwjulpp. Zyd vwnl jvl fyjlujqwm jy cy jvl pwgl. Zydk plsklj fwpptykc qp: JYWPJ

How did you do it?



1. Word frequency
2. Letter frequency
   - Most common: e,t,a,o,i,n
   - Least common: j,x,q,z

Classically: Iterative design

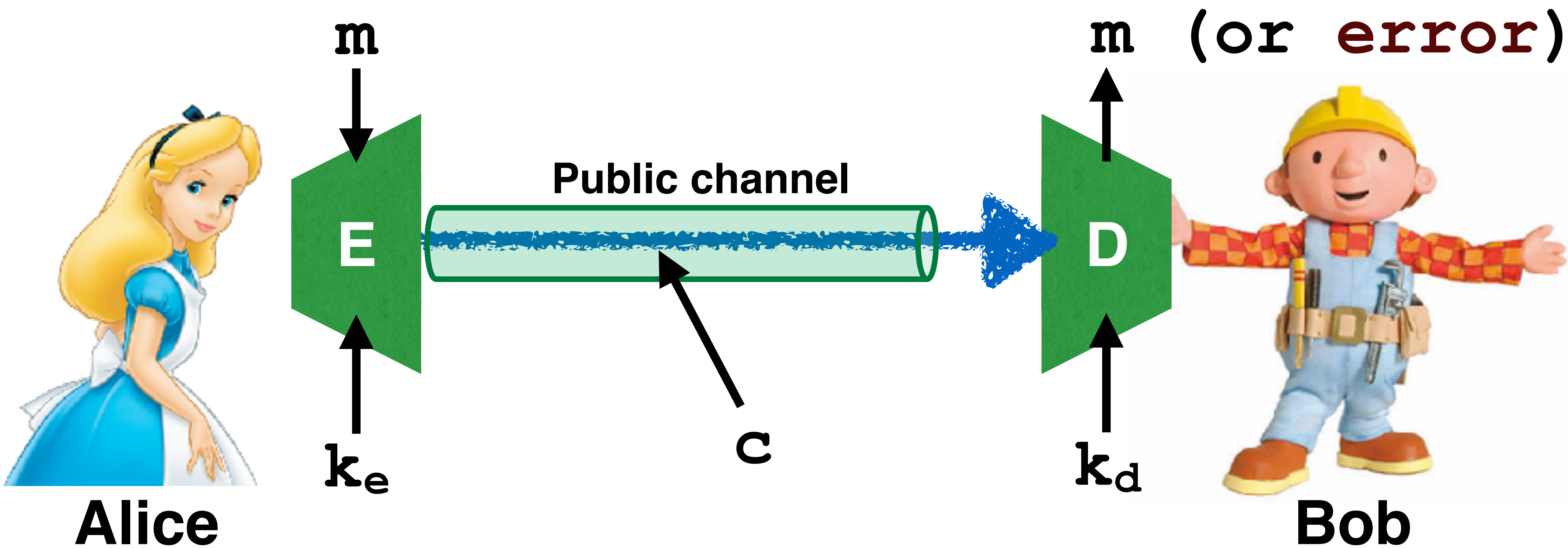


No way to prove security. How to know when broken?

Claude Shannon and Information Theory (1945)

- Formally define:
  - Security goals
  - Adversary models
  - Security of a system w.r.t. goals
- Beyond iterated design: Proof!

Defining a cryptosystem



m = message (aka "plaintext") (*message space* M)
c = ciphertext (*cipher space* C)
E = encryption algorithm
D = decryption algorithm
k_e = encryption key (*key space* K)
k_d = decryption key (*key space* K)

Defining a cryptosystem, ctd

- Three polynomial-time algorithms:
  - KeyGen(L): Returns random key of length L.
  - $E(k_e,m)$: Encrypts m with k_e, returns c in C
  - $D(k_d,c)$: Decrypts c with k_d, returns m in M
- Correctness condition:

$$\forall m \in M, k \in K : D(k, E(k,m)) = m$$

Attacker models

In order of weakest -> strongest

- Known ciphertext attack (KCA)
  - aka "Ciphertext only attack" (COA)
- Known plaintext attack (KPA)
  - Have one matching pair
- Chosen plaintext attack (CPA)
  - Encryption oracle
- Chosen ciphertext attack (CCA)
  - Decryption oracle

Matters if you are sending multiple messages with the same key!

One-Time Pad

Miller (1882) and Vernam (1917)

$$E(k, m) = k \oplus m = c$$

$$D(k, c) = k \oplus c = m$$

M = C = K = $\{0,1\}^n$

	m:	0	1	1	0	1	1	0	1
$\oplus$	k:	1	1	0	1	0	0	0	1
	c:	1	0	1	1	1	1	0	0
$\oplus$	k:	1	1	0	1	0	0	0	1
	m:	0	1	1	0	1	1	0	1

Case study: One-time pad

One-Time Pad

Miller (1882) and Vernam (1917)

$$E(k, m) = k \oplus m = c$$
$$D(k, c) = k \oplus c = m$$

M = C = K = $\{0,1\}^n$

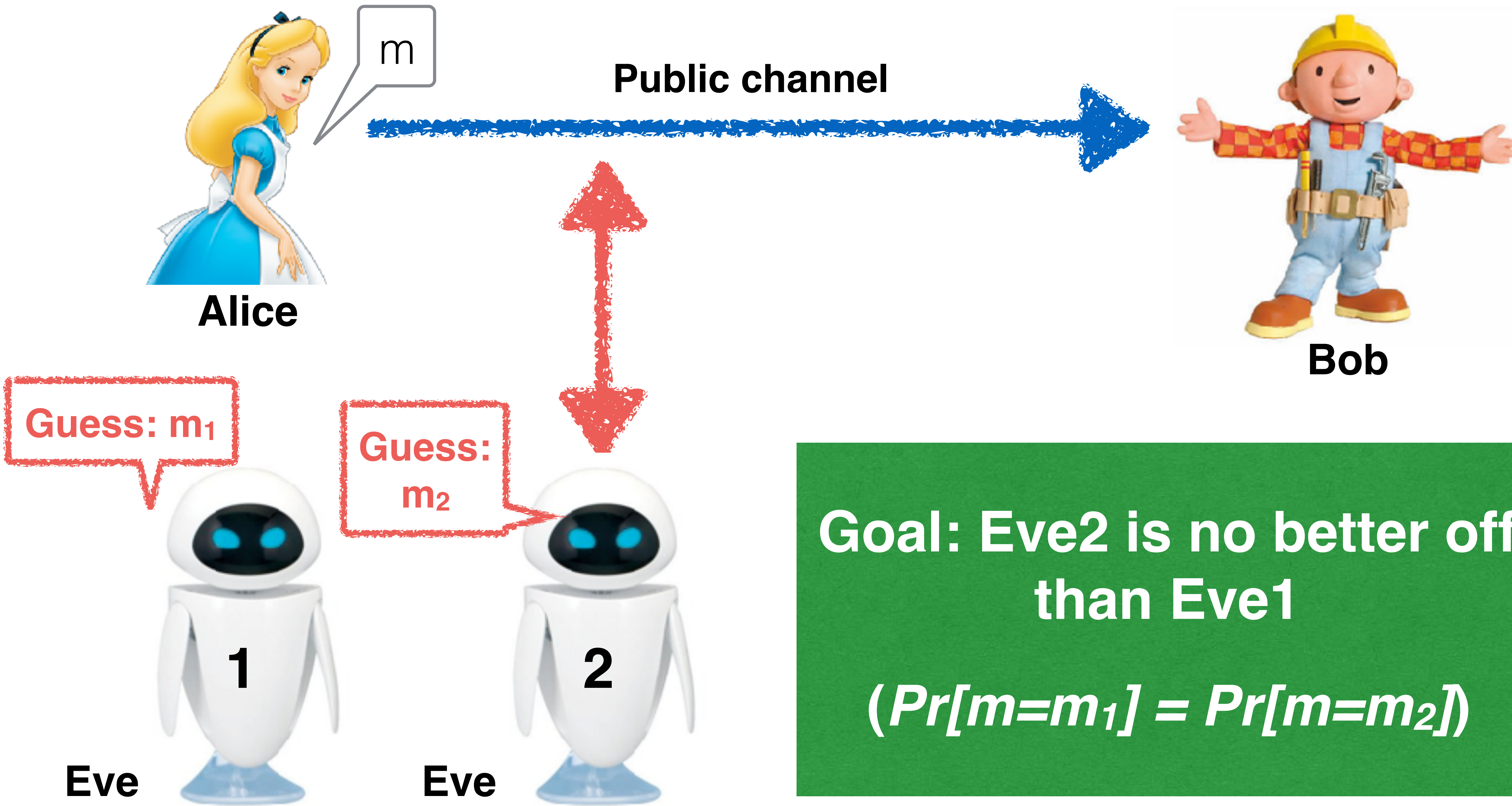
$$\begin{aligned} D(k, E(k, m)) &= D(k, k \oplus m) \\ &= k \oplus (k \oplus m) \\ &= 0 \oplus m \\ &= m \end{aligned}$$

Recall def'n:

- **Efficient** ✓
- **Correct** ✓

Perfect secrecy (Shannon)

a.k.a. Information Theoretic Secrecy



Goal: Eve2 is no better off than Eve1

(*$Pr[m=m_1] = Pr[m=m_2]$*)

Perfect secrecy (Shannon)

aka Information Theoretic Secrecy

- Formal definition:

$$\forall m_0, m_1 \in M. \text{ where } |m_0| = |m_1|$$

$$\forall c \in C.$$

$$\Pr\left[E(k, m_0) = c\right] = \Pr\left[E(k, m_1) = c\right]$$

Good news: OTP has perfect secrecy

- Goal: Show $\Pr\left[E(k, m_0) = c\right] = \Pr\left[E(k, m_1) = c\right]$
- Proof:

$$\Pr[E(k, m_0) = c] = \Pr[k \oplus m_0 = c] \tag{1}$$

$$= \frac{|k \in \{0,1\}^m : k \oplus m_0 = c|}{\{0,1\}^m} \tag{2}$$

$$= \frac{1}{2^m} \tag{3}$$

$$\Pr[E(k, m_1) = c] = \Pr[k \oplus m_1 = c] \tag{4}$$

$$= \frac{|k \in \{0,1\}^m : k \oplus m_1 = c|}{\{0,1\}^m} \tag{5}$$

$$= \frac{1}{2^m} \tag{6}$$

Which attacks does the one-time pad resist?

- Known ciphertext **Yes**
- Known plaintext **No?**
- Chosen plaintext
- Chosen ciphertext

key reuse issue!

Bad news #1: Two-time pad is insecure

- $c_1 = m_1 \oplus k$, $c_2 = m_2 \oplus k$
- No secrecy against known plaintext
- Worse: $c_1 \oplus c_2 = m_1 \oplus m_2$
  - Enough redundancy in ASCII (and English) to reveal m_1 and m_2 with high probability

Bad? News #2: All keys must be equally likely

- Let M = {000, 001}

2 possible messages

- Let K = {000, 001, 010, 011, 100, 101, 110, 111}

8 possible keys

- If k = 000 (random selection), then M = C
  - **OK** if all plaintext is equally likely:
    - 001 + 000 -> 001; 000 + 001 -> 001
  - **Bad** if plaintext is recognizable: “This is a secret”
  - **Necessary** so ciphertexts are equally likely

Bad? News #3: Brute force

- **C = MAEIXRBMYCIYKYYDQYDZVPD**
- Key1 = ABCDEFGHIJKLMNOPQRSTUVWXYZABC
- Key2 = QWERTYUIOPASDFGHJKLZXCVBNMQWE
- Key 3 = QAZWSXEDCRFVTGBYHNUJMIKOLPQAZ
- Can you find my secret message?
  - http://www.braingle.com/brainteasers/codes/onetimepad.php

Again, not a problem if all messages equally likely

More bad news

- Theorem: Perfect secrecy requires |K| >= |M|
  - Why is this bad news?
  - *If you could send K securely, you could send M securely and you don't need the encryption!*
- In practice, fall back to ***computational security***
  - Can't be solved by an attacker faster than X
    - In practice, faster than some assumed hard math problem, like integer factorization
  - Assumes limited computational resources

Recall security goals

- Privacy, integrity, authenticity
- Which apply to the one-time pad? Why?
  - *Only privacy!*

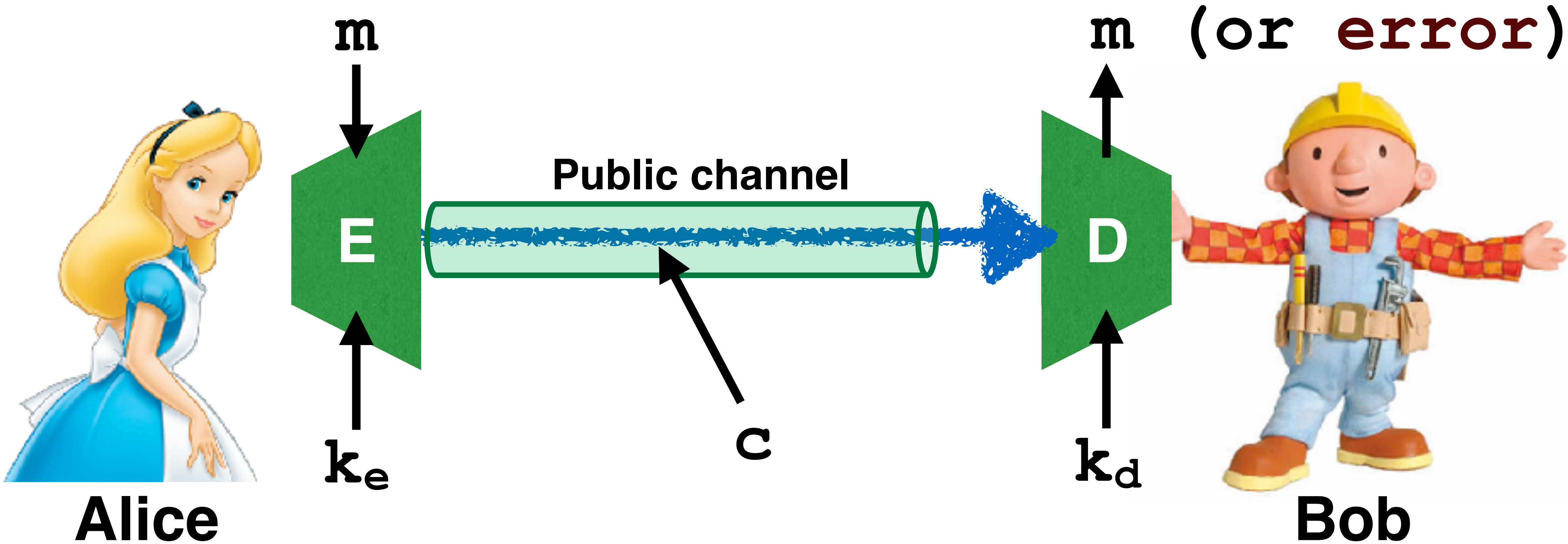
Preview of crypto unit

Covered in this class

	Symmetric trust model	Asymmetric trust model
Privacy	Private-key encryption • Stream ciphers • Block ciphers	Public-key encryption
Authenticity, Integrity	Hashes, MACs, authenticated encryption	Signatures, PKI, certificates, SSL/TLS, user authentication
	Everyone shares the same secret *k*	**Every party has her own secret**

Assumptions: (1) All algorithms **public**, (2) security based **only** on key size

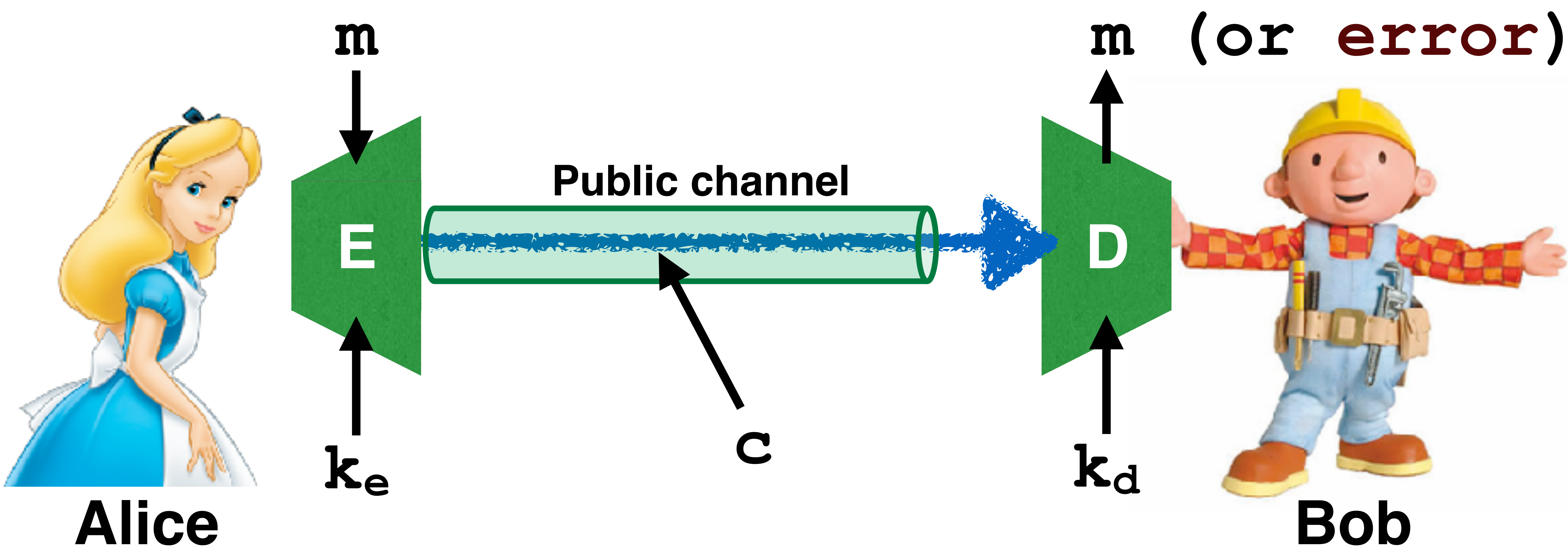
Symmetric crypto



- $k = k_e = k_d$
- Everyone who knows ***k*** knows the whole secret

- How did Alice and Bob both get the secret key?
  - That is a different problem
  - Not solved by symmetric crypto. ***Assumed.***

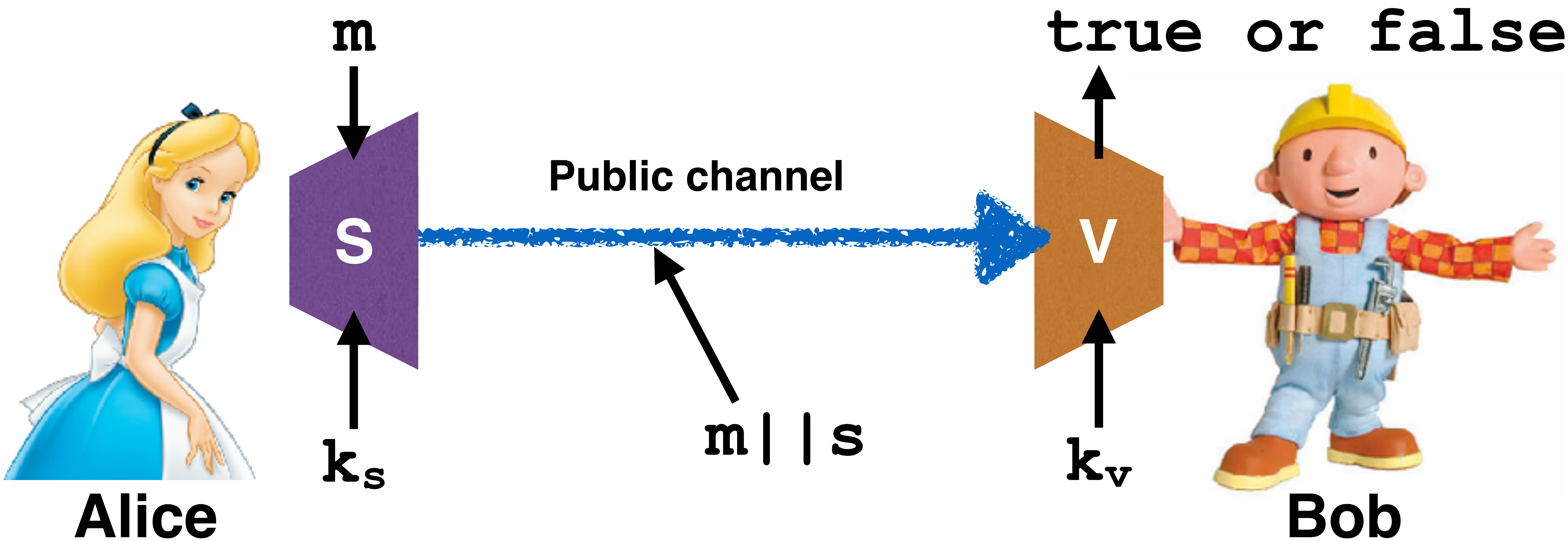
Asymmetric crypto



- k_e != k_d
- k_d = **private** key, k_e = **public** key
  - Bob computes both, gives public key to Alice
- Alice sends a message to Bob: $c = E(m, k_e)$
- Bob can decrypt it: $m = D(m, k_d)$
- Anyone can send, **only Bob can read!**

- How did Alice get Bob’s public key?
  - That’s easy, he sent it in plain / publicly
  - BUT, how does she know it came from Bob?
    - And not from Eve?
  - Again, this is a separate problem. ***Assumed.***

Message authentication



s = **S**ign(m, k_s)

Verify(m,s,k_v) ?= true

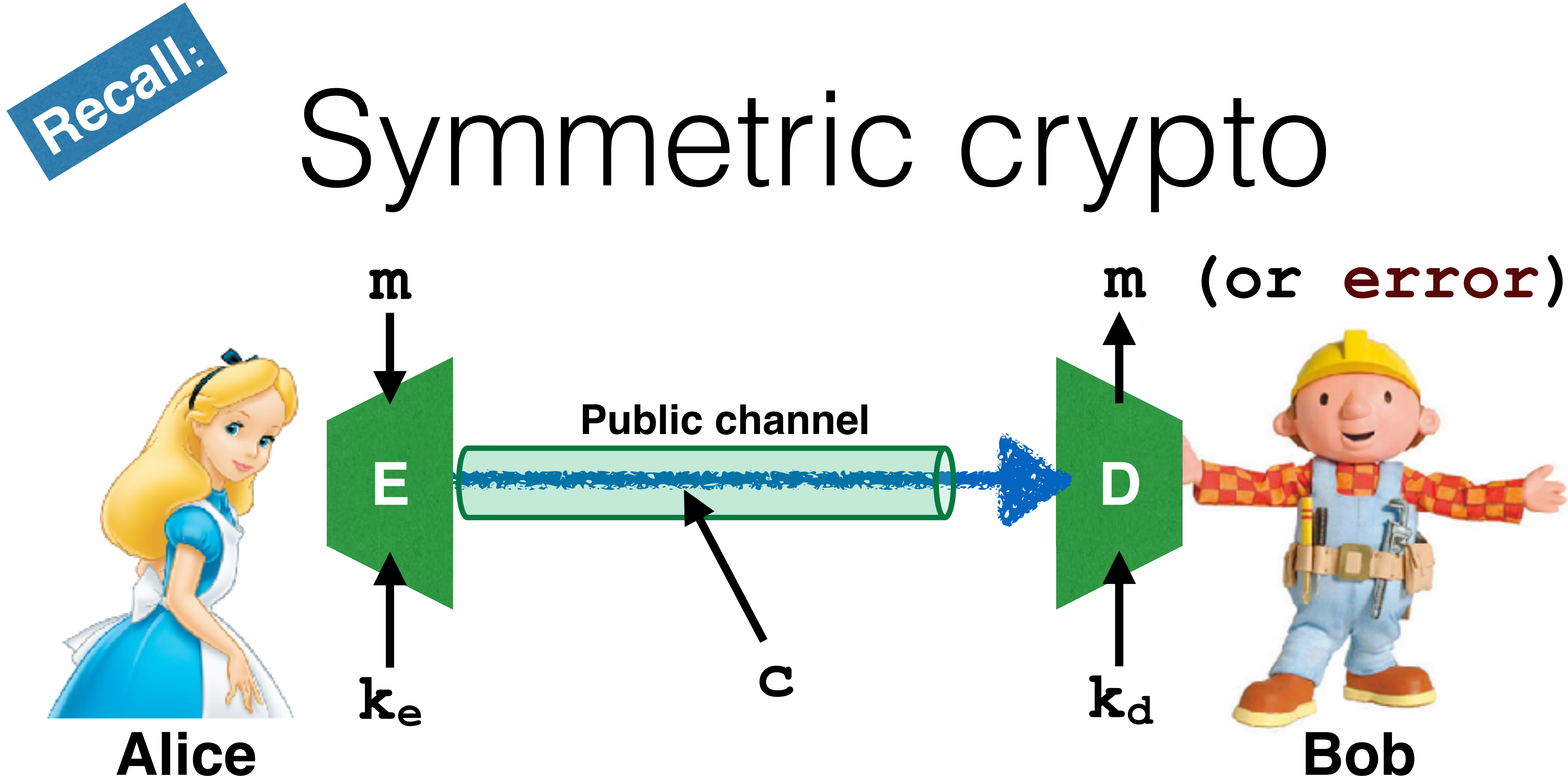
Only someone who knows k_s could have sent the message!

Session keys

- Generally bad idea to use your long-lived keys a lot
  - Increase opportunities for KPA
- Instead, generate ***session key***
  - Using existing keys, generate *fresh* session key
  - Be sure session key is authentic
  - Use session key for this session only
- Also faster (asymmetric crypto is slow)

Next… Block Ciphers

Symmetric crypto

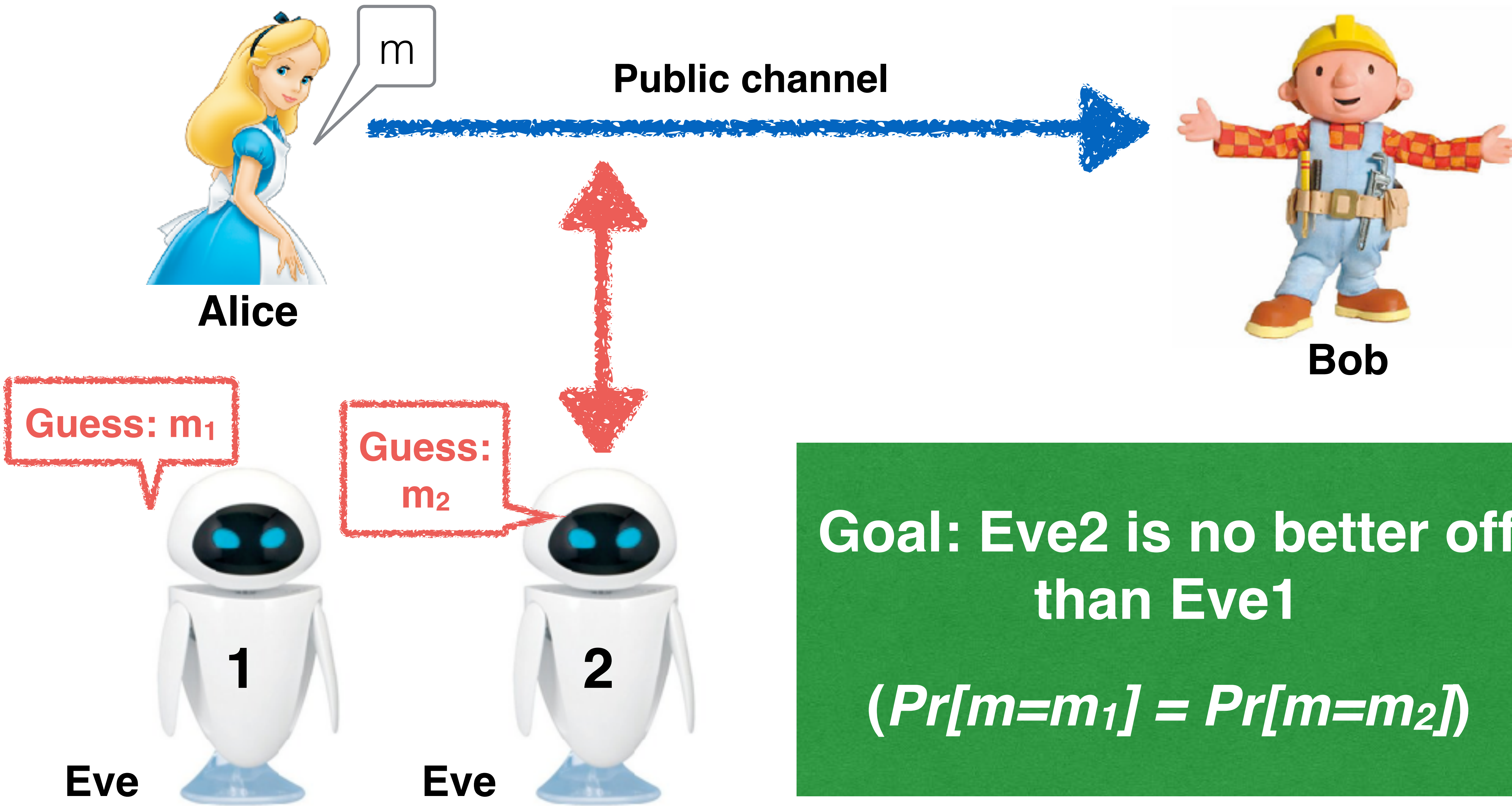


- $k = k_e = k_d$
- Everyone who knows ***k*** knows the whole secret

Recall:

Perfect secrecy (Shannon)

a.k.a. Information Theoretic Secrecy



Goal: Eve2 is no better off than Eve1

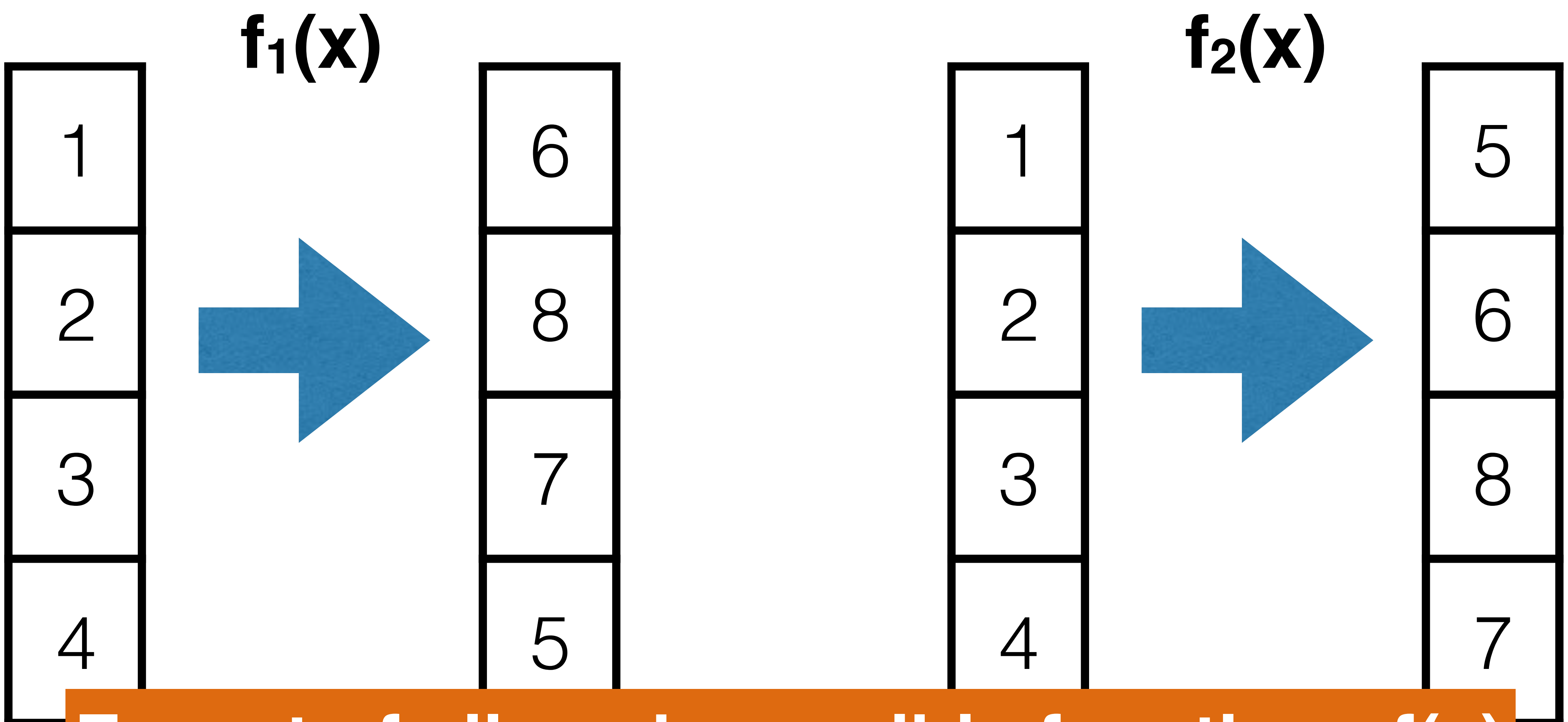
($Pr[m=m_1] = Pr[m=m_2]$)

Random functions

- Terminology note:
  - Capital letters (X,Y,F) = sets of things
  - Lowercase letters (x,y,f) = individual things in set

Concept model: f(x)

- f(x) maps inputs X to outputs Y (X = Y or X != Y)
- For reasonably-sized X and Y, LOTS of possibilities!



F = set of all such possible functions f(x)

Draw f from F at random

- Pr[f(x) = y) == 1/|Y|
- If f(3) = 8, then what is f(4)?
  - We don't know! And there is no way to predict (unless you know f). ***True for all values x.***
- Given that f(x) = 7, what is x?
  - Can't find out without brute-forcing.
    - How long will this take?
  - This is called a ***one-way function***

Why do we care?

Because one-way functions provide confidentiality!

- If Alice writes f(x) instead of x to a file, no one can recover the plaintext without brute-forcing.
  - Including Alice! (This is a problem.)

Alice

(Efficiently) Recovering x

- If everyone can invert f, no confidentiality
- Instead, we want a ***one-way trapdoor function***:
  - F(k,x) = y
  - If you know y and k, you can recover x easily
  - If you don't know k, you must brute-force

This is starting to resemble our cryptosystem model!

This is all imaginary

- Storing all the possible fs in F would be hard
- No true one-way trapdoor has been found
  - Unclear whether it’s possible
- Instead: **Approximate** this with ***pseudo-random functions*** (PRF)
  - These are really hard to create correctly!

Pseudo-Random Functions

Pseudo-random function family (PRF)

- F: ***family*** of functions f(x)
  - All have the same domain and range X, Y
- **Randomly** choose one function $f_k(x)$
  - Recall, k is our trapdoor
  - *k is which function in the family we chose!*
- Cannot distinguish between a true random function, and a randomly chosen function in F
  - Family is public!

PRF security

World 0

Setup: tbl[*] = random

— — —

Return: y = tbl[x]

x

y

Eve

(polynomial time)

World 1

Setup: k = rand()

— — —

Return: y = $f_k(x)$

x

$g_k(x)$

Eve's job: Provide x. Figure out which world we are in. With very high probability, she can't do better than random guessing.

- Note — if attacker is wrong most of the time (rather than half the time), that indicates *insecurity*.
    - She should switch guesses

Block ciphers

Block cipher basics

- Start with a PRF
  - That operates on fixed-length ***blocks**:* input size == output size
  - Each function is a ***permutation*** (it's invertible)
  - Each function, inverse is efficiently computable
- Key length: related to how many functions there are
- Block length: size of input/output block

Security goal

- Every f_k in F has the same range
    - Every output (ciphertext) belongs to some input
    - But the permutation is different for each k

- Goal: If you don't know k, you cannot distinguish!

- ***k is the only secret!***

Example block cipher

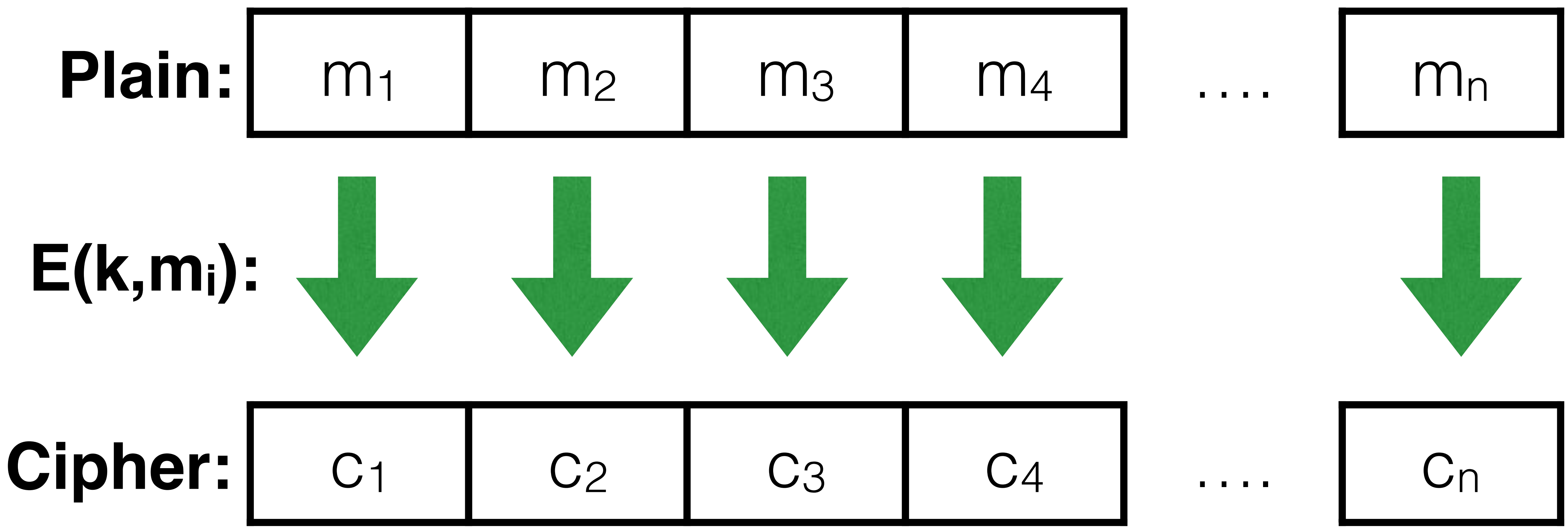
- Let block length == key length
- $E_k(m) = k \oplus x = c$; $Dk(c) = k \oplus c = x$

Beyond the block

- Block ciphers operate on a (small) fixed size block
  - AES = 128 bits
  - This is not enough for real applications
- Instead: Break input up into blocks
  - Strategy for doing this = ***encryption mode***
  - Different modes = different security, performance

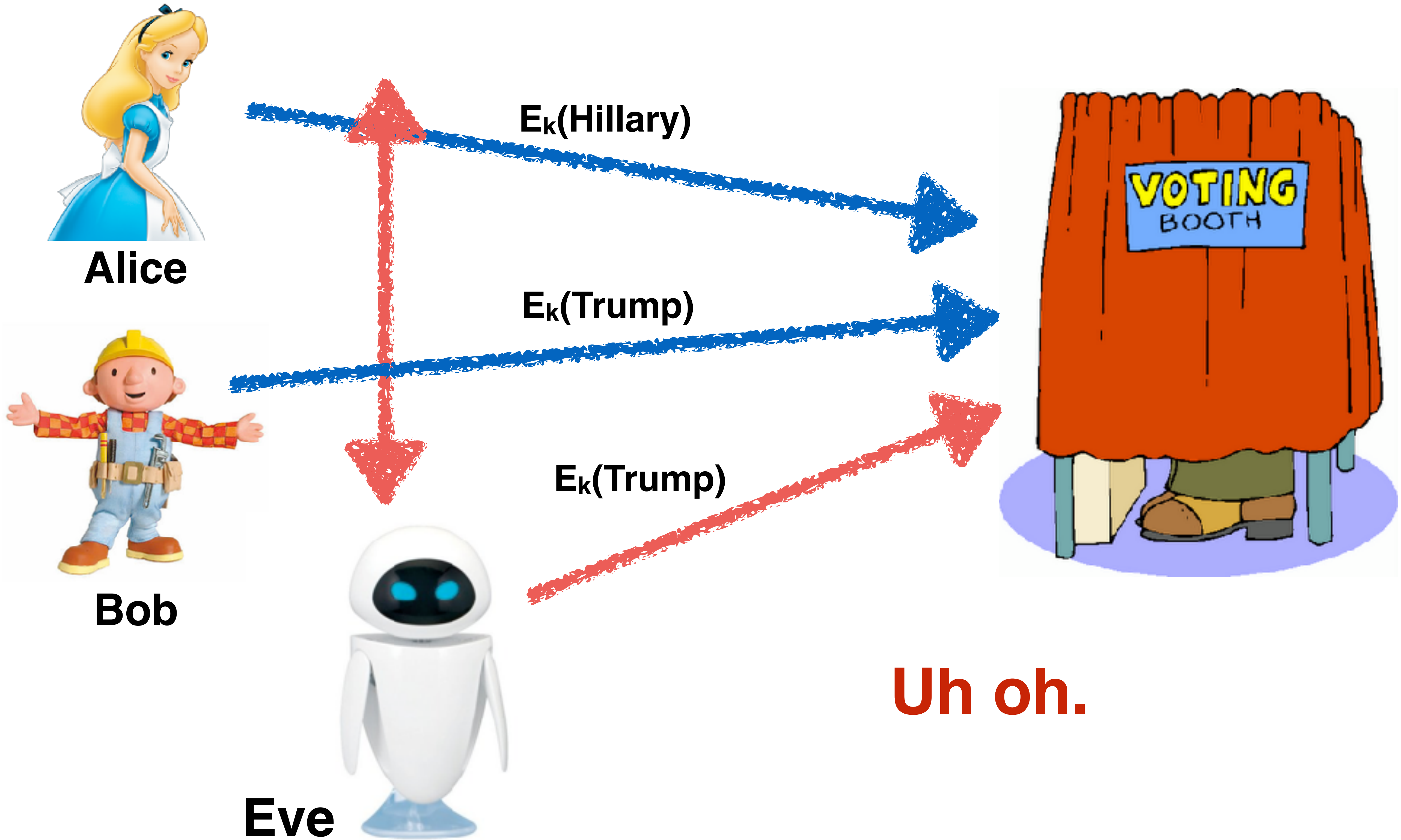
Block cipher modes

Electronic code book (ECB)



- What is the problem here?

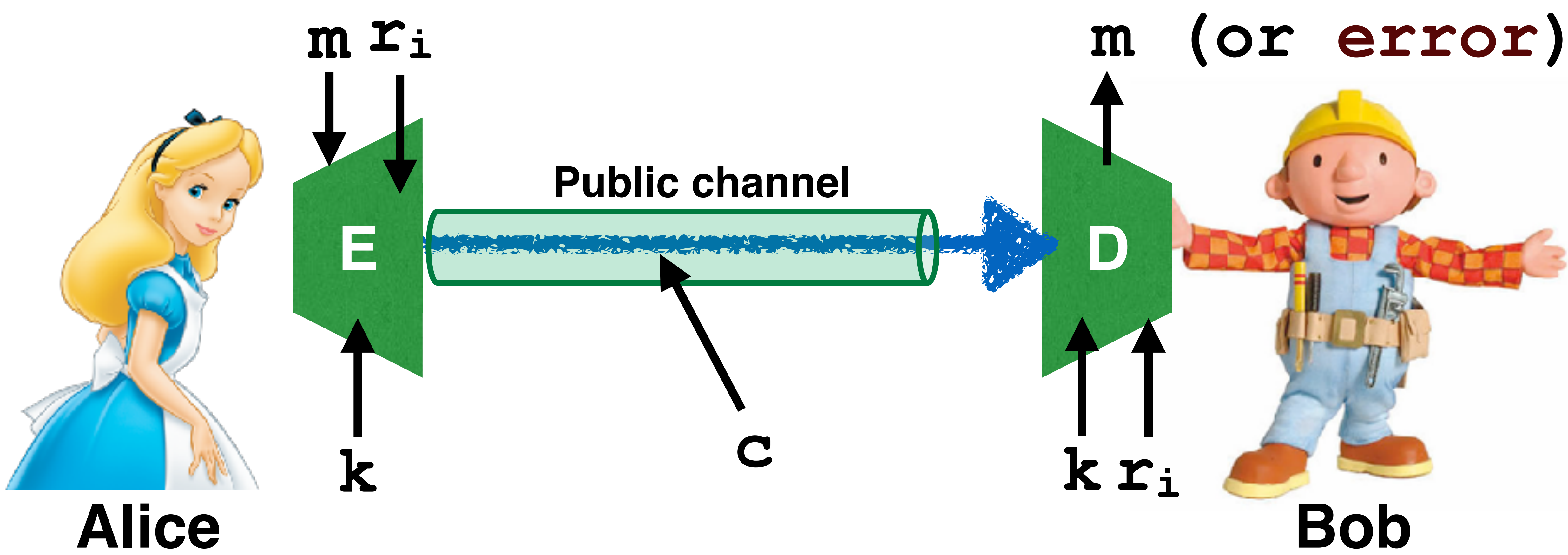
CPA revisited



Uh oh.

- CPA-resistance is ***mandatory***
- Deterministic schemes ***cannot be CPA-secure***
    - Nor are they secure to send multiple messages
- Moral: ***Always use randomized encryption!***
    - Which builds in a varying value per message
    - Never use ECB mode!

Randomizing block ciphers



- r is an ***Initialization Vector (IV)***
- c_i = E(k, m XOR r_i); m = r_i XOR D(k,c_i)
- r can be *random* or *unique* (see next slides)

Random IV

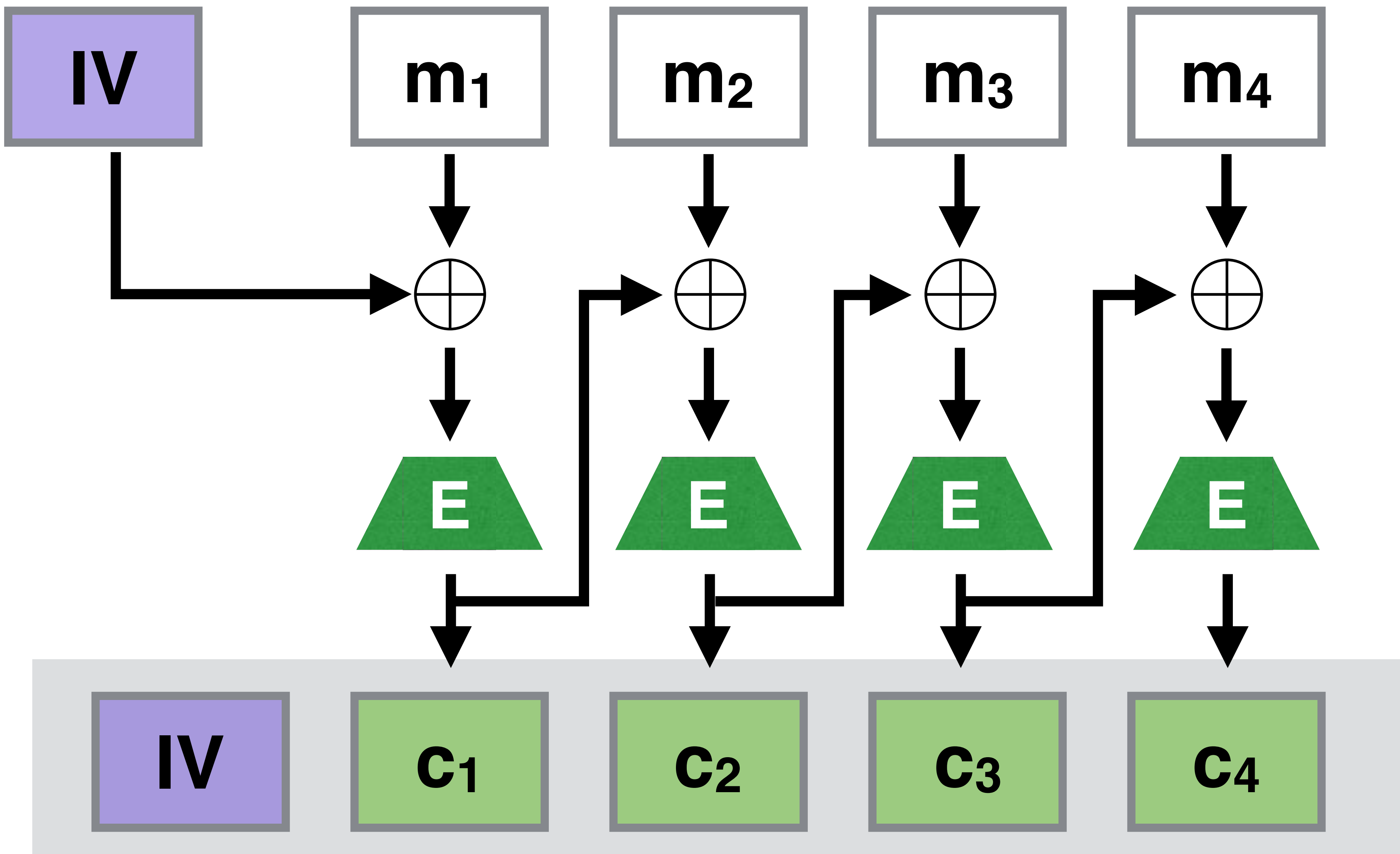
- A new string chosen at random per message
- Must send r along with c; use more bandwidth

Unique IV

- Should not repeat
- New IV for *every* ***message***
- Both sender and receiver can predict which IV is used for the next message; must remain synch'd.
    - Naive counter (not very secure, more later)
        - $IV_{i+1} = IV_i + 1$
    - Better: Calculate from previous: $IV_{i+1} = E(k,IV_i)$
- Less bandwidth but requires ***in-order*** delivery

- How to generate randomness at each block?
  - Lucky for us, E provides built-in randomness

Cipher-block chaining (CBC)



etc.

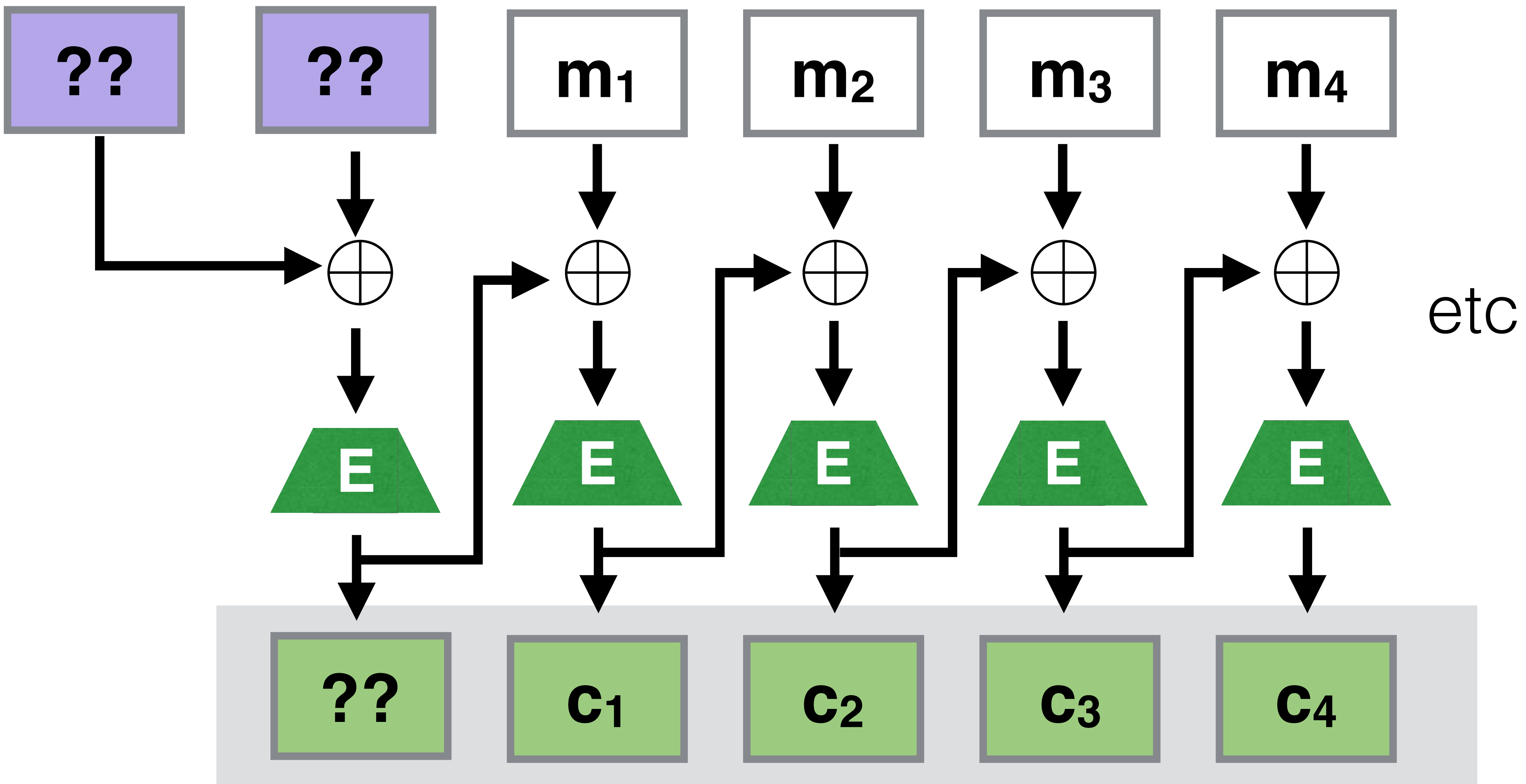
Ciphertext

Decrypt? $m_i = D(k, c_i)$ XOR c_{i-1}

CBC continued

- Solves the randomization/CPA problem
- Encryption is not parallelizable anymore
- Other similar approaches: PCBC, OFB, CFB
  - XOR in more/different stuff
  - Different performance characteristics

Avoid synching IV



First block of output is gibberish, but we don't care!

Attacking predictable IV

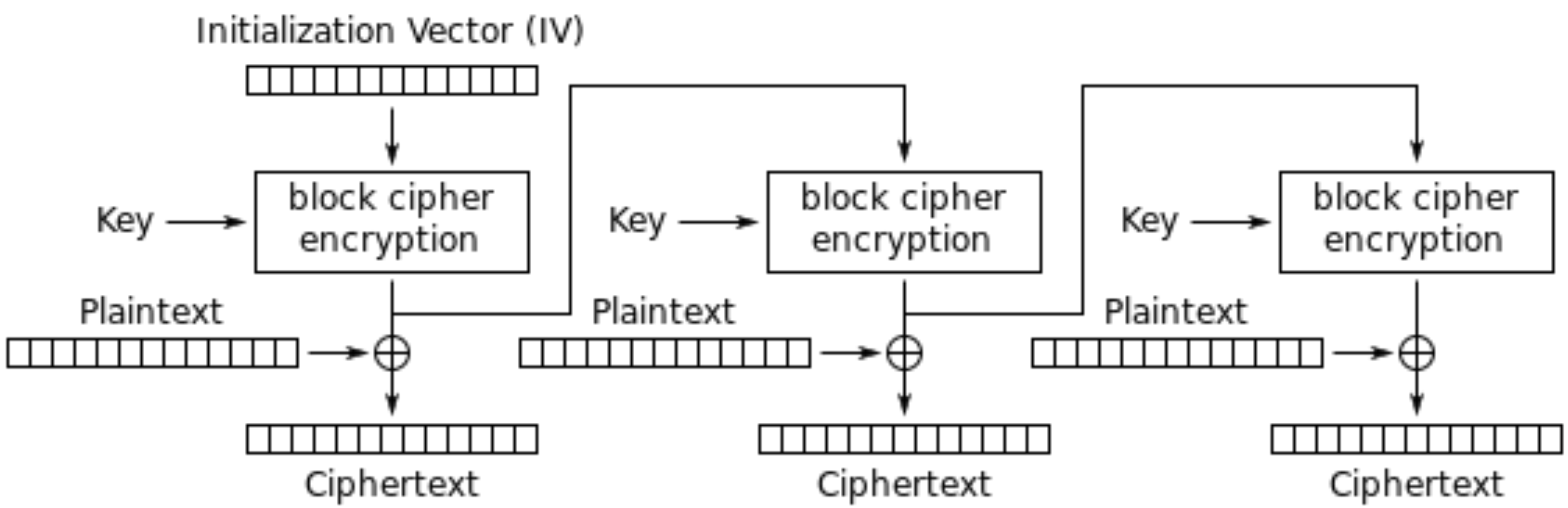
- Assume:
  - Options for m are known (e.g., {yes, no})
  - Last IV (for m_n)is known
  - Next IV is predictable
  - CPA attack
- Choose m_{n+1} = "yes" XOR IV_n XOR IV_{n+1}
  - If c_{n+1} == c_n then m_n was "yes" … Why?

Stream cipher

- Inspired by the one time pad
- Generate ongoing series of bits
  - XOR each bit with next bit of *m*
- Synchronous: Sender, receiver use same bitstream
  - Requires synchronizing where to start, no drops
- Musts: Good randomness, long period
- Several popular ones but partially busted
  - Especially used for GSM

Output feedback (OFB)

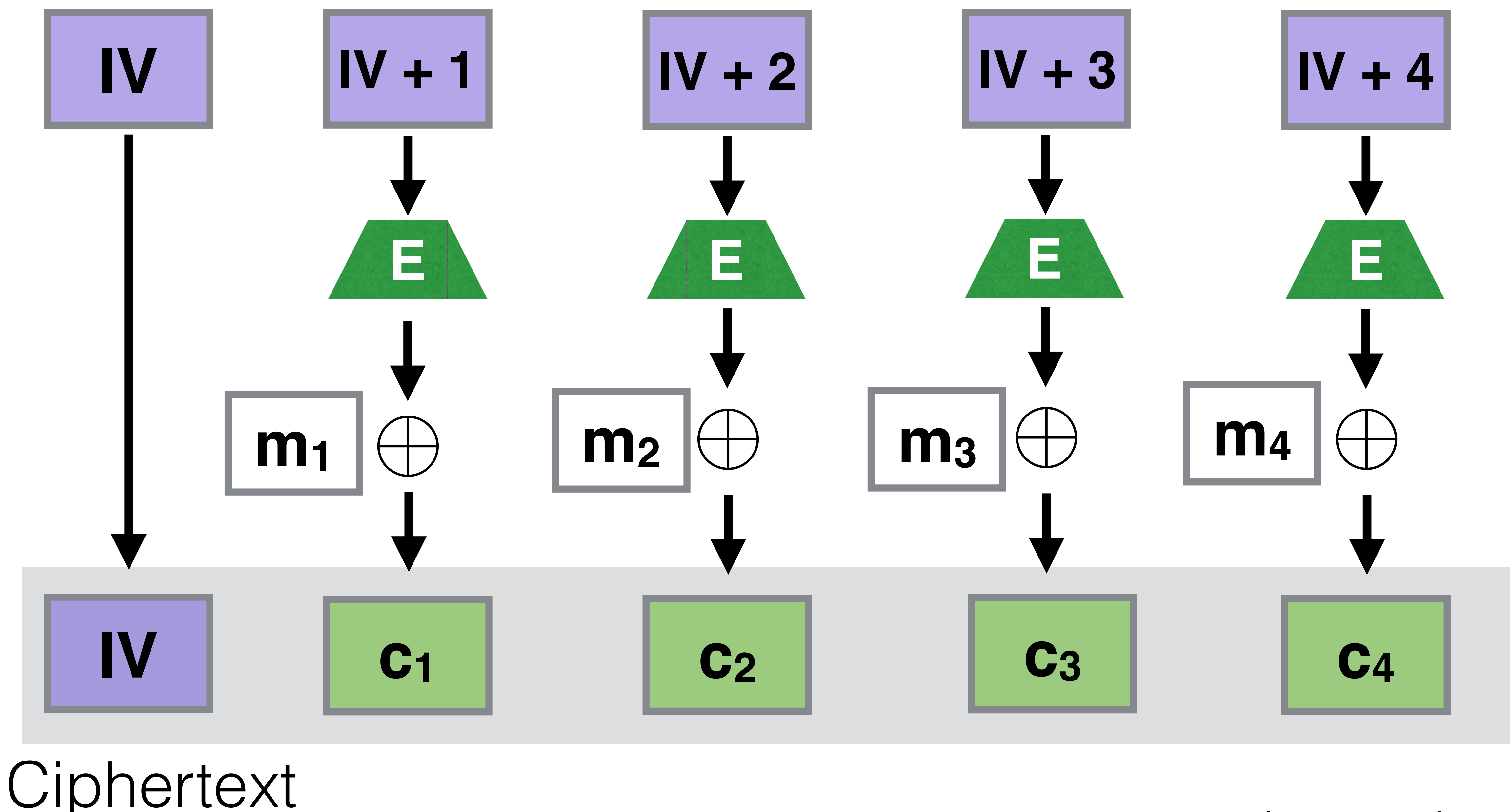
- Turns a block cipher into a stream cipher
  - Generate ***key stream*** which is XORed with data
- Can generate all keys ahead, encrypt in parallel



Used improperly, can cycle too fast!

https://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

Counter mode (CTR)



Decrypt? m_i = D(k,IV+i) XOR c_i

More on counter mode

- Also converts block to stream cipher
  - Like OFB, encrypt in parallel
- Generate keystream as a sequence
  - Sequence guaranteed not to repeat for a while
- Possible attacks:
  - If initial IV is not random ***and*** not transformed or concatenated
  - When used properly, very secure

- This can cause trouble if you naively increment the IV every time between messages.
- For message A, the input to the block cipher would be IV_A, $IV_A + 1$, $IV_A + 2$, etc.
- Then if for message B, you use $IV_B = IV_A + 1$
    - Input to blocks will be equivalent to $IV_A + 1$, $IV_A + 2$, etc.
- In effect you are reusing inputs to the block cipher in a predictable way, which is bad.

Block cipher padding

- If your message doesn't divide evenly into blocks, then what?
  - Doesn't apply in streaming mode (e.g. CTR)
  - ***Pad*** with extra bytes in some known pattern
- Susceptible to ***padding oracle*** attacks
  - Brute-force one byte at a time (from end) in m_{n-1}
  - Because of XOR, affects padding at the end of m_n
  - If you guessed right, padding will be valid

Lesson: Don't return informative errors

- Cool walkthrough of padding oracle attack:
    - http://robertheaton.com/2013/07/29/padding-oracle-attack/

Common block ciphers

Data Encryption Standard (DES)

- Developed at IBM/NSA in 1970s (non-public process)
- 56-bit key, 64-bit block length
- Concerns:
  - Short key length — can be brute-forced in days
  - Short block length — repeat blocks too often

Triple DES

- Triple the key length: $k = (k_1, k_2, k_3)$
  - Still short block length
  - How much does brute-force increase by?

- New block cipher:
  - $E3(k,m) = E(k_1,D(k_2,E(k_3,m)))$
  - One version: $k_1 = k_3$
    - Effective key length = 112

- Fairly slow, but used in practice (back compatible)

Advanced Encryption Standard (AES)

- Public contest at NIST, 1997
  - 15 candidates, winner selected in 2000
  - Lots of analysis of each candidate
- Efficiency + security considered
  - "Most secure" didn't win!
- Supports keys: 128/192/256 bits
  - Nanoseconds since big bang: ~2^{90}

Summing up (so far)

- Symmetric crypto is very fast in practice
  - Especially stream ciphers
- If ***used properly*** it can be very secure
- Next time:
  - Message authentication
  - Key exchange