15-853:Algorithms in the Real World

Announcement (reminder):

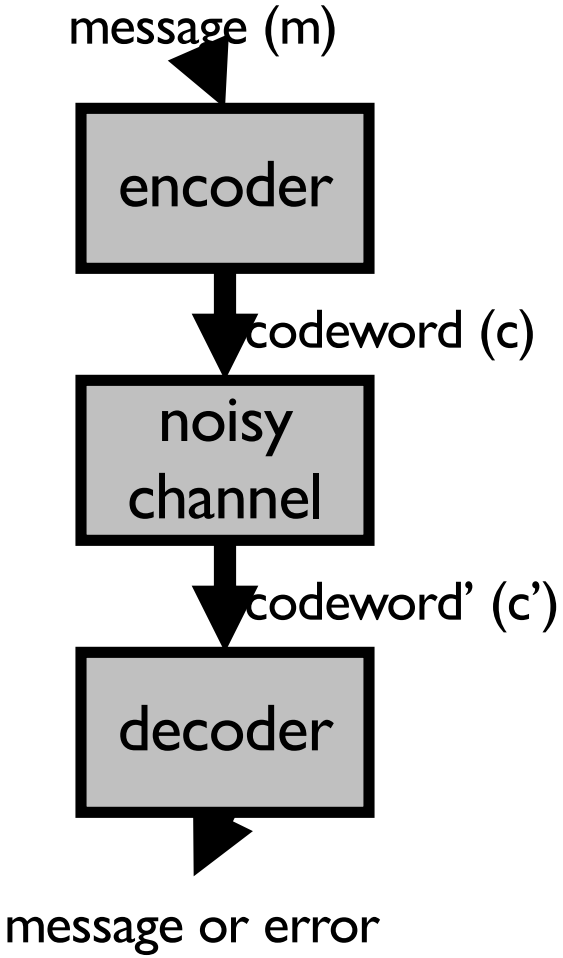
- There is recitation this week:
  - HW3 solution discussion and a few problems
- Exam: Nov. 26
  - 5-pages of cheat sheet allowed
  - Need not use all 5 pages of course!
  - At least one question from each of the 5 modules (Will test high level concepts learned)
- Clarification on Eigen values of the empirical covariance matrix

Today: A high level summary of the course

- With more emphasis on stuff we covered earlier
- Can’t cover everything

Error Correcting Codes

General Model



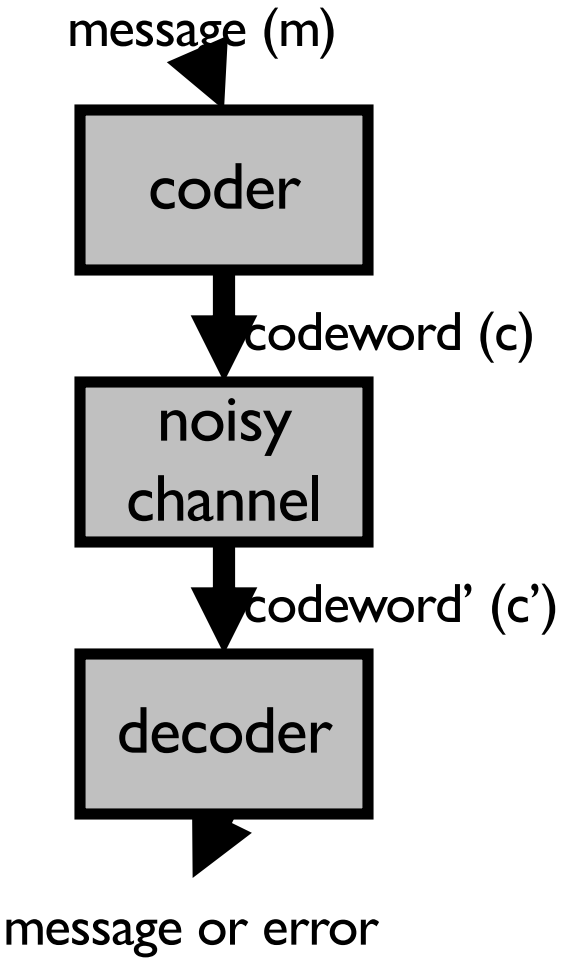
"Noise" introduced by the channel:

- changed fields in the codeword vector (e.g. a flipped bit).
  - Called **errors**
- missing fields in the codeword vector (e.g. a lost byte).
  - Called **erasures**

How the decoder deals with errors and/or erasures?

- **detection** (only needed for errors)
- **correction**

Block Codes



Each message and codeword is of fixed size

Σ = codeword alphabet

k =|m| **n** = |c| **q** = $|\Sigma|$

C = "code" = set of codewords

C $\subseteq \Sigma^n$ (codewords)

Δ(x,y) = number of positions s.t. $x_i \neq y_i$

d = $\min\{\Delta(x,y) : x,y \in C, x \neq y\}$

Code described as: $\mathbf{(n,k,d)_q}$

Role of Minimum Distance

Theorem:

A code C with minimum distance "d" can:

1. detect any (d-1) errors
2. recover any (d-1) erasures
3. correct any <write> errors

Stated another way:

For s-bit error detection $d \geq s + 1$

For s-bit error correction $d \geq 2s + 1$

To correct a erasures and b errors if

$$d \geq a + 2b + 1$$

Recap: Linear Codes

If $\sum$ is a field, then $\sum^n$ is a vector space

Definition: C is a linear code if it is a linear subspace of $\sum^n$ of dimension k.

This means that there is a set of k independent vectors $v_i \in \sum^n$ $(1 \leq i \leq k)$ that span the subspace.

i.e. every codeword can be written as:

$$c = a_1 v_1 + a_2 v_2 + \ldots + a_k v_k \quad \text{where } a_i \in \sum$$

"Linear": linear combination of two codewords is a codeword.

Minimum distance = weight of least-weight codeword

Recap: Generator and Parity Check Matrices

Generator Matrix:

A k x n matrix **G** such that: $C = \{ xG \mid x \in \sum^k \}$

Made from stacking the spanning vectors

Parity Check Matrix:

An (n – k) x n matrix **H** such that: $C = \{y \in \sum^n \mid Hy^T = 0\}$

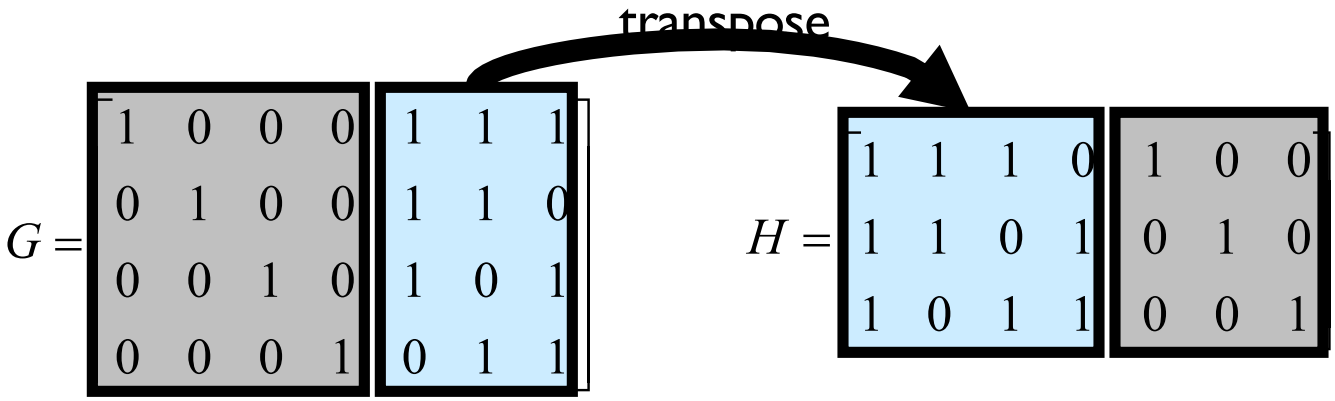
(Codewords are the null space of H.)

These **always exist for linear codes**

Recap: Relationship of G and H

Theorem: For linear codes, if G is in standard form $[I_k \; A]$ then $H = [-A^T \; I_{n-k}]$

Example of (7,4,3) Hamming code:



Recap: Dual Codes

For every code with

$G = [I_k \; A]$ and $H = [A^T \; I_{n-k}]$

we have a **dual code** with

$G = [I_{n-k} \; A^T]$ and $H = [A \; I_k]$

Another way to define dual codes:

- Dual code of a linear code C is the null space of the code

That is, subspace which is orthogonal to every vector in the subspace defined by the code.

The generator matrix of the dual code in this strict sense is the parity check matrix H (of code C)

Recap: Properties of Syndrome and connection to error locations

Hy^T is called the **syndrome** (0 if a valid codeword).

In **general** we can find the error location by creating a table that maps each syndrome to a set of error locations.

Theorem: assuming $s \leq (d-1)/2$ errors, every syndrome value corresponds to a unique set of error locations.

Recap: Singleton bound and MDS codes

Theorem: For every $(n, k, d)_q$ code, $n \geq k + d - 1$

Codes that meet Singleton bound with equality are called
Maximum Distance Separable (MDS)

Only two binary MDS codes!

1. Repetition codes
2. Single-parity check codes

Need to go beyond the binary alphabet!
(We need some number theory for this)

Recap: Groups

A **Group** (G,*,I) is a set *G* with operator * such that:

1. **Closure**. For all $a,b \in G$, $a * b \in G$
2. **Associativity.** For all $a,b,c \in G$, $a*(b*c) = (a*b)*c$
3. **Identity.** There exists $I \in G$, such that for all $a \in G$, $a*I=I*a=a$
4. **Inverse.** For every $a \in G$, there exist a unique element $b \in G$, such that $a*b=b*a=I$

An **Abelian or Commutative Group** is a Group with the additional condition

5. **Commutativity.** For all $a,b \in G$, $a*b=b*a$

Fields

A **Field** is a set of elements F with binary operators * and + such that

1. (F, +) is an **abelian group**
2. (F \ I_+, *) is an **abelian group** the "multiplicative group"
3. **Distribution**: a*(b+c) = a*b + a*c
4. **Cancellation**: a*I_+ = I_+

Example: The reals and rationals with + and * are fields.

The **order (or size)** of a field is the number of elements.

A field of finite order is a **finite field**.

Recap: Finite fields

- Size (or order): Prime or power of prime
- Size = prime:
  - Z_p and modulo p arithmetic suffices
- Power-of-prime finite fields:
  - Constructed using polynomials
  - Mod by irreducible polynomial
- Correspondence between polynomials and vector representation

Recap: GF(2^n)

$\mathbb{F}_{2^n}$ = set of polynomials in $\mathbb{F}_2[x]$ modulo irreducible polynomial $p(x) \in \mathbb{F}_2[x]$ of degree n.

Elements are all polynomials in $\mathbb{F}_2[x]$ of degree $\leq n-1$.
Has 2^n elements.
Natural correspondence with bits in $\{0,1\}^n$.

Elements of $\mathbb{F}_{2^8}$ can be represented as a byte, one bit for each term.

E.g., $x^6 + x^4 + x + 1 = 01010011$

RS code: Polynomials viewpoint

Message: $[a_{k-1}, \ldots, a_1, a_0]$ where $a_i \in GF(q^r)$

Consider the polynomial of degree k-1

$$P(x) = a_{k-1} x^{k-1} + \cdots + a_1 x + a_0$$

RS code:

Codeword: [P(1), P(2), …, P(n)]

To make the i in p(i) distinct, need field size $q^r \geq n$

That is, need sufficiently large field size for desired codeword length.

Recap: Minimum distance of RS code

Theorem: RS codes have minimum distance d = n-k+1

Proof:

1. *RS is a linear code:* if we add two codewords corresponding to P(x) and Q(x), we get a codeword corresponding to the polynomial P(x) + Q(x). Similarly any linear combination..
2. *So look at the least weight codeword*. It is the evaluation of a polynomial of degree k-1 at some n points. So it can be zero on only k-1 points. Hence non-zero on at most (n-(k-1)) points. This means distance at least n-k+1
3. Apply Singleton bound

Meets Singleton bound: RS codes are MDS

Recap: Generator matrix of RS code

Q: What is the generator matrix?

<board>

"Vandermonde matrix"

Special property of Vandermonde matrices:
Full rank (columns linearly independent)

Vandermonde matrix: Very useful in constructing codes.

Concatenation of Codes

Take any $(N, K, D)_{q^k}$ code.

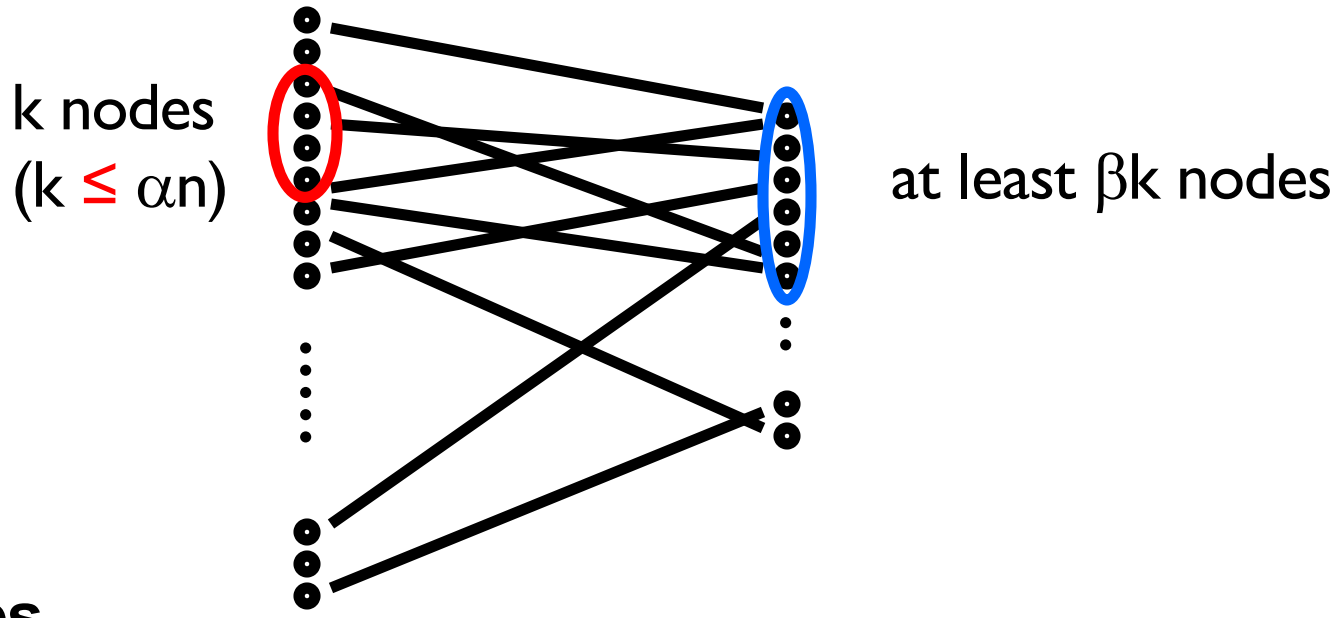
Can encode each alphabet symbol of k bits using another $(n, k, d)_q$ code.

Theorem:

The concatenated code is a $(Nn, Kk, Dd)_q$ code

LDPC codes

(α, β) Expander Graphs (bipartite)



Properties

- **Expansion:** every small subset ($k \leq \alpha n$) on left has many ($\geq \beta k$) neighbors on right
- **Low degree –** not technically part of the definition, but typically assumed

Recap: Expander Graphs: Constructions

Theorem: For every constant 0 < c < 1, can construct bipartite graphs with

n nodes on left,

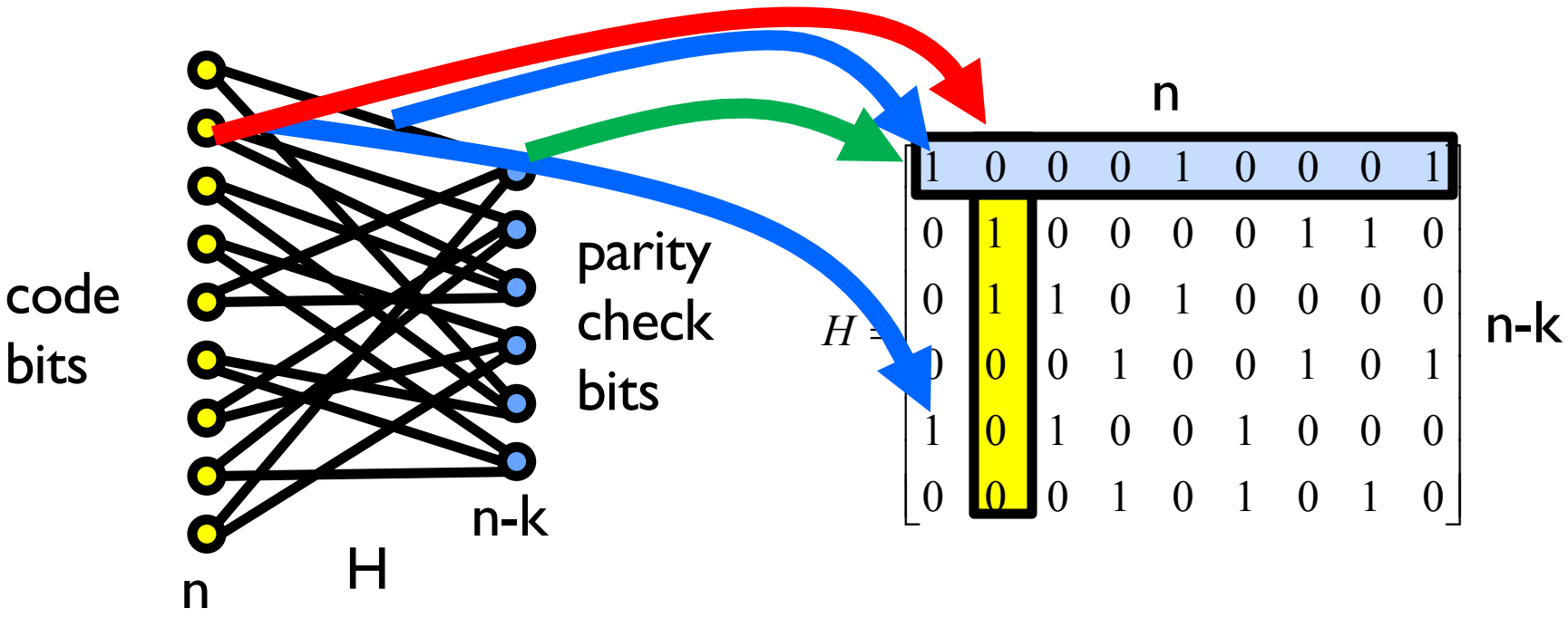
cn on right,

d-regular (left),

that are $(\alpha, 3d/4)$ expanders, for constants α and d that are functions of c alone.

"Any set containing at most alpha fraction of the left has (3d/4) times as many neighbors on the right"

Recap: Low Density Parity Check (LDPC) Codes



Each row is a vertex on the right and each column is a vertex on the left.

A codeword on the left is valid if each right "parity check" vertex has parity 0.

The graph has O(n) edges (**low density**)

The random erasure model

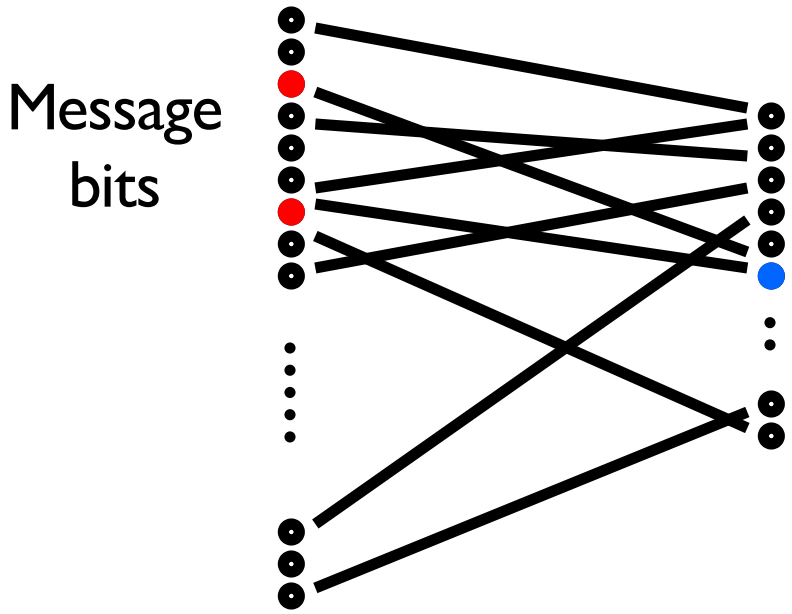
Recovering from erasures

Q: Why erasure recovery is quite useful in real-world applications?

Hint: Internet

Packets over the Internet often gets lost (or delayed) and packets have sequence numbers!

Tornado Codes



$c_6 = m_3 \oplus m_7$

Similar to standard LDPC codes but right side nodes are not required to equal zero.

(i.e., the graph does not represent H anymore).

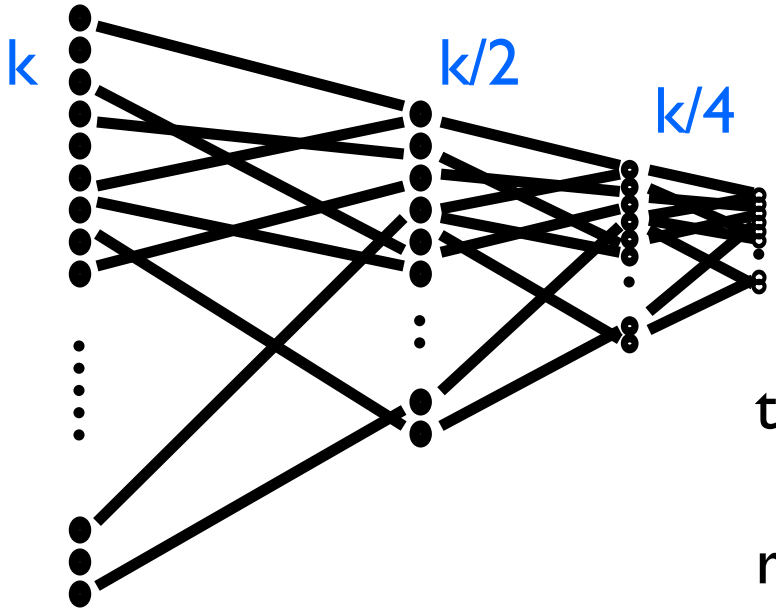
Decoding

If parity bits not lost, then works out.

What if parity bits are lost?

Cascading

- Use another bipartite graph to construct another level of parity bits for the parity bits
- Final level is encoded using RS or some other code



stop when $k/2^t$ "small enough"

total bits $n \leq k(1 + \frac{1}{2} + \frac{1}{4} + \ldots)$

$= 2k$

rate = $k/n = \frac{1}{2}$.

(assuming p =1/2)

Tornado codes enc/dec complexity

Encoding time?

- for the first t stages : |E| = d x |V| = O(k)
- for the last stage: poly(last size) = O(k) by design.

Decoding time?

- start from the last stage and move left
- Last stage is O(k) by design
- Rest proportional to |E| = O(k)

So get very fast (linear-time) coding and decoding.

100s-10,000 times faster than RS

Fountain Codes

Recap: Ideal properties of Fountain Codes

1. Source can generate any number of coded symbols
2. Receiver can decode message symbols from any subset with small reception overhead and with high probability
3. Linear time encoding and decoding complexity

"Digital Fountain"

Recap: LT Codes

- First practical construction for Fountain Codes
- Graphical construction
- **Encoding algorithm**
  - Goal: Generate coded symbols from message symbols
  - Steps:
    - Pick a degree d randomly from a "degree distribution"
    - Pick d distinct message symbols
    - Coded symbols = XOR of these d message symbols

Recap: LT Codes Decoding

Goal: Decode message symbols from the received symbols

Algorithm: Repeat following steps until failure or stop successfully

1. Among received symbols, find a coded symbol of degree 1
2. Decode the corresponding message symbol
3. XOR the decoded message symbol to all other received symbols connected to it
4. Remove the decoded message symbols and all its edges from the graph
5. Repeat if there are unrecovered message symbols

Peek into the analysis

Theorem: Under Robust Soliton degree distribution, the decoder fails to recover all the msg symbols with prob at most δ from any set coded symbols of size:

$$k + \underbrace{O\left(\ln^2(k/\delta)\cdot\sqrt{k}\right)}_{\text{overhead}}$$

And, the number of operations on average used for **encoding** each coded symbol:

$$O\left(\ln(k/\delta)\right)$$

And, the number of operations on average used for **decoding:**

$$O\left(k\ln(k/\delta)\right)$$

Peek into the analysis

So, even Robust Soliton does not achieve the goal of linear enc/dec complexity…

The $\ln(k/\delta)$ terms comes due to the same reason why we had ln(k) in the coupon collector problem.

Lets revisit that..

Q: Why do we need so many draws in the coupon collector problem when we want to collect **ALL** coupons?

Last few coupons require a lot of draws since...

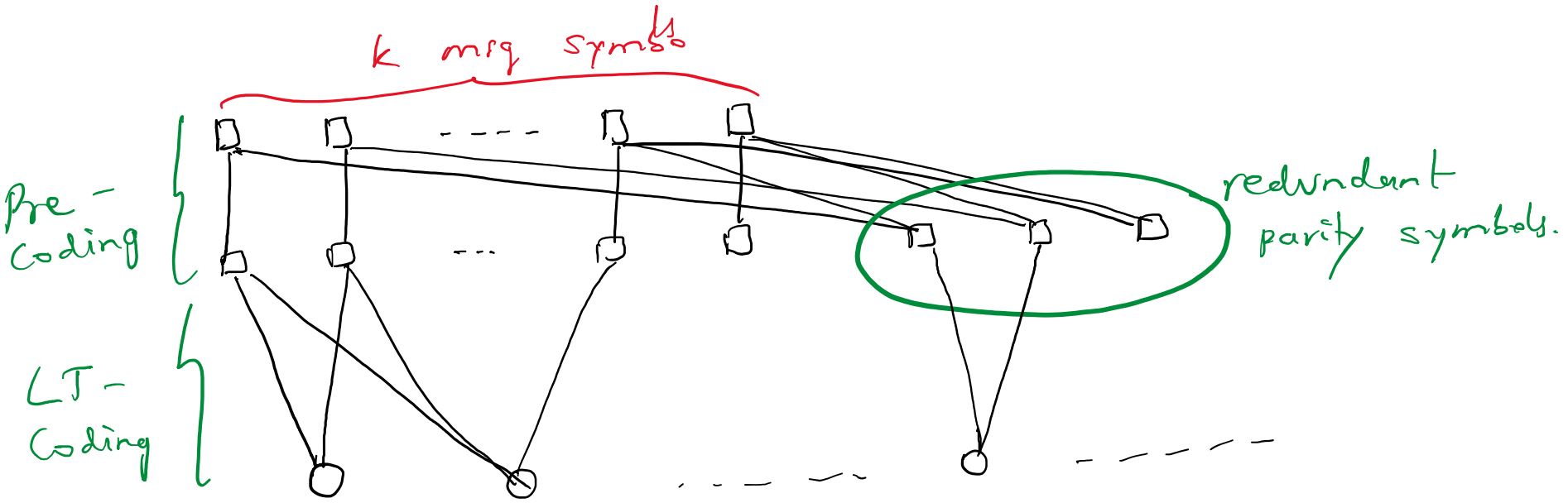
probability of seeing a distinct coupons keeps decreasing.

Raptor codes

Encode the msg symbols using an easy to decode classical code and then perform LT encoding!

"Pre-code"

Raptor Codes = Pre-code + LT encoding



Raptor codes

Theorem: Raptor codes can generate infinite stream of coded symbols s.t. for any $\epsilon > 0$

1. Any subset of size k $(1 + \epsilon)$ is sufficient to recover the original k symbols with high prob
2. Num. operations needed for each coded symbol

$O(\ln(1/\epsilon))$

3. Num. operations needed for decoding msg symbols

$O(k \ln(1/\epsilon))$

Linear encoding and decoding complexity!

Included in wireless standards, multimedia communication standards as RaptorQ

Compression

Recap

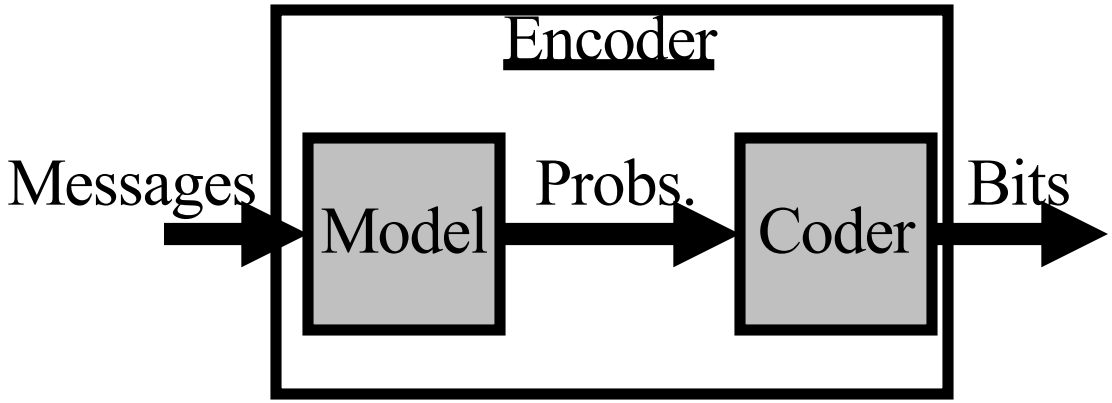
Will use **"message"** in generic sense to mean the data to be compressed

Lossless: Input message = Output message

Lossy: Input message ≈ Output message

Recap: Model vs. Coder

To compress we **need a bias on the probability of messages**. The **model** determines this bias



Recap: Entropy

For a set of messages S with probability $p(s)$, $s \in S$, the **self information** of s is:

$$i(s) = \log \frac{1}{p(s)} = -\log p(s)$$

Measured in **bits if the log is base 2**.

Entropy is the weighted average of self information.

$$H(S) = \sum_{s \in S} p(s) \log \frac{1}{p(s)}$$

Recap: Conditional Entropy

The **conditional entropy** is the weighted average of the conditional self information

$$H(S \mid C) = \sum_{c \in C} \left(p(c) \sum_{s \in S} p(s \mid c) \log \frac{1}{p(s \mid c)} \right)$$

Recap: Uniquely Decodable Codes

A **variable length code** assigns a bit string (codeword) of variable length to every message value

e.g. `a = 1, b = 01, c = 101, d = 011`

What if you get the sequence of bits `1011` ?

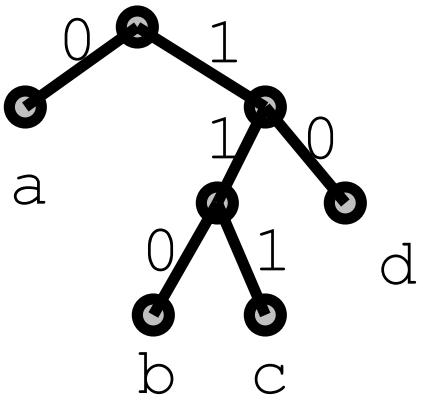
Is it `aba, ca,` or, `ad`?

A **uniquely decodable code** is a variable length code in which bit strings can always be uniquely decomposed into its codewords.

Recap: Prefix Codes

A **prefix code** is a variable length code in which no codeword is a prefix of another word.

e.g., a = 0, b = 110, c = 111, d = 10



All prefix codes are uniquely decodable

Can be viewed as a binary tree with message values at the leaves and 0s or 1s on the edges

Codeword = values along the path from root to the leaf

Recap: Average Length

Let $l(c)$ = length of the codeword c (a positive integer)

For a code C with associated probabilities $p(c)$ the **average length** is defined as

$$l_a(C) = \sum_{c \in C} p(c) l(c)$$

We say that a prefix code C is **optimal** if for all prefix codes C', $l_a(C) \leq l_a(C')$

Recap: Relationship between Average Length and Entropy

Theorem (lower bound): For any probability distribution p(S) with associated uniquely decodable code C,

$$H(S) \leq l_a(C)$$

(Shannon's source coding theorem)

Theorem (upper bound): For any probability distribution p(S) with associated optimal prefix code C,

$$l_a(C) \leq H(S) + 1$$

Kraft McMillan Inequality

Theorem (Kraft-McMillan): For any uniquely decodable code C,

$$\sum_{c \in C} 2^{-l(c)} \leq 1$$

Also, for any set of lengths L such that $\sum_{l \in L} 2^{-l} \leq 1$

there exists a prefix code C such that

$$l(c_i) = l_i (i = 1,...,|L|)$$

(We will not prove this in class. But use it to prove the upper bound on average length.)

Recap: Another property of optimal codes

Theorem: If C is an optimal prefix code for the probabilities $\{p_1, \ldots, p_n\}$ then $p_i > p_j$ implies

$$l(c_i) \leq l(c_j)$$

Proof: (by contradiction)

Recap: Huffman Codes

Huffman Algorithm:

Start with a forest of trees each consisting of a single vertex corresponding to a message s and with weight p(s)

Repeat until one tree left:

- Select two trees with minimum weight roots p_1 and p_2
- Join into single tree by adding root with weight $p_1 + p_2$

Theorem: The Huffman algorithm generates an optimal prefix code.

Proof: (by induction)

Recap: Problem with Huffman Coding

Consider a message with probability .999.

The self information of this message is

$$-\log(.999) = .00144$$

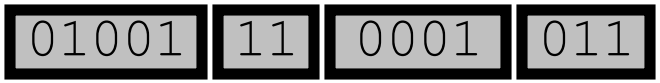
If we were to send a 1000 such message we might hope to use 1000*.0014 = 1.44 bits.

Using Huffman codes we require at least one bit per message, so we would require 1000 bits.

Recap: Discrete or Blended

Discrete: each message is a fixed set of bits

– Huffman coding, Shannon-Fano coding



message: 1 2 3 4

Blended: bits can be "shared" among messages

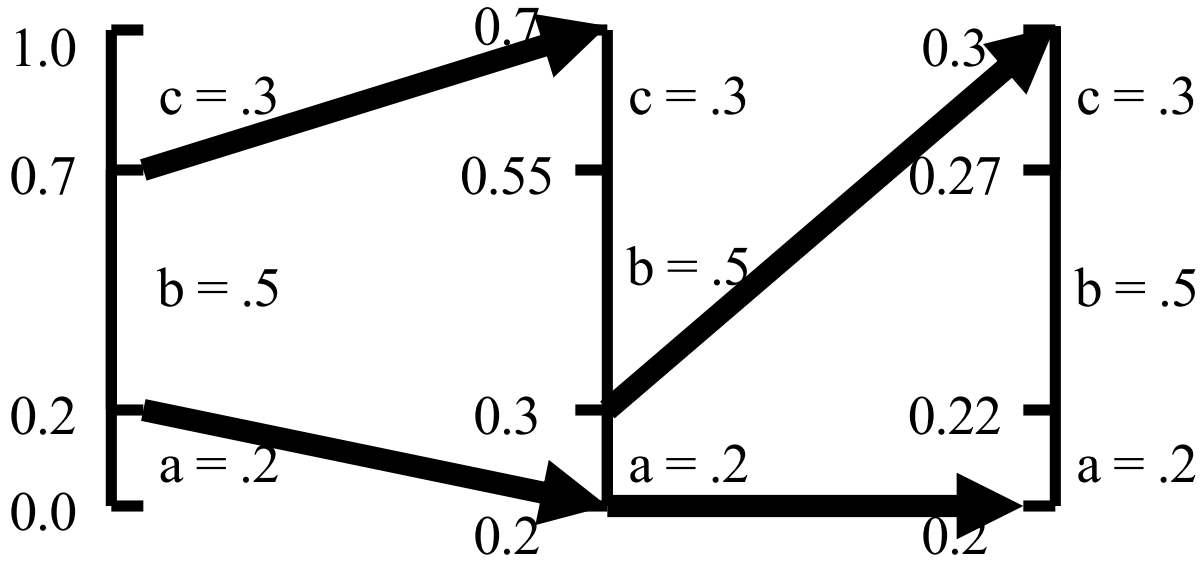
– **Arithmetic coding**

message: 1,2,3, and 4

Arithmetic Coding: sequence intervals

Code a **message sequence** by composing intervals.

For example: **bac**



The final interval is **[.27,.3)**

We call this the **sequence interval**

Recap: Exploiting context

Technique 1: transforming the data

- Run length coding (ITU Fax standard)
- Move-to-front coding (Used in Burrows-Wheeler)
- Residual coding (JPEG LS)

Technique 2: using conditional probabilities

- Fixed context (JBIG…almost)
- Partial matching (PPM)

Recap: Run Length Coding

Code by specifying message value followed by the number of repeated values:

e.g. **abbbaacccca => (a,1),(b,3),(a,2),(c,4),(a,1)**

The characters and counts can be coded based on frequency (i.e., probability coding).

Q: Why?

Typically low counts such as 1 and 2 are more common => use small number of bits overhead for these.

Used as a sub-step in many compression algorithms.

Reap: Move to Front Coding

- Transforms message sequence into sequence of integers
- Then probability code
- Takes advantage of **temporal locality**

Start with values in a total order: e.g.: [a,b,c,d,…]

For each message

- output the position in the order
- move to the front of the order.

e.g.: **c a**

c => output: 3, new order: [c,a,b,d,e,…]
a => output: 2, new order: [a,c,b,d,e,…]

Used as a sub-step in many compression algorithms.

Residual Coding

Typically used for message values that represent some sort of amplitude:

e.g. gray-level in an image, or amplitude in audio.

Basic Idea:

- Guess next value based on current context.
- Output difference between guess and actual value.
- Use probability code on the output.

E.g.: Consider compressing a stock value over time.

Residual coding is used in JPEG Lossless

Applications of Probability Coding

How do we generate the probabilities?

Using character frequencies directly does not work very well (e.g. 4.5 bits/char for text).

Technique 1: transforming the data

- Run length coding (ITU Fax standard)
- Move-to-front coding (Used in Burrows-Wheeler)
- Residual coding (JPEG LS)

Technique 2: using conditional probabilities

- Fixed context (JBIG…almost)
  - → in reading notes
- Partial matching (PPM)

Recap:
PPM: Using Conditional Probabilities

Makes use of conditional probabilities

- Use **previous *k* characters as context**.

Builds a context table

Each context has its own probability distribution

Some challenges in design:

- Conditional prob. for unseen context
- Avoid sending multiple escapes

Recap: Lempel-Ziv Algorithms

Dictionary-based approach

Codes groups of characters at a time (unlike PPM)

High level idea:

- Look for longest match in the preceding text for the string starting at the current position
- Output the position of that string
- Move past the match
- Repeat

Gets theoretically optimal compression for (really) long strings

Recap: Burrows -Wheeler

Breaks file into fixed-size blocks and encodes each block separately.

For each block:

- Create full context for each character (wraps around)
- Reverse lexical sort each character by its full context.

Then use move-to-front transform on the sorted characters.

Recap: Burrows -Wheeler

Context	Char
ecode$_6$	d$_1$
coded$_1$	e$_2$
odede$_2$	c$_3$
dedec$_3$	o$_4$
edeco$_4$	d$_5$
decod$_5$	e$_6$

Sort Context →

Context	Output	
dedec$_3$	o$_4$	
coded$_1$	e$_2$	
decod$_5$	e$_6$	
odede$_2$	c$_3$	
ecode$_6$	d$_1$	⇐
edeco$_4$	d$_5$	

Gets similar characters together
(because we are ordering by context)

Can be viewed as giving a dynamically sized context.
(overcoming the problem of choosing the right “k” in PPM)

Recap: Inverting BW Transform

Context	Output	
c_3	o_4	
d_1	e_2	
d_5	e_6	
e_2	c_3	
e_6	d_1	⇐
o_4	d_5	

Sort the output column to get the last column of the context!

Theorem: After sorting, equal valued characters appear in the same order in the output column as in the last column of the sorted context.

Multiple ideas used in practice
Example: BZIP

Transform 1: (Burrows Wheeler)
- **input** : character string (block)
- **output** : reordered character string

Transform 2: (move to front)
- **input** : character string
- **output** : MTF numbering

Transform 3: (run length)
- **input** : MTF numbering
- **output** : sequence of run lengths

Probabilities: (on run lengths)

Dynamic based on counts for each block.

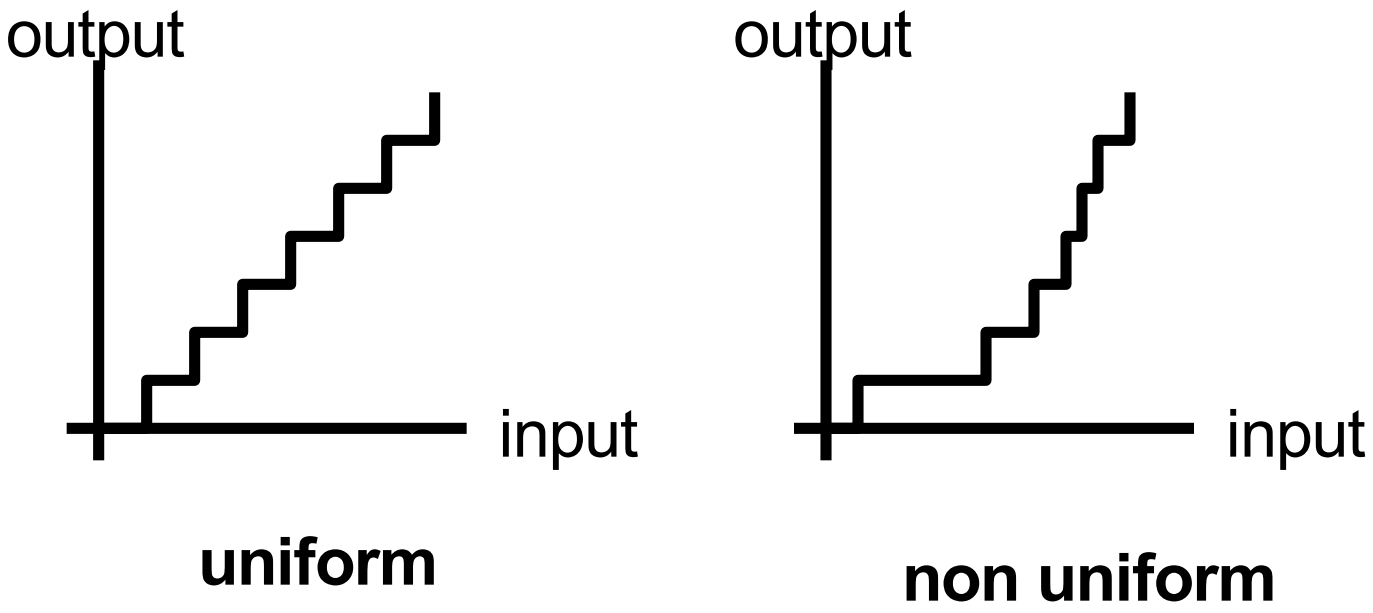
Coding: Originally arithmetic, but changed to Huffman in bzip2 due to patent concerns

Lossy compression

1. Scalar quantization:
   – Quantize regions of values into a single value

2. Vector quantization
   – Quantizing vectors rather than single values

Scalar Quantization

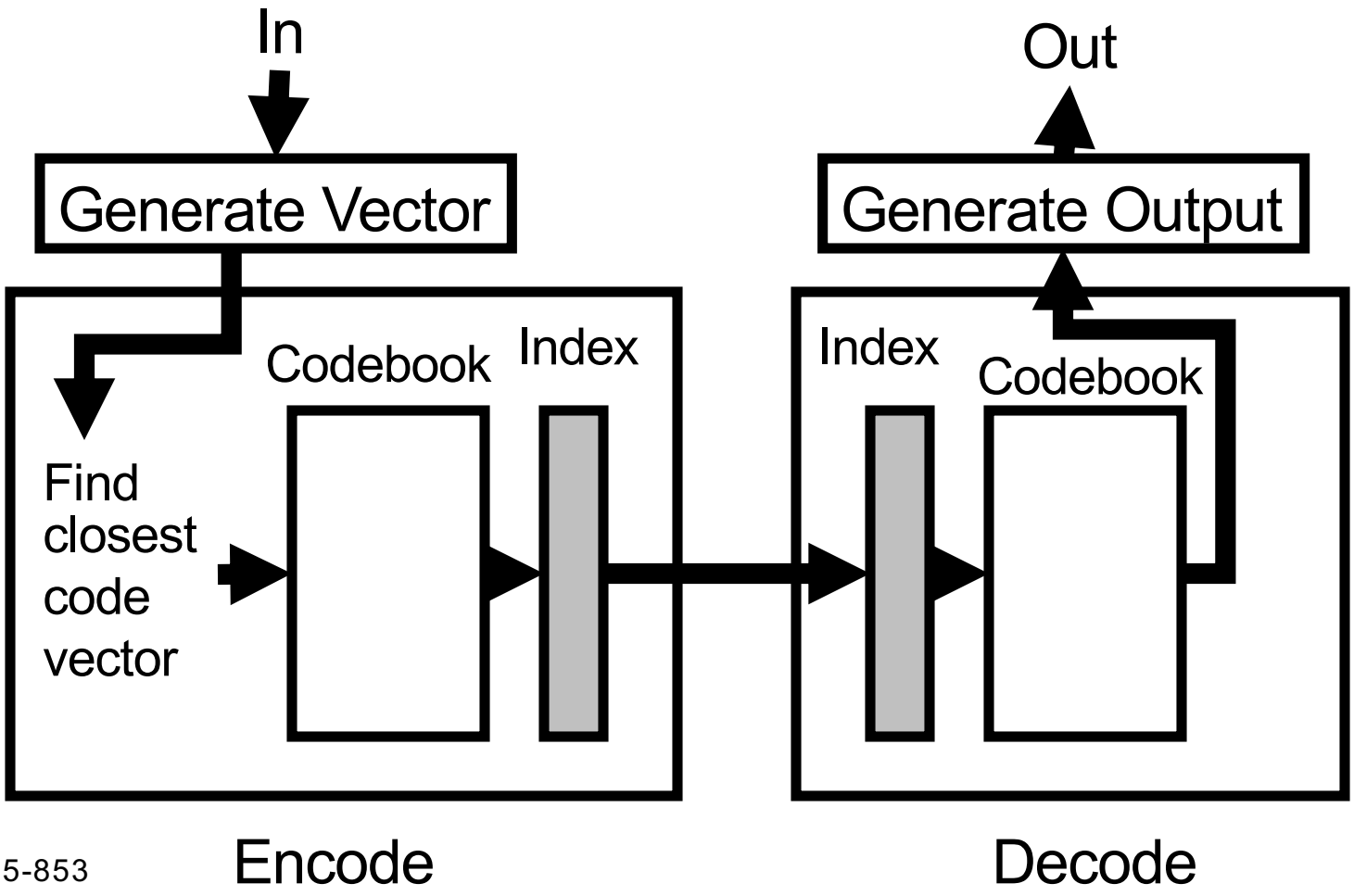


Q: Why use non-uniform?
Error metric might be non-uniform.
E.g. Human eye sensitivity to specific color regions

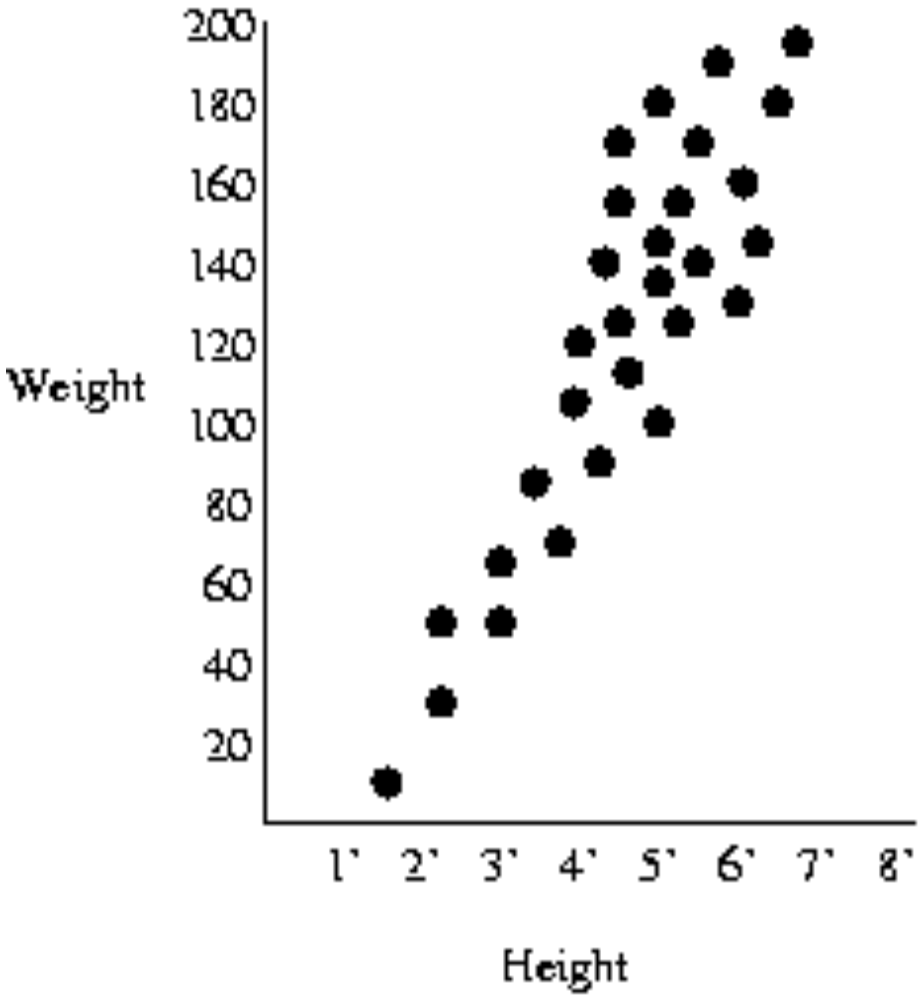
Can formalize the mapping problem as an optimization problem

Vector Quantization

Mapping a multi-dimensional space into a smaller set of messages



Vector Quantization: Example



Observations:

1. Highly correlated:
   Concentration of representative points

2. Higher density is more common regions.

Linear Transform Coding

Goal: Transform the data into a form that is easily compressible (through lossless or lossy compression)

Select a set of linear basis functions ϕ_i that span the space

- sin, cos, spherical harmonics, wavelets, …

After the transformation, the data is more easier to compress

Cryptography

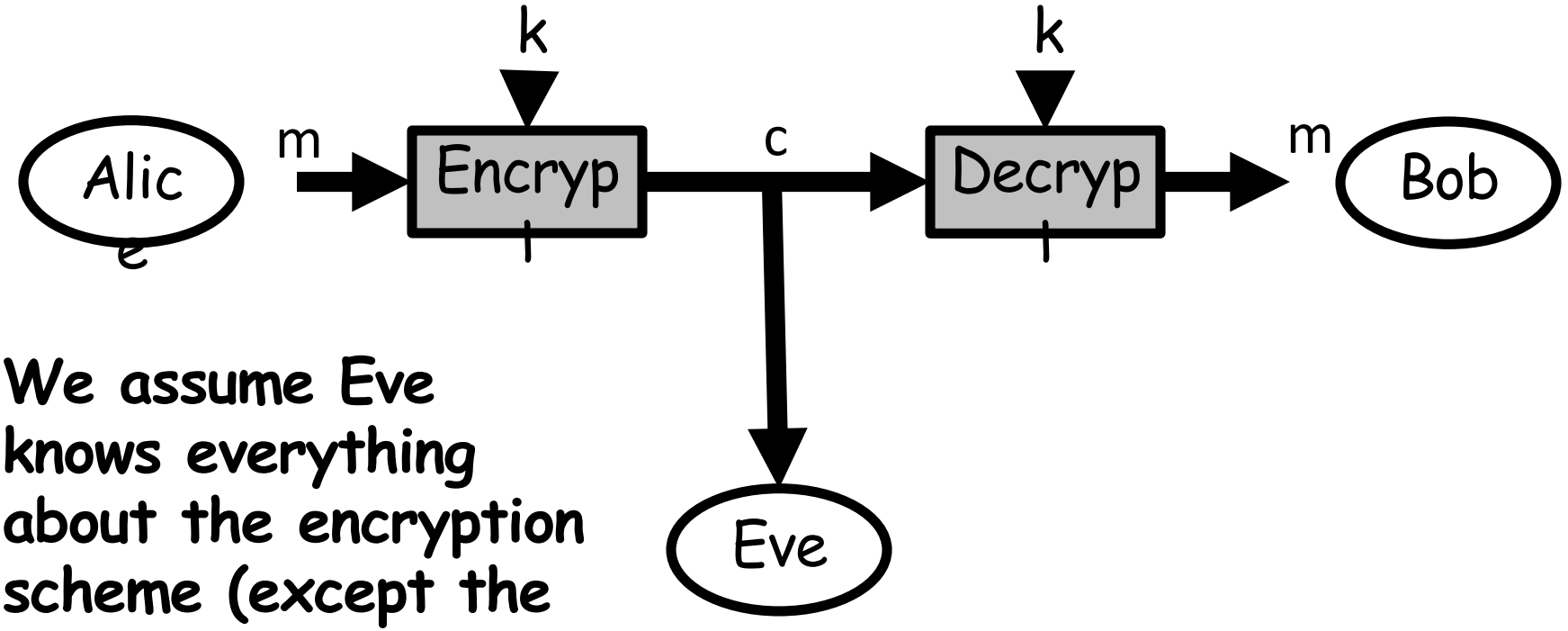
Cryptography Outline

Private-Key Encryption: One-Time Pad, Rijndael, DES

Number Theory

Public-Key Encryption: RSA, ElGamal, Diffie-Hellman

Private key encryption



We assume Eve knows everything about the encryption scheme (except the secret key)

Perfect secrecy

- Let M, C be r.v.s for the message and ciphertext.
- For every message **m** and ciphertext **c** with Pr[C=**c**] > 0:

$$\Pr[M = \mathbf{m} \mid C = \mathbf{c}] = \Pr[M = \mathbf{m}]$$

- Ciphertext contains no information about message!

One-time pad

- Key generation:
  - Input: length n (in unary)
  - Output: uniformly random $k \in \{0,1\}^n$
- Encryption:
  - Input: $m \in \{0,1\}^n$, $k \in \{0,1\}^n$
  - Output: $c = m \oplus k$
- Decryption:
  - Input: $c \in \{0,1\}^n$, $k \in \{0,1\}^n$
  - Output: $m = c \oplus k$
- One-Time pad is perfectly secret.

Computational secrecy

- Perfect secrecy requires the key to be at least as long as the message. This is impractical!
- We need to settle for a **weaker definition**.
  - Any **efficient** adversary succeeds in breaking the scheme with at most **negligible** probability.
- Efficient = runs in probabilistic polynomial time (PPT).
- Negligible = goes to zero faster than any inverse poly:
  - A positive function f is **negligible** if for every positive integer c, there exists N_c such that:

$$f(n) < n^{-c}, \quad \text{for all } n > N_c$$

  - Denoted as f = negl(n).

Private Key: Block Ciphers

A **Block cipher** C is a function with:

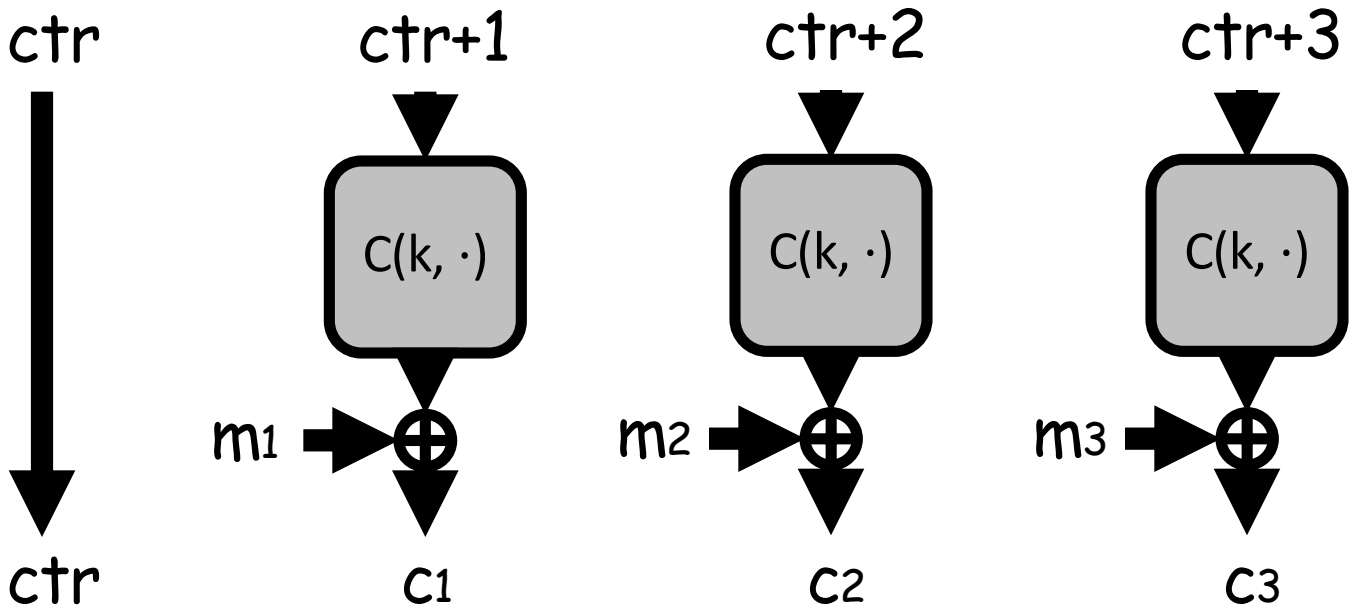
- **Input**: a key $k \in \{0,1\}^{|k|}$, block $x \in \{0,1\}^n$ (with $|k| \leq n$)
- **Output**: a block $y \in \{0,1\}^n$

- **Objective**: should be hard to distinguish from a random permutation from $\{0,1\}^n$ to $\{0,1\}^n$.

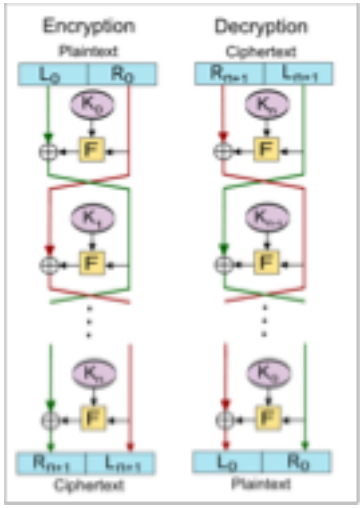
Private Key: Block Ciphers

Intuition: generate a “fresh” one-time pad for each block.

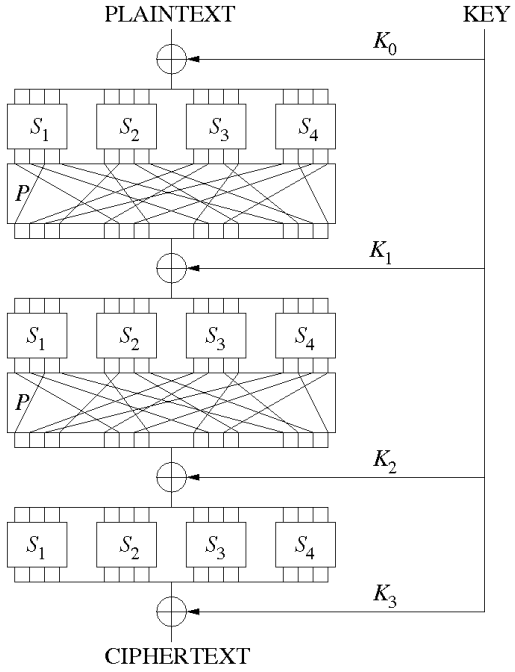
Counter (CTR) mode:



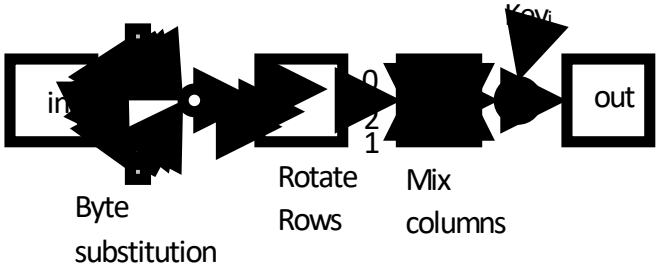
Block cipher implementations



Feistel Networks



Substitution Permutation Network



Rijndael/AES

Groups

A **Group** (G,*,I) is a set *G* with operator * such that:

1. **Closure**. For all $a,b \in G$, $a * b \in G$
2. **Associativity.** For all $a,b,c \in G$, $a*(b*c) = (a*b)*c$
3. **Identity.** There exists $I \in G$, such that for all $a \in G$, $a*I=I*a=a$
4. **Inverse.** For every $a \in G$, there exist a unique element $b \in G$, such that $a*b=b*a=I$

An **Abelian or Commutative Group** is a Group with the additional condition

5. **Commutativity.** For all $a,b \in G$, $a*b=b*a$

The Euler Phi Function

$$\phi(n) = \left|Z_n^*\right| = n\prod_{p|n}(1-1/p)$$

If n is a product of two primes p and q, then

$$\phi(n) = pq(1-1/p)(1-1/q) = (p-1)(q-1)$$

Fermat-Euler Theorem:

$$a^{\phi(n)} = 1 \pmod{n} \text{ for } a \in Z_n^*$$

Or for n = pq

$$a^{(p-1)(q-1)} = 1 \pmod{n} \text{ for } a \in Z_{pq}^*$$

This will be very important in RSA!

Diffie-Hellman Key Exchange

Can A and B agree on a secret through a public channel?

A group (G,*) and a generator g are made public.

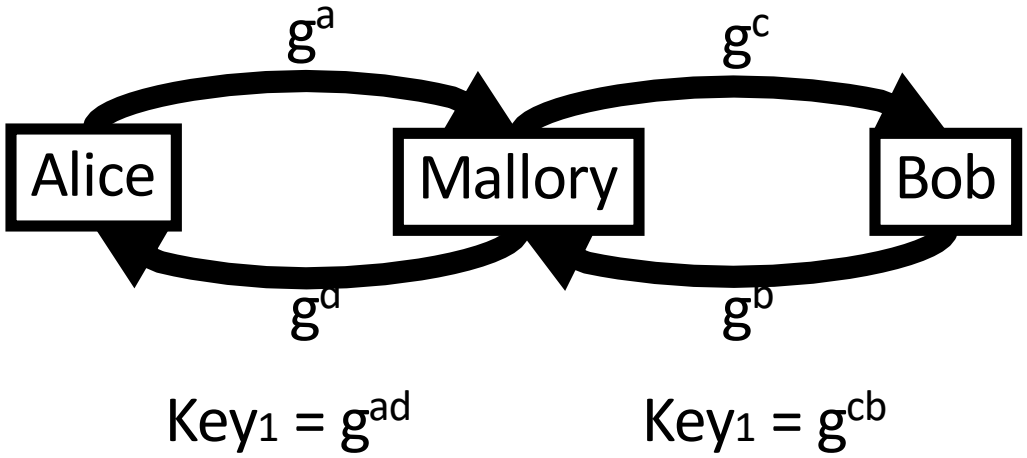
- Alice picks a, and sends g^a to Bob
- Bob picks b and sends g^b to Alice
- The shared key is g^{ab}

The shared key is easy for Alice or Bob to compute, but (we believe) it's hard for Eve to compute g^{ab} from (g, g^a, g^b).

If Discrete Log is easy, this protocol is broken.

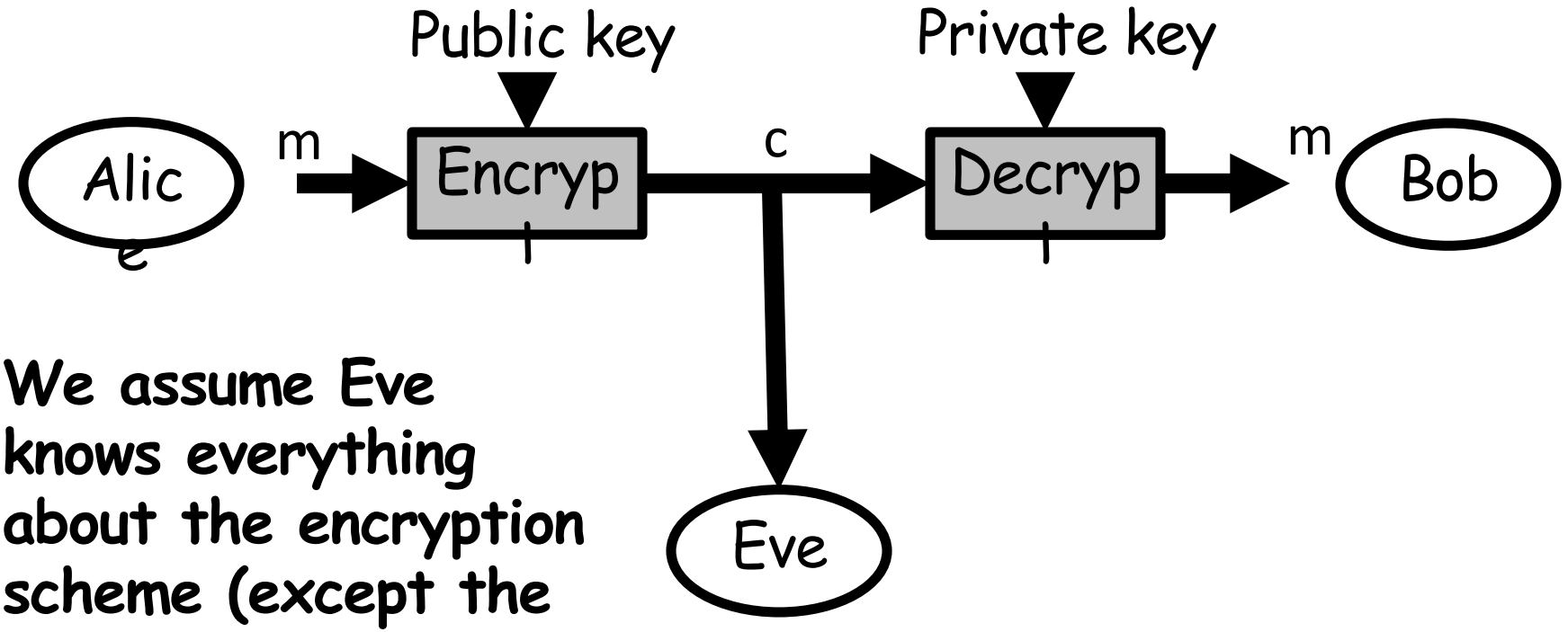
What could go wrong with this protocol?

Person-in-the-middle attack



Mallory could impersonate Alice or Bob!
This is a problem in general, but later we will see how it's solved in practice for public key crypto.

Public key encryption



We assume Eve knows everything about the encryption scheme (except the private key)

ElGamal Public-key Cryptosystem

(G,*) is a group

- α a generator for G
- a $\in Z_{|G|}$
- $\beta = \alpha^a$

G is selected so that it is hard to solve the discrete log problem.

Public Key: (α, β) and some description of G

Private Key: a

Encode:

Pick random r $\in Z_{|G|}$

$E(m) = (y_1, y_2)$
$= (\alpha^r, m * \beta^r)$

Decode:

$D(y) = y_2 * (y_1^a)^{-1}$
$= (m * \beta^r) * (\alpha^{ra})^{-1}$
$= m * \beta^r * (\beta^r)^{-1}$
$= m$

You need to know a to easily decode y!

RSA Public-key Cryptosystem

What we need:

- p and q, primes of approximately the same size
- $n = pq$
  $\phi(n) = (p-1)(q-1)$
- $e \in Z_{\phi(n)}^*$
- $d = e^{-1} \bmod \phi(n)$

Public Key: (e,n)
Private Key: d

Encode:
$m \in Z_n$
$E(m) = m^e \bmod n$

Decode:
$D(c) = c^d \bmod n$

Hashing

Concentration Bounds

Central question:

What is the probability that a R.V. deviates much from its expected value

- Typically want to say a R.V. stays “close to” it expectation “most of the time”

Useful in analysis of randomized algorithms

Markov's Inequality

The most basic concentration bound.

Let X be a **non-negative** R.V. with mean μ then

$$P(X \geq \alpha) \leq \frac{\mu}{\alpha}$$

Proof: (Did last class)

In other terms,

$$P(X \geq k\mu) \leq \frac{1}{k}$$

Uses expectation only

Chebyshev's Inequality

More powerful than Markov's

Let X be a R.V. with mean μ and variance σ^2

$$P(|X-\mu| \geq \epsilon) \leq \frac{\sigma^2}{\epsilon^2}$$

Proof: Ideas ?

In other terms,

$$P(|X-\mu| \geq k\sigma) \leq \frac{1}{k^2}$$

Stronger since it uses variance information
Smaller the variance more concentrated the R.V. around mean

Chernoff Bound

For any R.V. X, for any t>0

$$P(X \geq a) \leq \frac{E[e^{tX}]}{e^{ta}}$$

$$\Rightarrow P(X \geq a) \leq \min_{t>0} \frac{E[e^{tX}]}{e^{ta}}$$

There are many different variants of Chernoff bounds applied to various different distributions

Chernoff Bounds for Binomial

Binomial = sum of Bernoulli (i.e. Binary valued) R.V.s

Let $X = \sum_{i=1}^{n} X_i$

Where X_i's = Bernoulli (p) and independent.

$\mu = E[X] = np$

Then for all $\delta > 0$

$$P(X - np \geq \delta) \leq e^{-\frac{2\delta^2}{n}}$$

$$P(X - np \leq -\delta) \leq e^{-\frac{2\delta^2}{n}}$$

Chernoff Bounds for Binomial

Binomial = sum of Bernoulli (I.e. Binary valued) R.V.s

Let $X = \sum_{i=1}^{n} X_i$

Where X_i's = Bernoulli (p_i) and independent (more general)

$\mu = E[X] = \sum_{i=1}^{n} p_i$

Then for all $\delta > 0$

$$P(X \geq (1+\delta)\mu) \leq C_\delta^{\mu}$$

where $$C_\delta = \frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}$$

Recap: Load balancing

N balls and N bins

Randomly put balls into bins

Theorem: The max-loaded bin has $O(\frac{\log N}{\log \log N})$ balls with probability at least 1 – 1/N.

Proof. High level steps:

1. We will first look at probability of any particular bin receiving more than $O(\frac{\log N}{\log \log N})$ balls.
2. Then we will look at the probability of there being a (i.e., at least one) bin with more than these many balls.

Load balancing

Another useful and interesting result.

It turns out that the **bound is tight**!

Theorem. With high probability the max load is $\Omega\left(\frac{\log n}{\log \log n}\right)$

Uniformly randomly placing balls into bins does not balance the load after all!

Load balancing: power-of-2-choice

When a ball comes in, pick two bins and place the ball in the bin with smaller number of balls.

Turns out with just **checking two bins** maximum number of balls drops to **O(log log n)**!

=> called “power-of-2-choices”

Intuition: Ideas?

Even though max loaded bins has $O(\frac{\log N}{\log \log N})$ balls, most bins have far fewer balls.

Load balancing: power-of-d-choice

When a ball comes in, **pick d bins** and place the ball in the bin with smallest number of balls.

Theorem:

For any d>=2 the d-choice process gives a maximum load of

$$\frac{\log \log N}{\log d} \pm O(1)$$

with probability at least 1 – O(1/N)

Observations:

Just looking at two bins gives huge improvement.

Diminishing returns for looking at more than 2 bins.

Rest of the topics in hashing and in dimension reduction where covered recently in class.

I expect them to be fresh in your minds and hence not doing a recap of those topics.