

What can cryptography do for coding theory?

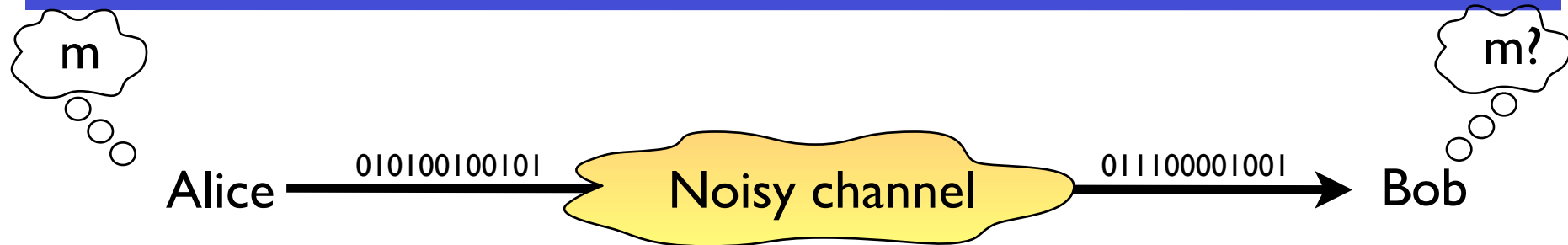
Adam Smith

Computer Science & Engineering Department
Penn State

<http://www.cse.psu.edu/~asmith>

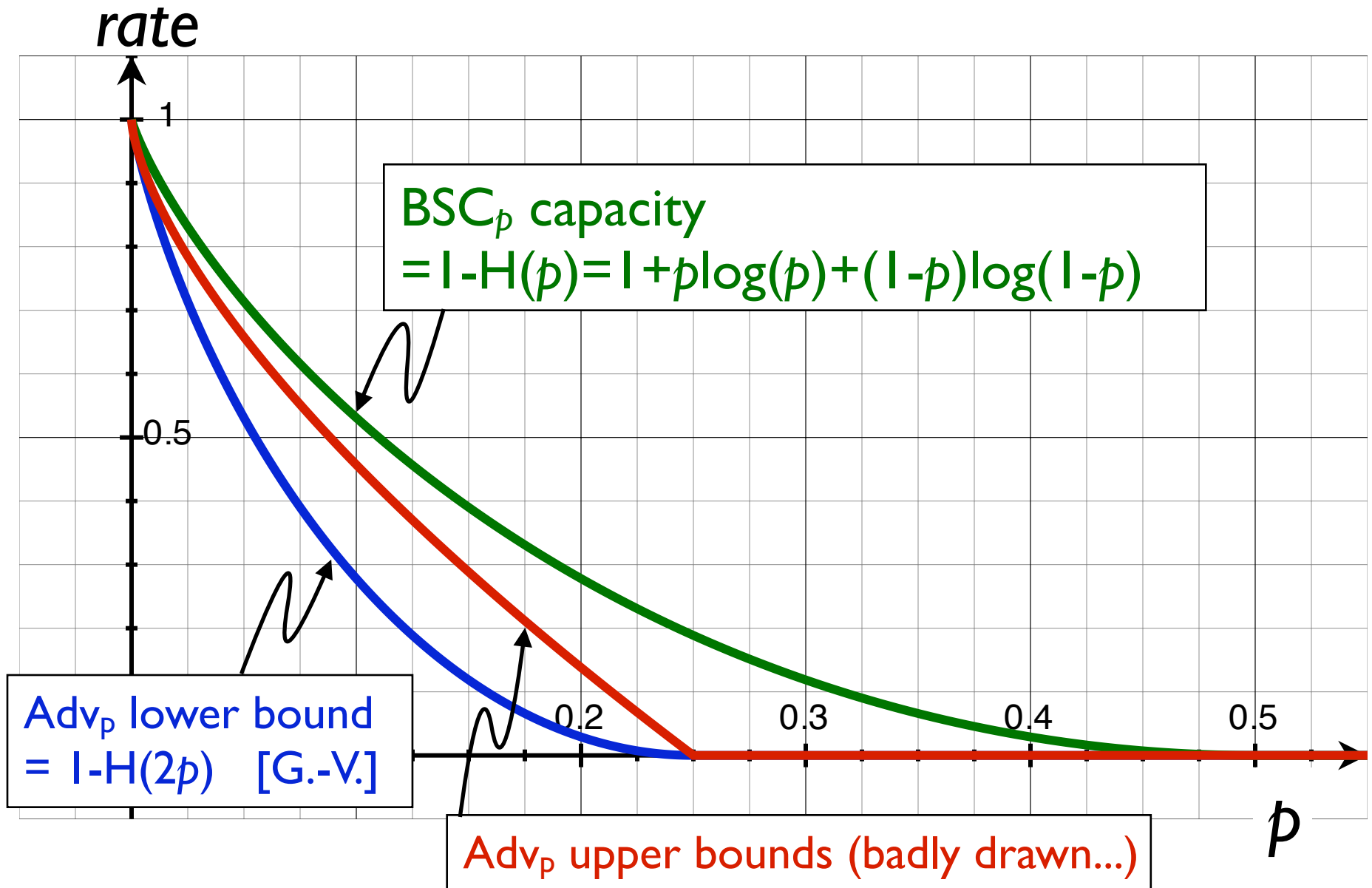
ICITS 2009

Two classic channel models



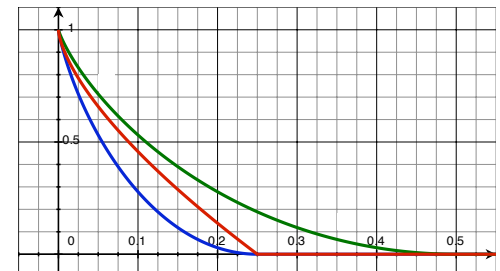
- Alice sends n bits
- Binary symmetric channel BSC_p
 - Flips each bit with probability p
 - Shannon: maximum possible rate is $1-H(p)$
 - Forney: concatenated codes achieve capacity efficiently
- Worst-case (adversarial) errors ADV_p
 - Channel outputs an arbitrary word within distance pn of input
 - Optimal rate still unknown

Known Bounds



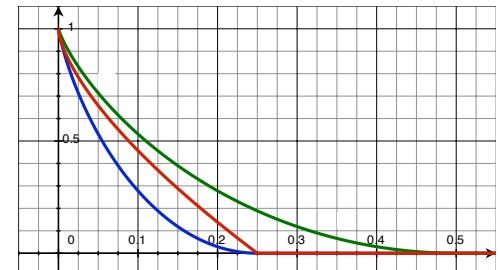
Why care about worst-case errors?

- Combinatorial interest
 - key building block for designs, authentication schemes, etc
- Modeling **unknown** or **varying** channels
 - Codes designed for one channel may fail if model wrong
 - E.g., concatenated codes do badly against bursty errors



This talk: cryptographic tools in coding

- Models of uncertain binary channels
 - strong enough to capture wide variety of channel behavior
 - But reliable communication at Shannon capacity
- Theme: cryptographic perspective
 - modeling “limited” adversarial behavior
 - simpler existence proofs
 - techniques for efficient constructions: indistinguishability, pseudorandomness



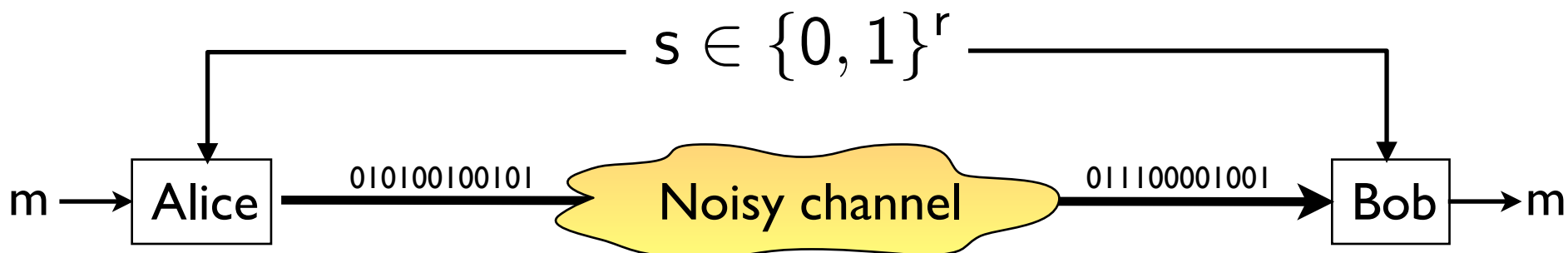
-
- Two kinds of models:
 - shared secrets for Alice and Bob
 - limited channels
 - Two basic techniques
 - Sieving list decodable codes
 - “Scrambling” (randomizing) adversarial errors

Outline

- Developing tools: shared secrets
- Computationally limited channels
- Recent results:
 - Explicit constructions for worst-case “additive” errors [GS’09]
 - Logspace channels [forthcoming]

Shared Randomness

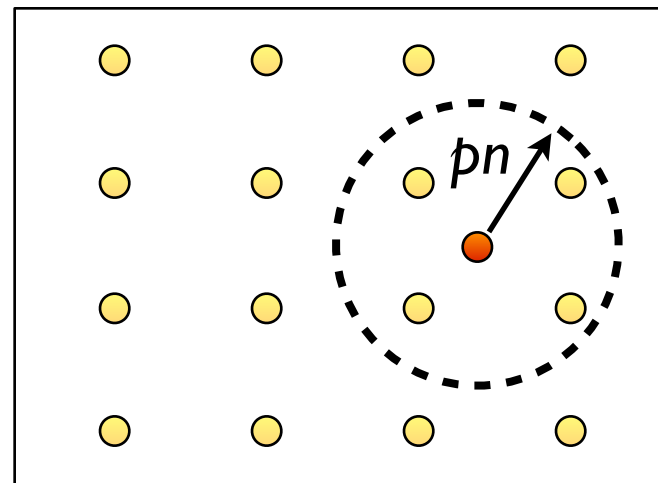
Shared Randomness



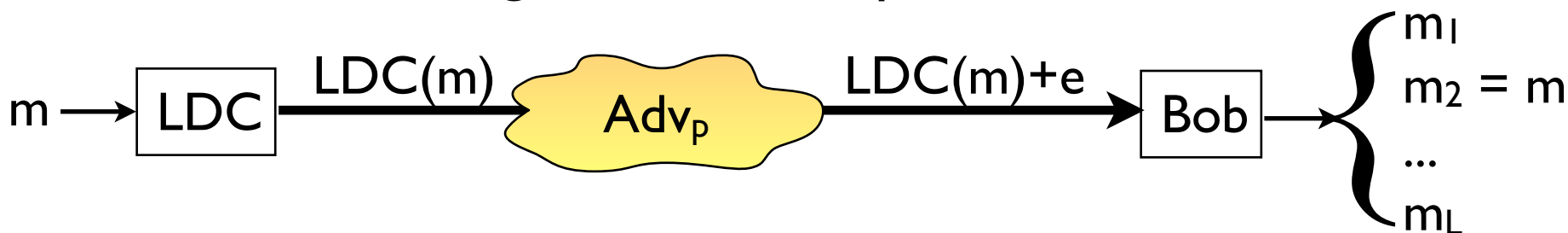
- Encoder/decoder share random bits s
 - code is known to channel but s is unknown
- **Theorem I** [Langberg '04, ?]: With $r=O(\log n)$ shared bits, Alice can send $\approx n(1-H(p))$ bits reliably over Adv_p .
 - (not necessarily computationally efficient)
- A simple “cryptographic” proof
 - Tools: list-decoding, message authentication

Tool: List-decodable codes

- A code LDC: $\{0, 1\}^k \rightarrow \{0, 1\}^n$ is (pn, L) list-decodable code if
 - Every vector in $\{0, 1\}^n$ is within distance pn of at most L codewords



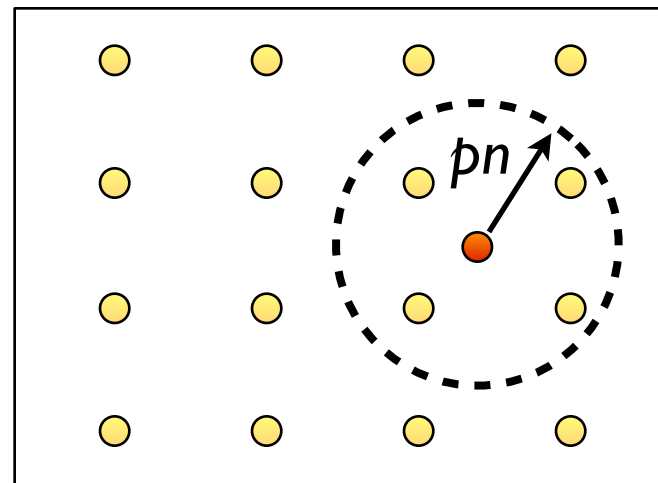
- With LDC, Bob gets a list of L possible codewords



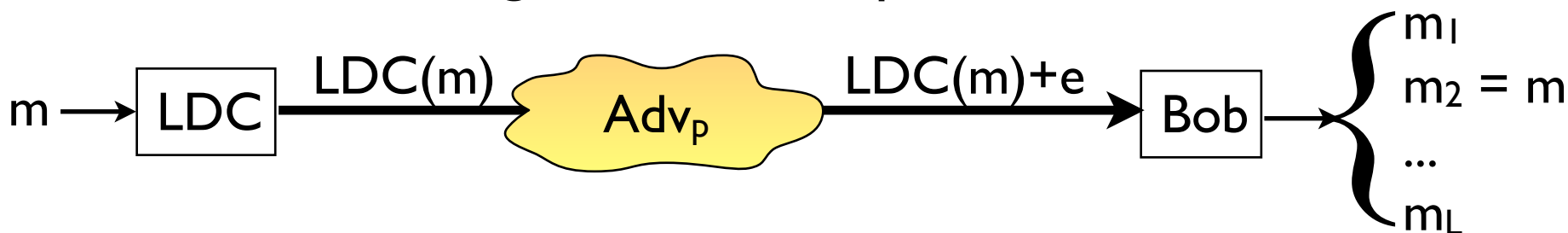
- **Proposition [Elias]:** There exist (pn, L) list-decodable codes with rate $1 - H(p) - \epsilon$ and list size $L = 1/\epsilon$.

Tool: List-decodable codes

- A code LDC: $\{0, 1\}^k \rightarrow \{0, 1\}^n$ is (pn, L) list-decodable code if
 - Every vector in $\{0, 1\}^n$ is within distance pn of at most L codewords



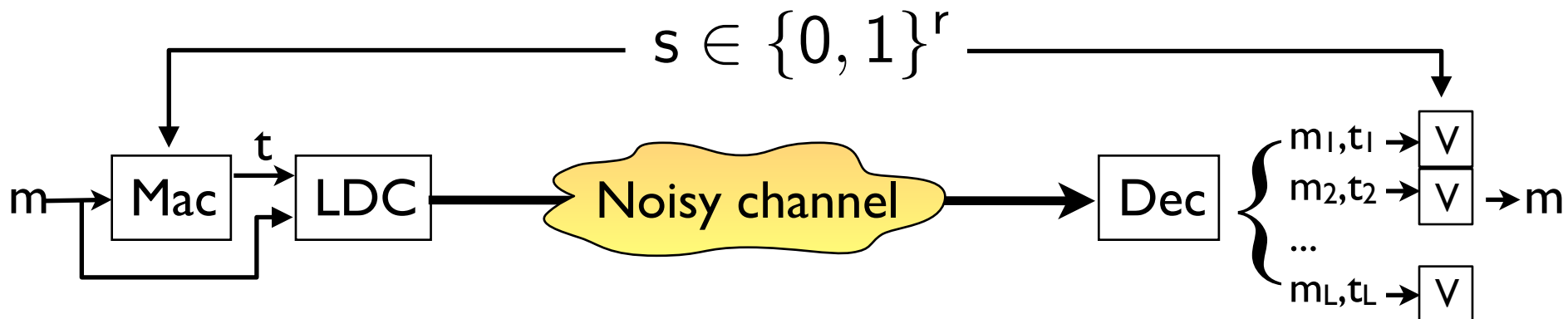
- With LDC, Bob gets a list of L possible codewords



- **Proposition [Elias]:** There exist (pn, L) list-decodable codes with rate $1 - H(p) - \epsilon$ and list size $L = 1/\epsilon$.

How can Bob figure out which is the right codeword?

Sieving the List



- Idea: Alice authenticates m using s as key
- **Theorem 1** [Langberg '04, ?]: With $r=O(\log n)$ shared bits, Alice can send $\approx n(1-H(p))$ bits reliably over Adv_p .
- **Proof:** If MAC has forgery probability δ , then Bob corrects Adv_p errors with probability $\leq L \delta$
 - Adversary gets at most L chances to forge
 - MAC tag can have tag/key length $O(\log n)$

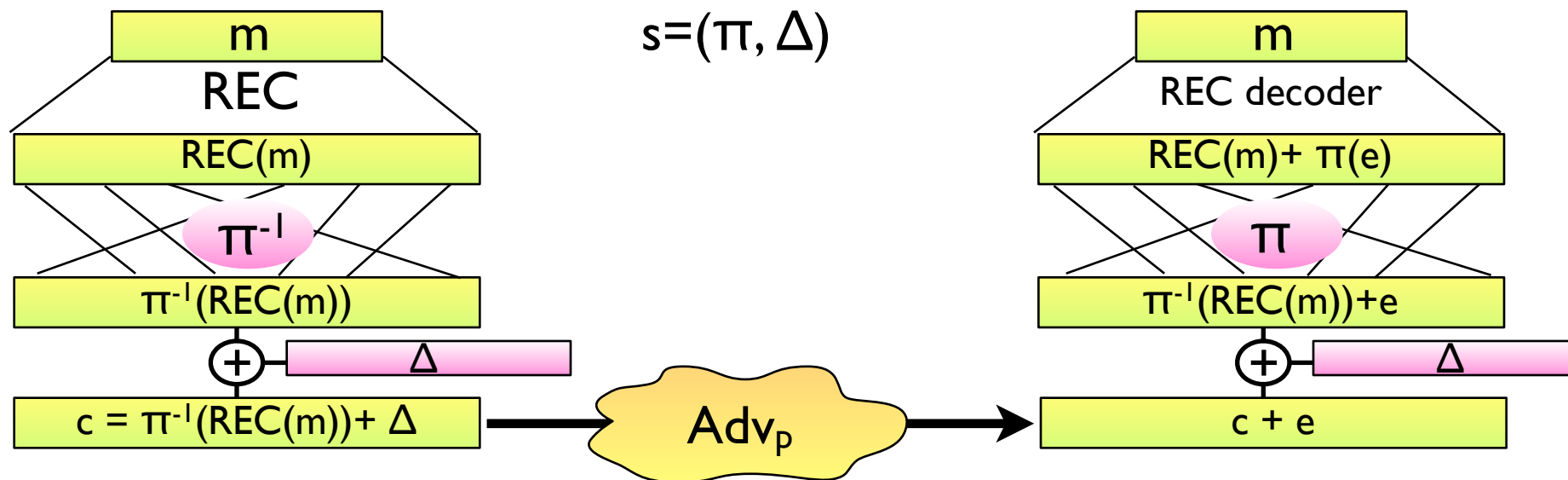
Computational Efficiency?

- Problem with list-decoding: **efficient** constructions only known for $p \approx 0$ and $p \approx 1/2$
 - for other values of p , efficient constructions have rate well below capacity



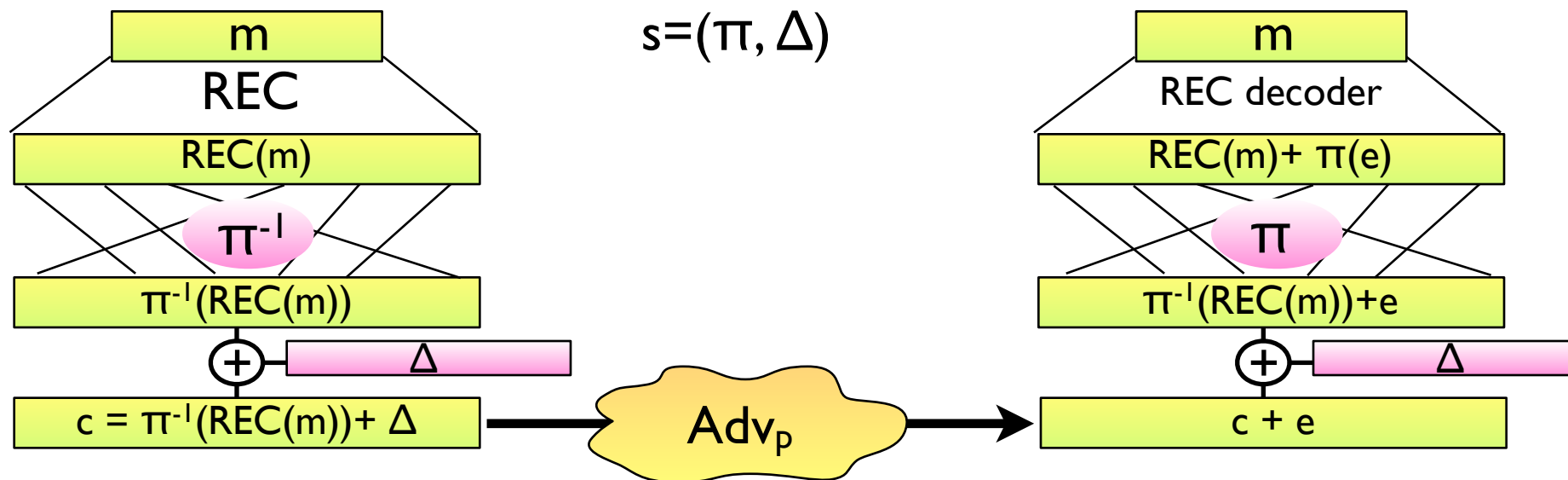
- **Theorem 2** [Lipton'94]: With $r \approx n \log(n)$ shared bits, Alice can Bob can efficiently and reliably communicate $\approx n(1-H(p))$ bits over Adv_p

Technique #2: Code Scrambling



- Shared randomness to permute errors randomly
 - Code REC corrects random errors with rate $1-H(p)$ [Forney]
 - $s = (\pi, \Delta)$ where π is a random permutation of $\{1, \dots, n\}$ and Δ is a random offset in $\{0, 1\}^n$
 - Encoding: $c = \pi^{-1}(REC(m)) + \Delta$

Technique #2: Code Scrambling

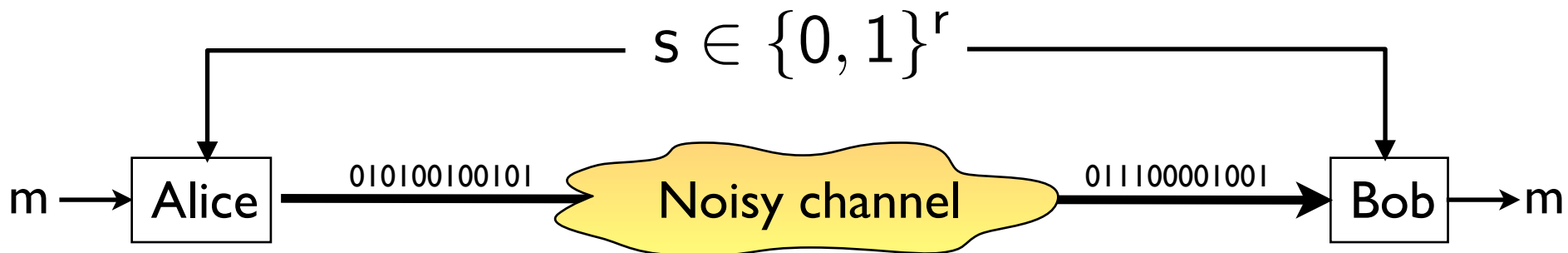


- **Theorem 2** [Lipton]: Scrambled code corrects pn adversarial errors with rate $\approx 1 - H(p)$
- **Proof:** Δ acts as one-time pad
 - e is independent of π
 - $\pi(e)$ is a uniformly random vector of same weight as e ($< pn$)

Computational Efficiency w/ Short Keys?

- Code scrambling uses a long key: $\log(n!) + n$ bits
- **Open Question:** Can we get efficient codes of rate $n(1-H(p))$ that correct pn errors with keys of $o(n)$ bits?
- **Partial Answer:** $n+o(n)$ bits of key suffice
 - π just has to be random enough to “fool” the REC decoder
 - **Lemma** [S’07]: Concatenated codes corrected $\log(n)$ -wise independent errors up to Shannon capacity
 - π only has to be a $\log(n)$ -wise independent permutation
 - **Lemma** [KNR’05]: $\log^2(n)$ bits suffice to select π
 - Get keys of length $n + \log^2(n)$... bottleneck is one-time pad!

Shared Randomness



- Can correct adversarial errors up to Shannon capacity
 - Two techniques: sieving list and code scrambling

Scheme	Key length	Efficient?
Sieving list	$\log(n)$	No*
Scrambling	$n \log(n)$	Yes
Scrambling with t -wise π	$n + \log^2(n)$	Yes

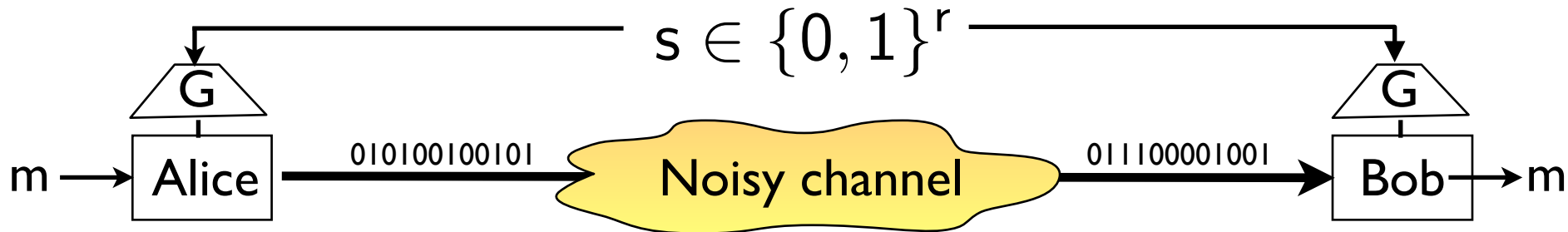
Limited Channels

Limited channels

- Idea: consider adversarial yet limited class of channels
 - processes in nature may vary in strange ways
but they are **computationally simple**
- Polynomial-time channels [Lipton]
 - Can strengthen results for shared randomness model
 - Models with no setup?
- Additive channels [A,CN]
 - Model noise that is oblivious to individual bits
 - Explicit, poly-time constructions

Polynomial-time Adversaries

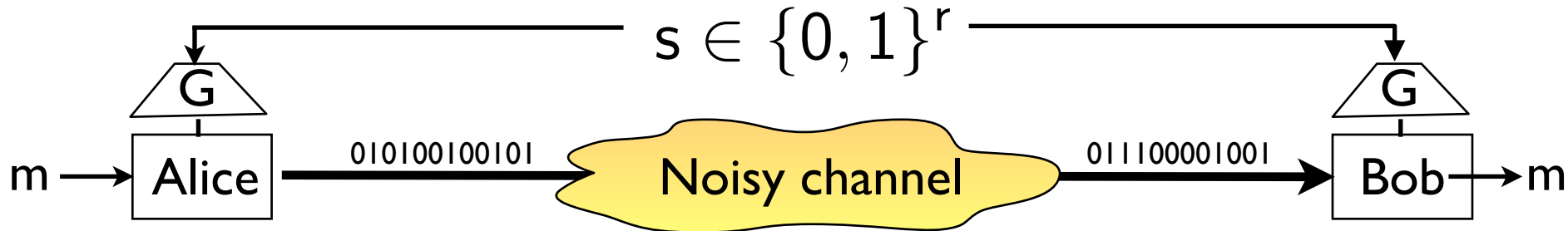
- Shared key setting [Lipton]
 - Use a p.r.g. to do code scrambling with a short seed
 - Get $O(\log n)$ -bit keys and efficient decoding (assuming OWF)



Polynomial-time Adversaries

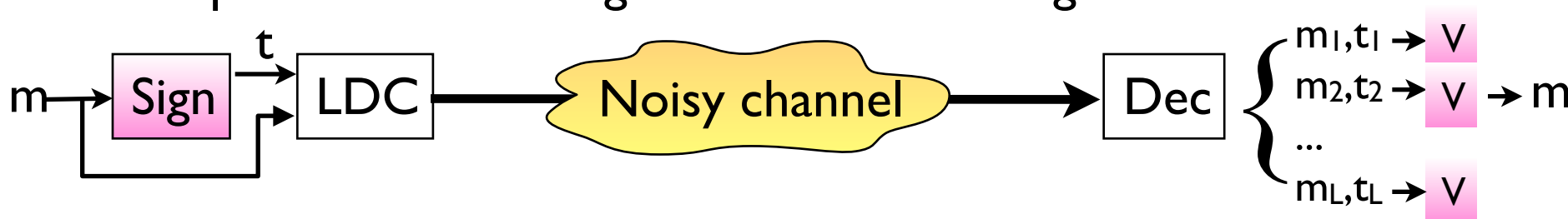
- Shared key setting [Lipton]

- Use a p.r.g. to do code scrambling with a short seed
- Get $O(\log n)$ -bit keys and efficient decoding (assuming OWF)



- Public key setting [Micali, Peikert, Sudan, Wilson]

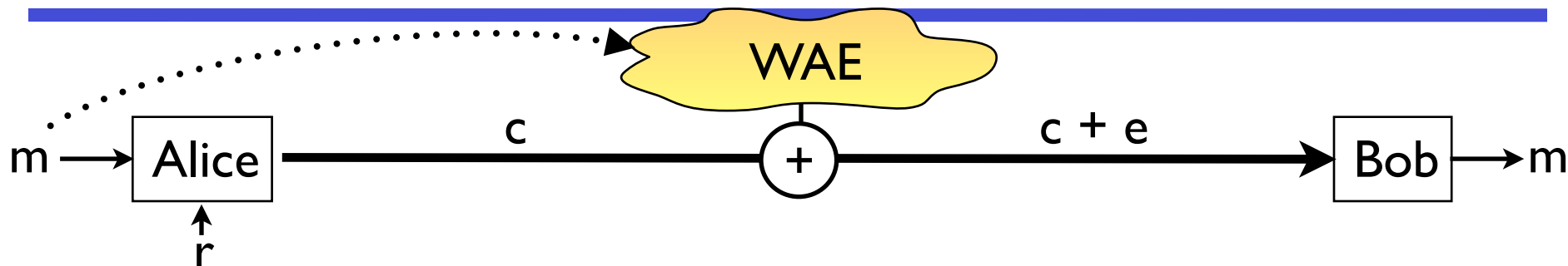
- Alice broadcasts a public key; keeps a secret key
- Replace MAC with signatures in list-sieving



What about models without setup?

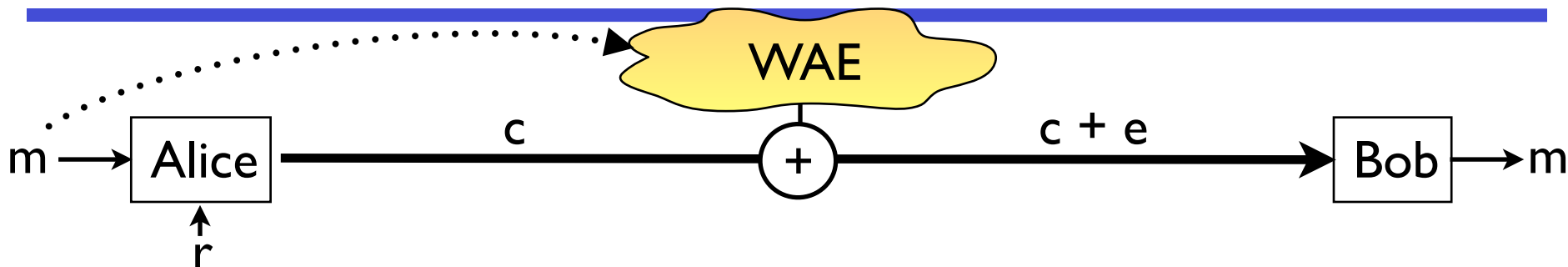
- Nothing (significant) is known regarding polynomial time
 - Some bounds clearly apply (e.g. Shannon bound)
 - Unclear if one can beat information-theoretic bounds for adversarial channels
- Different extreme: **additive channels**
 - **very simple** channels
 - error pattern is adversarial, but independent of codeword

Worst-case additive errors



- Adversary picks error pattern e of weight $< pn$
before seeing codeword
 - Adversary knows **code** and **message**
 - Alice generates local random bits r (unknown to Bob/channel)
- Generalizes natural symmetric error models
 - e.g., BSC, burst errors
- Natural step towards general classes of channels
- Special case of **state-constrained AVC** [Csiszár-Narayan]

Worst-case additive errors



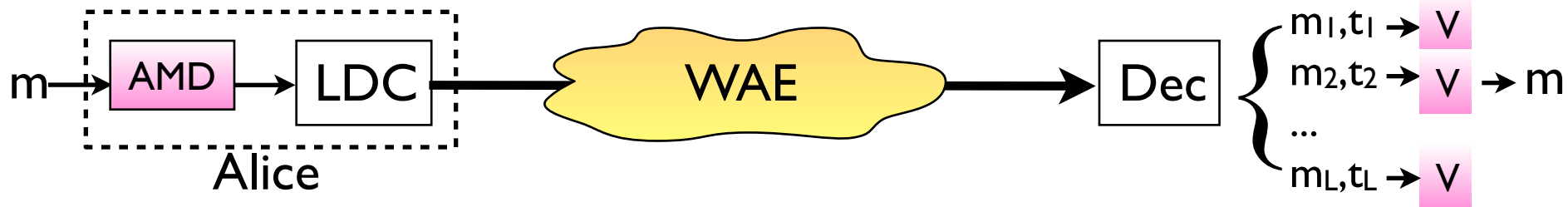
- AVC's literature has general upper/lower bounds
- **Theorem 3** [Csiszár-Narayan, Langberg]: There exist codes with rate $\approx 1 - H(p)$ that correct pn additive errors.
 - Complex random coding arguments
- [Guruswami-S., '09]: This talk
 - Simpler existence proof via sieving LDC's
 - Explicit construction with efficient encoding / decoding

Tool: Algebraic Manipulation Detection [CDFPW'08]

- “Error detection” for additive errors
- Randomized encoding AMD: $m \mapsto \text{AMD}(m,r)$
 - $\text{Verify}(\text{AMD}(m,r)) = 1$ always
 - For all fixed error patterns e , w.h.p. over r ,
 $\text{Verify}(\text{AMD}(m,r)+e) = \text{false}$
- Simple construction expands m by $O(\log(n))$ bits
 - Use m to choose coefficients of low-degree polynomial f_m
 - $\text{AMD}(m,r) = (m, r, f_m(r))$
 - **Lemma** [DKRS]: If we ensure that the leading coefficients of f_m have the right form, then for all m and for all offsets a,b,c :
 $\Pr(f_{m+a}(r+b) = f_m(r)+c)$ is small

Good codes for additive errors [GS'09]

- Use AMD scheme to sieve list of **linear** LDC



- This corrects as many errors as LDC
 - For any string x , $\text{Dec}(\text{LDC}(x)+e) = \{x, x+e_2, \dots, x+e_L\}$
 - Since LDC is **linear**, errors e_2, \dots, e_L **independent of x**
 - AMD rejects all non-zero errors w.h.p.
- Lemma** [Guruswami-Hastad-Sudan-Zuckerman]: There exist linear LDC with rate $1-H(p)-\epsilon$ and list size $O(1/\epsilon)$.
- Consequence: additive errors codes w. rate $1-H(p)$ exist

Efficient Constructions

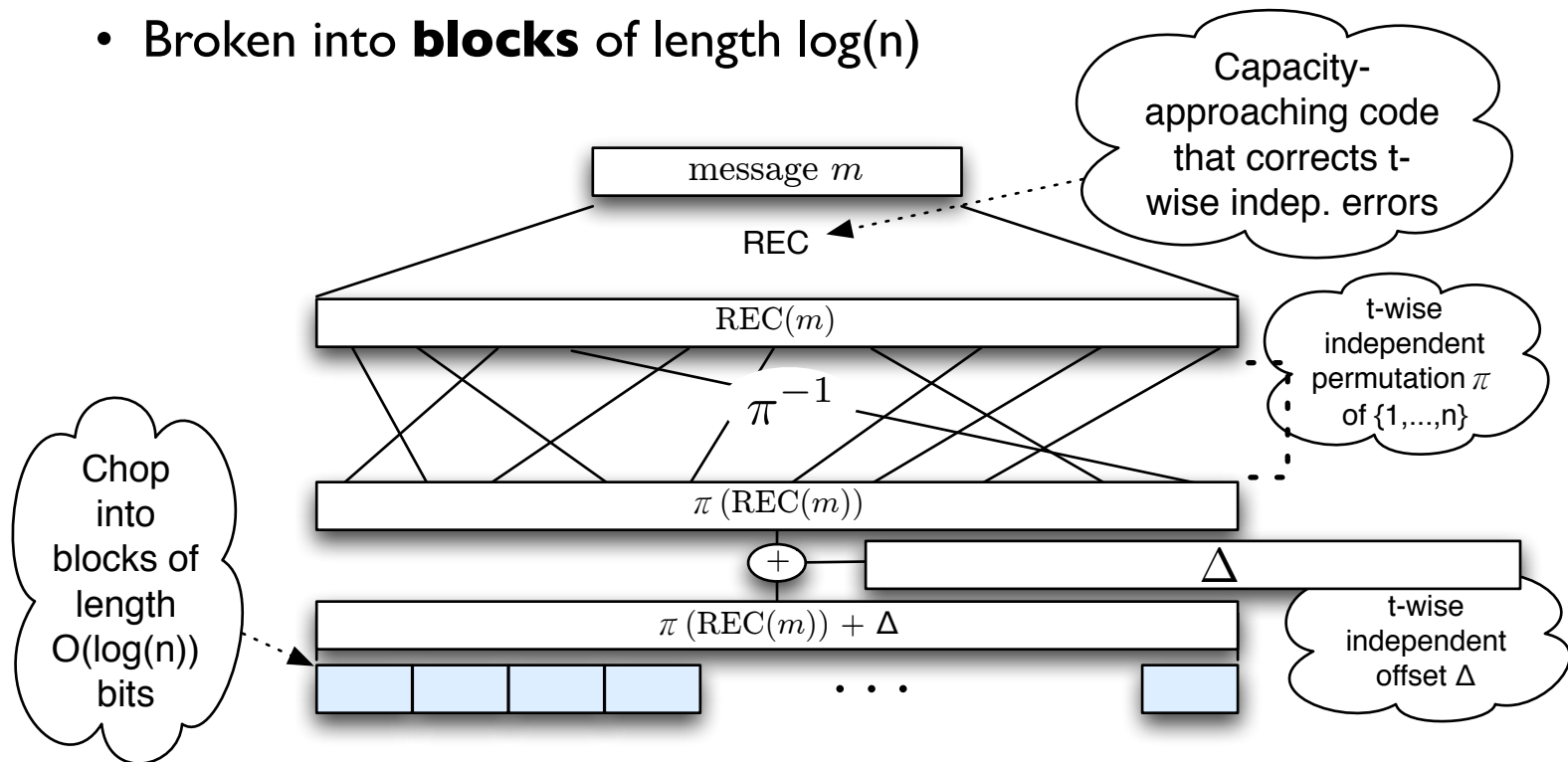
- List-decoding construction not efficient in general
 - Would like to get to capacity for all error rates p
- Idea:
 - bootstrap from “small” code (decodable by brute force) to “big code” (decodable efficiently)
 - Standard tool: concatenation [Forney]
 - Use big code over large alphabet + small code to encode symbols
 - Concatenation works poorly for worst-case errors
 - Adversary can concentrate errors in blocks (e.g. bursts)
 - Instead: use small code to share secret key for scrambling
 - Interleave small code blocks into big code blocks pseudorandomly

Control/payload construction

- Two main pieces

➤ Scrambled “payload codeword”: $\pi^{-1}(\text{REC}(m)) + \Delta$

- π is a $\log^2(n)$ -wise independent permutation,
- Δ is a $\log^2(n)$ -wise independent bit string
- Broken into **blocks** of length $\log(n)$

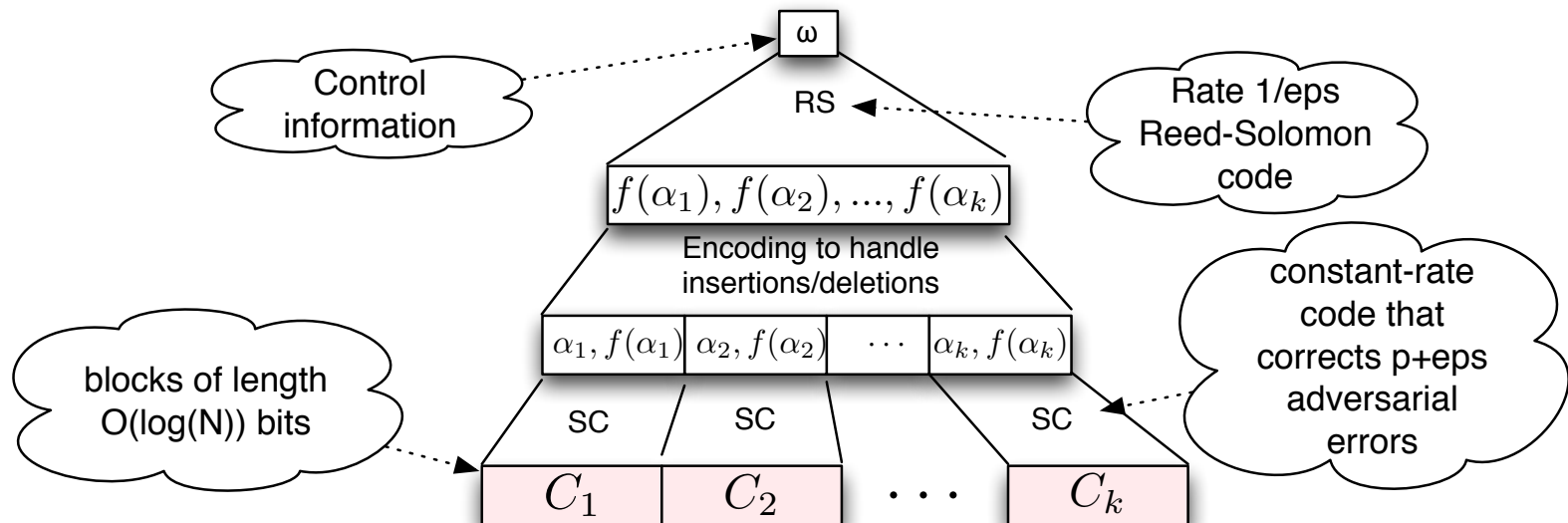


Control/payload construction

- Two main pieces
 - Scrambled “payload codeword”: $\pi^{-1}(\text{REC}(m)) + \Delta$

Control/payload construction

- Two main pieces
 - Scrambled “payload codeword”: $\pi^{-1}(\text{REC}(m)) + \Delta$
 - “Control information”: $\omega = (\pi, \Delta, T)$
 - T is a set of blocks in $\{1, \dots, n/\log(n)\}$
 - ω is encoded using Reed-Solomon-code into “control blocks”
 - Each control block encoded using small LDC+AMD code

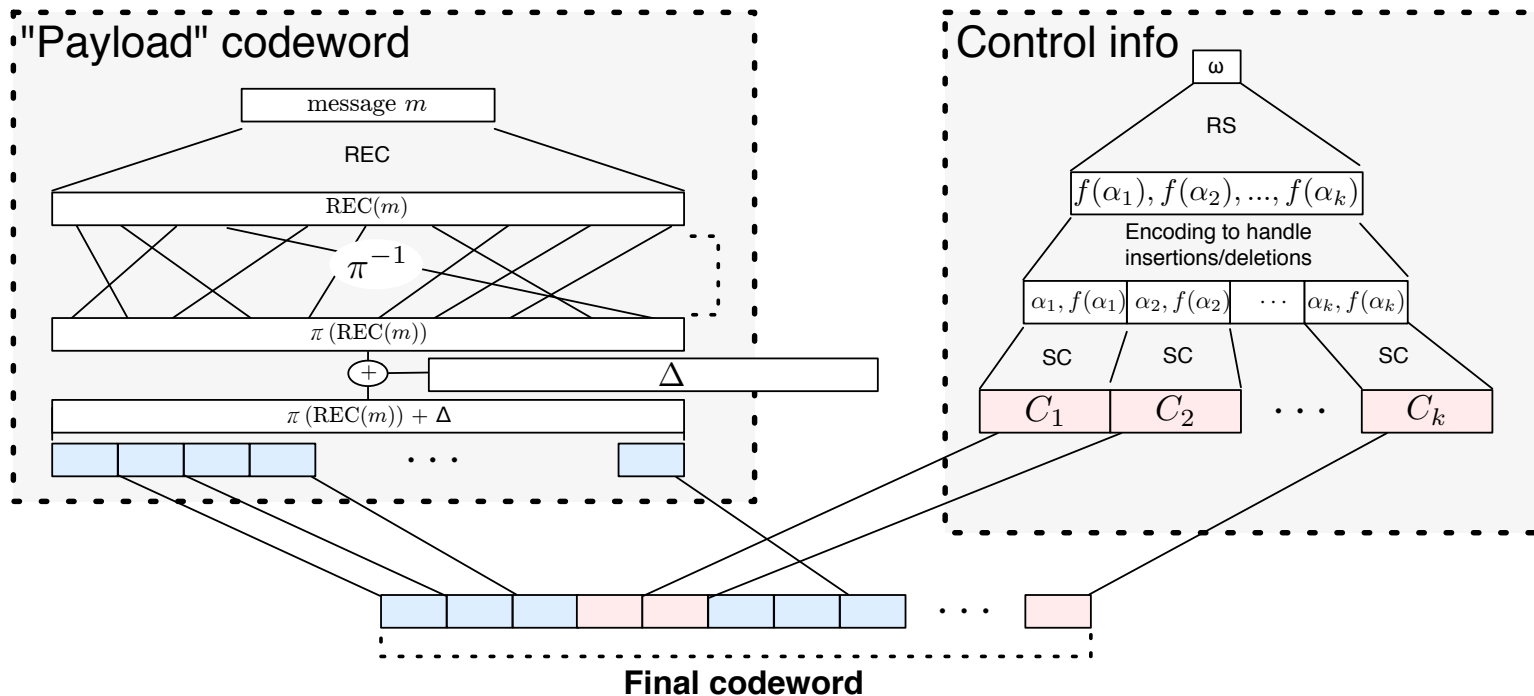


Control/payload construction

- Two main pieces
 - Scrambled “payload codeword”: $\pi^{-1}(\text{REC}(m)) + \Delta$
 - “Control information”: $\omega = (\pi, \Delta, T)$

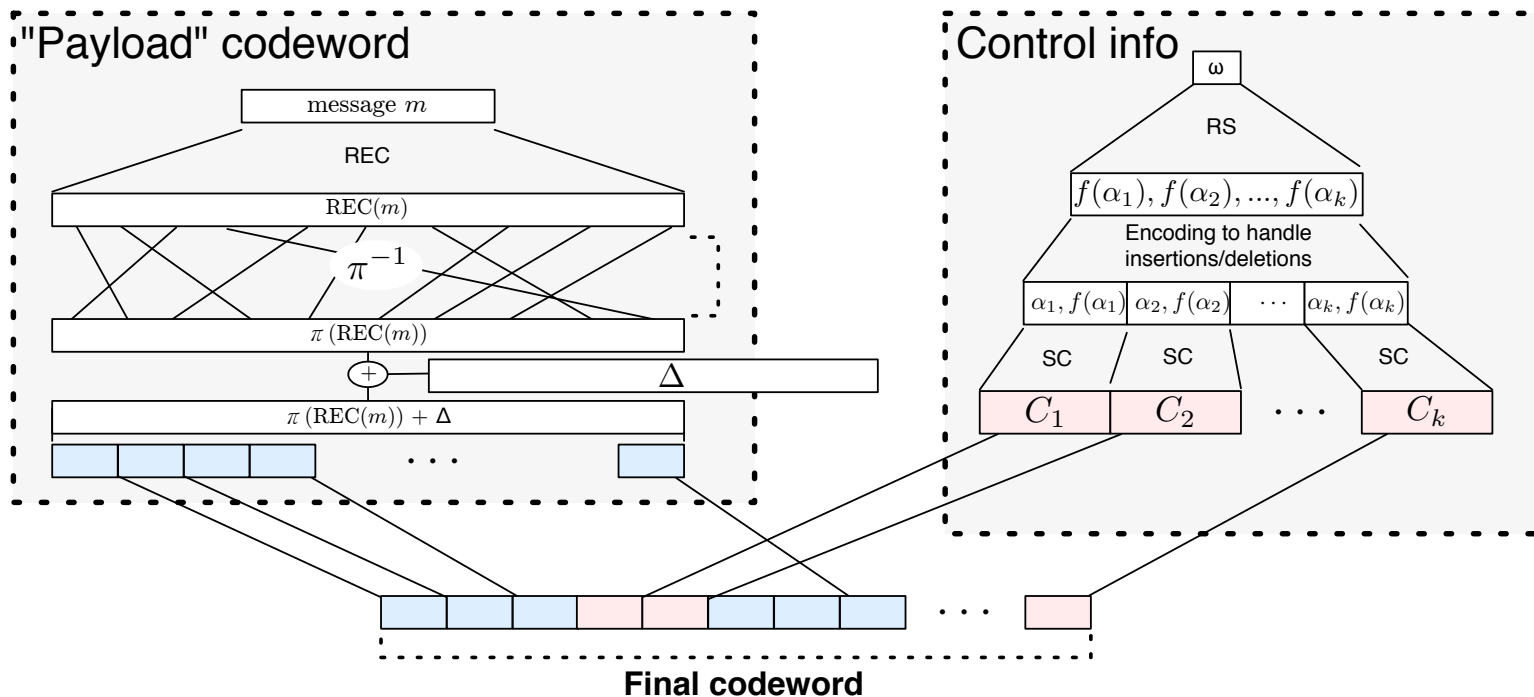
Control/payload Construction

- Two main pieces
 - Scrambled “payload codeword”: $\pi^{-1}(\text{REC}(m)) + \Delta$
 - “Control information”: $\omega = (\pi, \Delta, T)$
- Combine by interleaving according to T



Control/payload Construction

- Decoding idea
 - First decode control information, block by block
 - Given control information, unpermute scrambled code
 - Analysis delicate but follows lines of intuition



Outline

- Developing tools: shared secrets
- Computationally limited channels
- Recent results:
 - Explicit constructions for worst-case “additive” errors [GS’09]
 - Logspace channels [forthcoming]

Logspace channels

- Additive channels natural but maybe too limited
 - What if channel **sets** bits to 0/1?
 - Flips 0 to 1 more often than 1 to 0?
- Limited-memory channels
 - Errors introduced online, as codeword passes through channel
 - Channel can only remember t bits
 - Modeled as branching program with width 2^t
 - $t = O(\log n)$ captures every channel I can think of...
 - $t=n$: “online channels” [Langberg], known to be quite powerful
- Can we achieve Shannon capacity?

Conclusions

- Models for achieving maximum transmission rates in binary channels, despite uncertain or adversarial channel behavior
- Perspective, tools from cryptography / derandomization
 - Disciplinary lines are artificial
 - Crypto / information theory communities share many questions and techniques
 - But also lots of ideas take time to cross over