

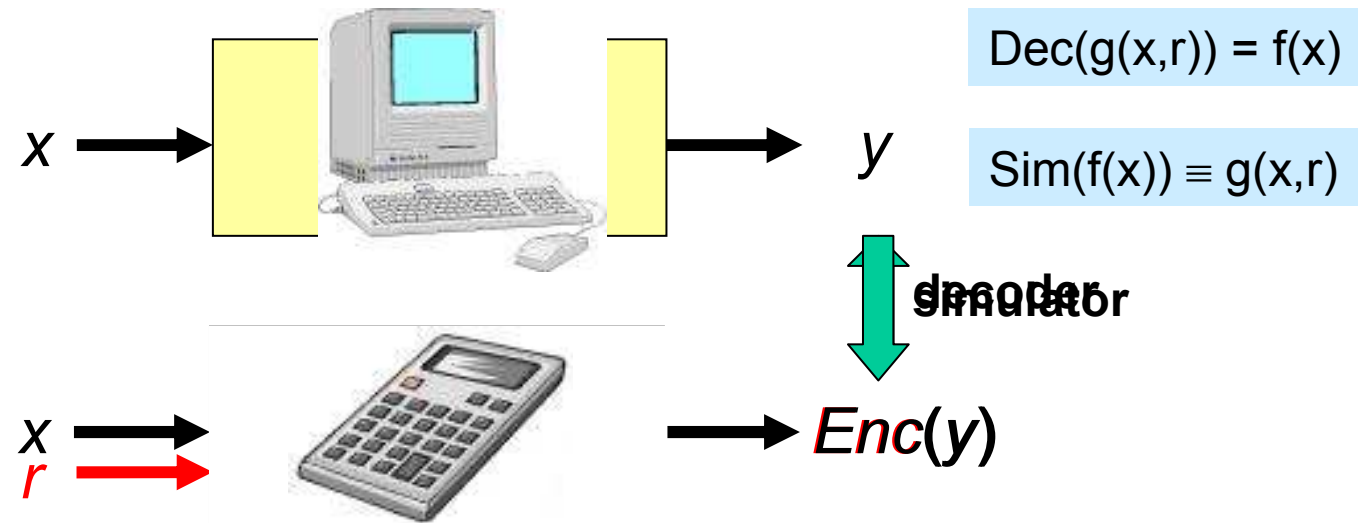
Randomization Techniques for Secure Computation and Parallel Cryptography

Eyal Kushilevitz (Technion)

Course based on joint works with
Yuval Ishai and Benny Applebaum

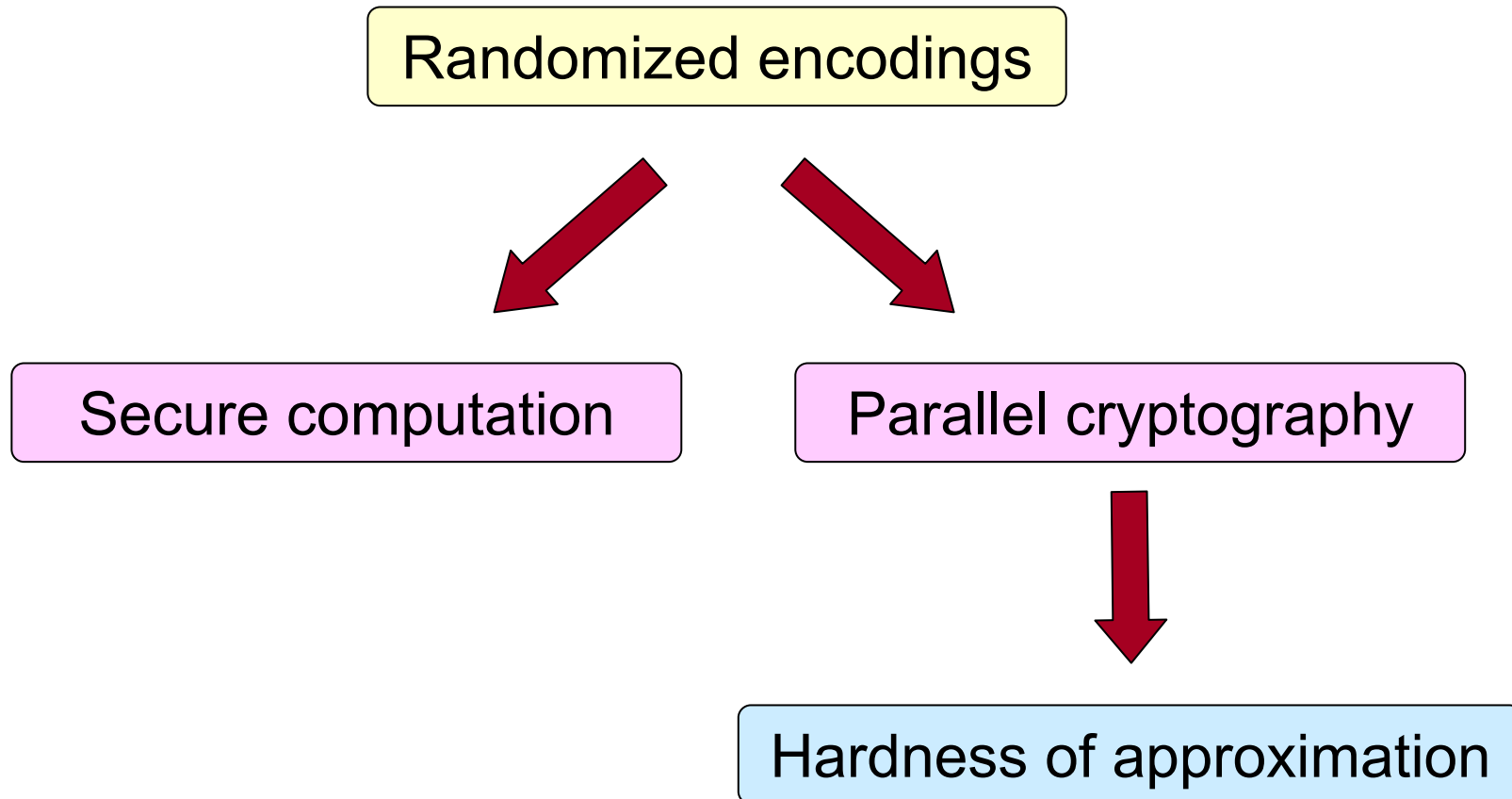
(.....,FOCS 00, ICALP 02, FOCS 04, CCC 05, RANDOM 06, CRYPTO 07)

The Basic Question

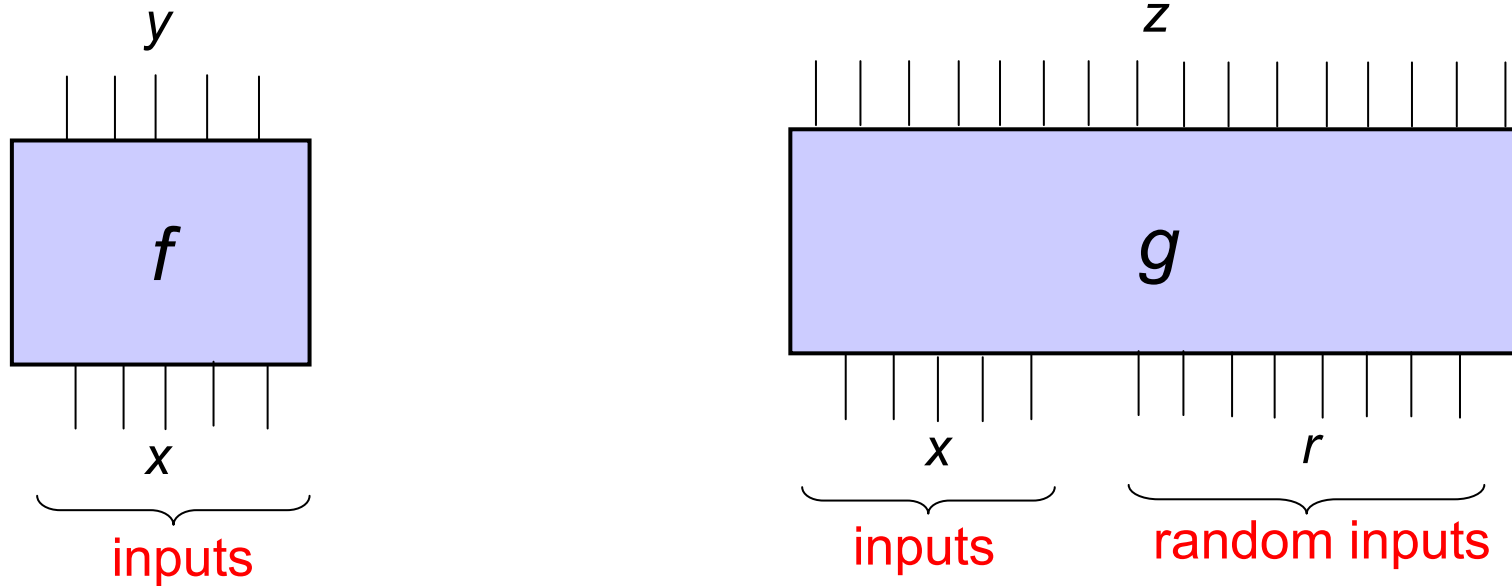


- g is a “randomized encoding” of f
 - Nontrivial relaxation of computing f
- Hope:
 - g can be “simpler” than f
(meaning of “simpler” determined by application)
 - g can be used as a substitute for f
 - g inherits security properties of f

Applications at a Glance



Randomized Encoding - Syntax

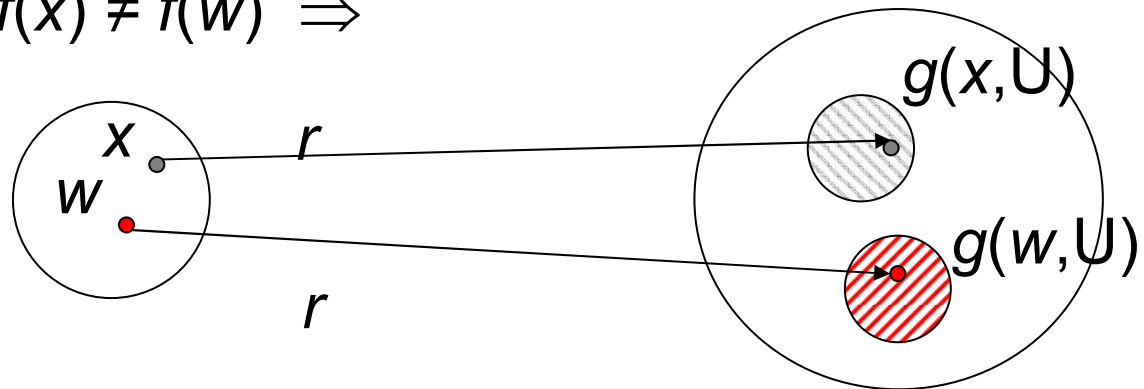


$f(x)$ is encoded by $g(x,r)$

Randomized Encoding - Semantics

- **Correctness:** $f(x)$ can be efficiently **decoded** from $g(x,r)$.

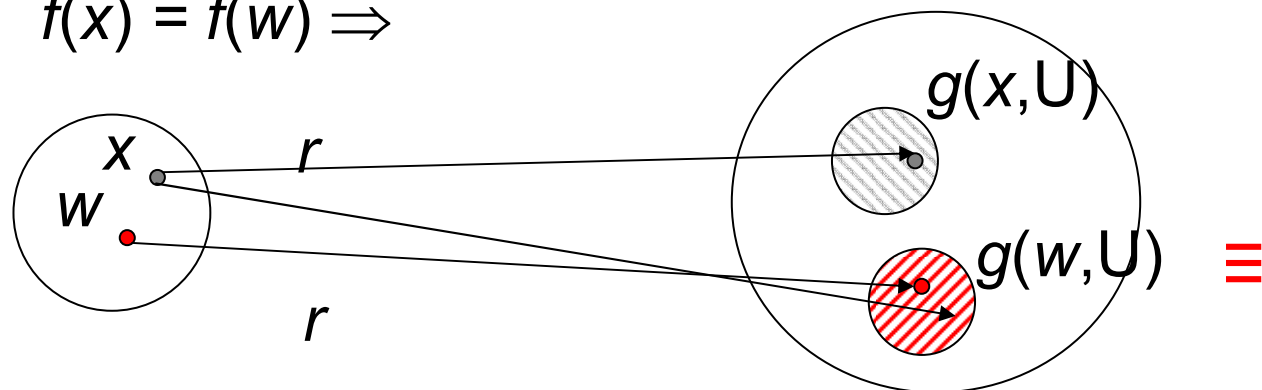
$$f(x) \neq f(w) \Rightarrow$$



- **Privacy:** \exists efficient **simulator** Sim such that $\text{Sim}(f(x)) \equiv g(x,U)$

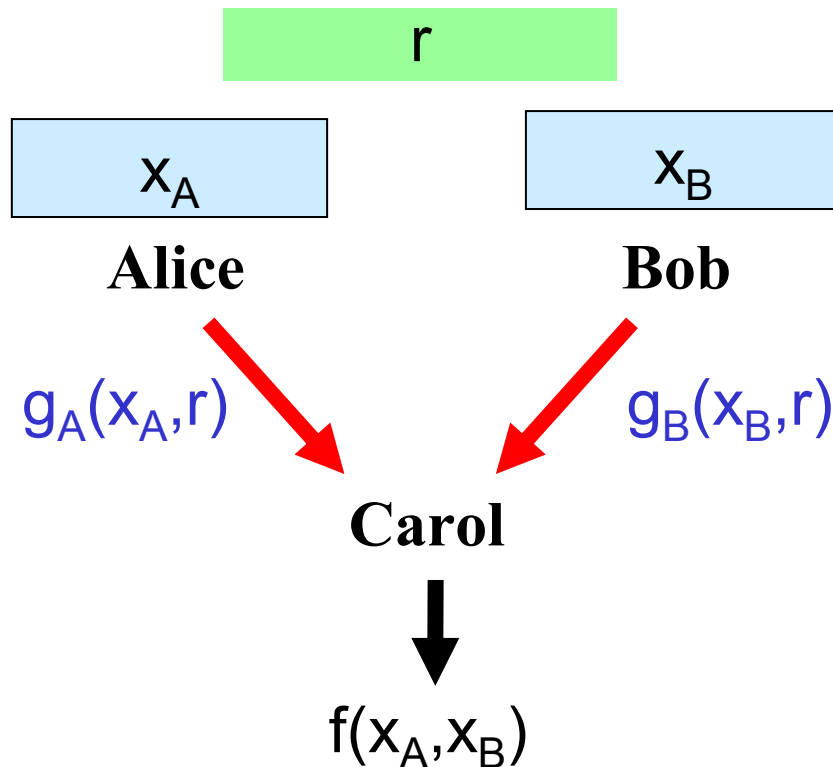
– $g(x,U)$ depends **only** on $f(x)$

$$f(x) = f(w) \Rightarrow$$



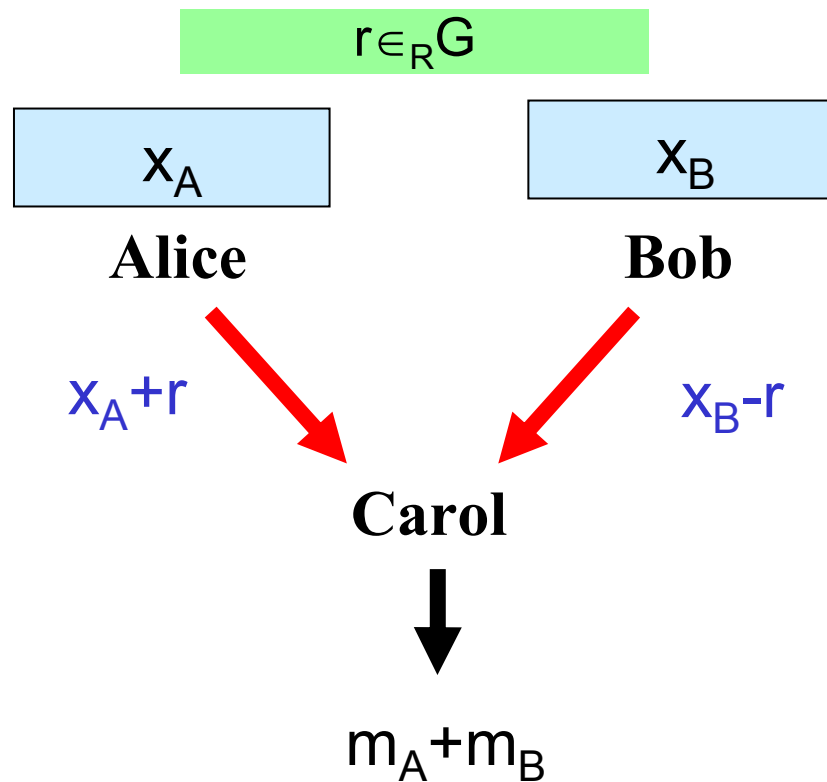
Notions of Simplicity - I

- **Application:** “minimal model for secure computation”
[Feige-Kilian-Naor 94, ...]
- **2-decomposability:** $g((x_A, x_B), r) = (g_A(x_A, r), g_B(x_B, r))$



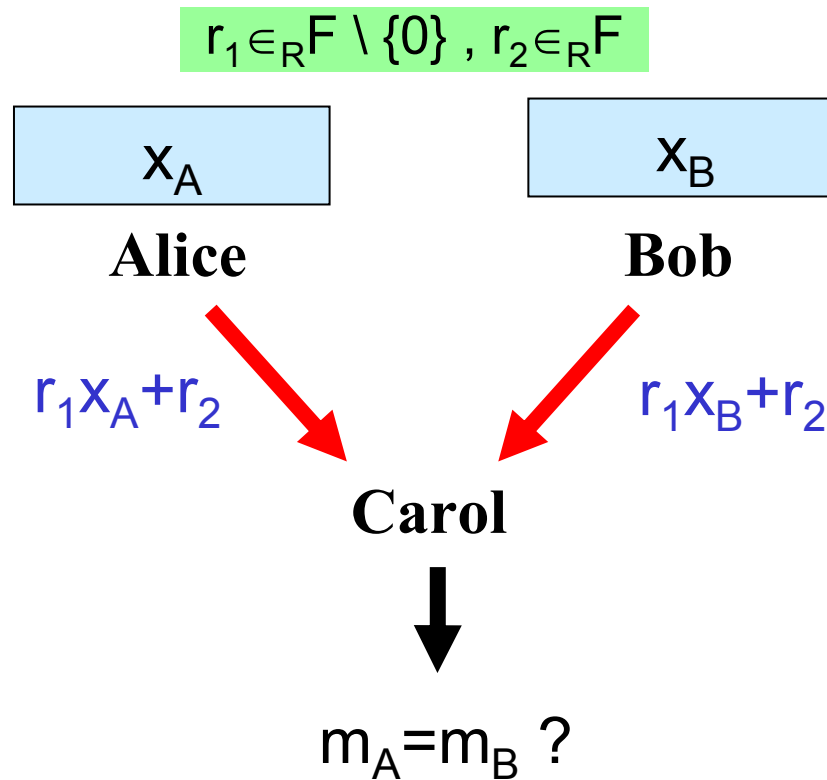
Example: sum

- $f(x_A, x_B) = x_A + x_B$ ($x_A, x_B \in$ finite group G)



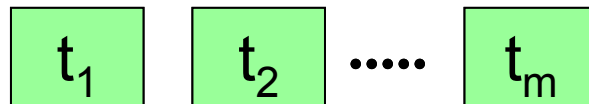
Example: equality

- $f(x_A, x_B) = \text{equality}$ ($x_A, x_B \in \text{finite field } F$)



Example: ANY function

- $f(x_A, x_B) = x_A \wedge x_B$ ($x_A, x_B \in \{0, 1\}$)
 - Reduction to equality: $x_A \rightarrow 0/1$, $x_B \rightarrow 2/1$
- **General boolean f**: write as **disjoint 2-DNF**
 - $f(x_A, x_B) = \bigvee_{(a,b):f(a,b)=1} (x_A=a \wedge x_B=b) = t_1 \vee t_2 \vee \dots \vee t_m$



000000000000 \rightarrow 0
000001000000 \rightarrow 1

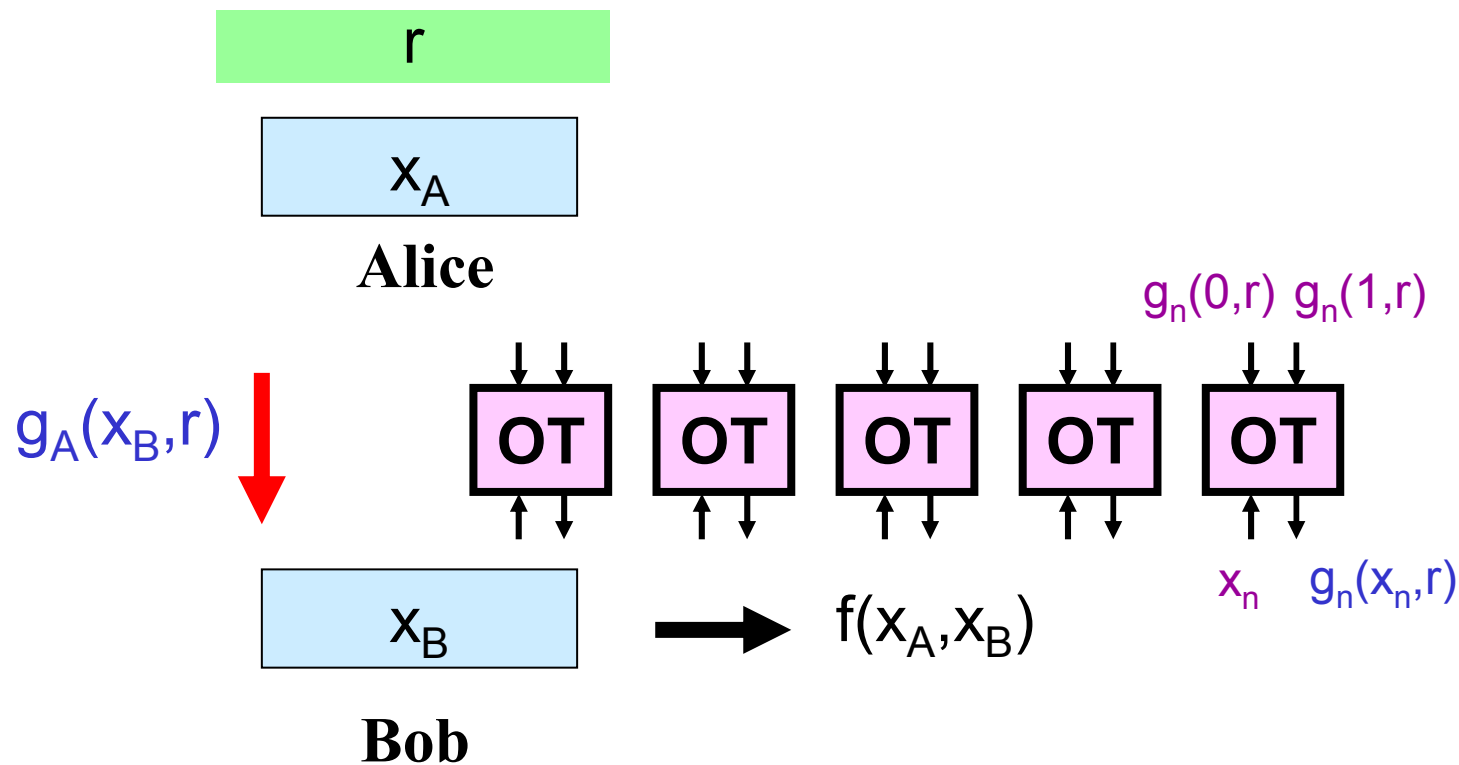
Exponential complexity

Notions of Simplicity - II

- Full decomposability:

$$g((x_1, \dots, x_n), r) = (g_1(x_1, r), \dots, g_n(x_n, r))$$

- Application: Basing SFE on OT [Kilian 88, ...]



Example: iterated group product

- Abelian case

- $f(x_1, \dots, x_n) = x_1 + x_2 + \dots + x_n$

- $g(x, (r_1, \dots, r_{n-1})) =$

$$x_1 + r_1 \quad x_2 + r_2 \quad \dots \quad x_{n-1} + r_{n-1} \quad x_n - r_1 - \dots - r_{n-1}$$

- General case [Kilian 88]

- $f(x_1, \dots, x_n) = x_1 x_2 \dots x_n$

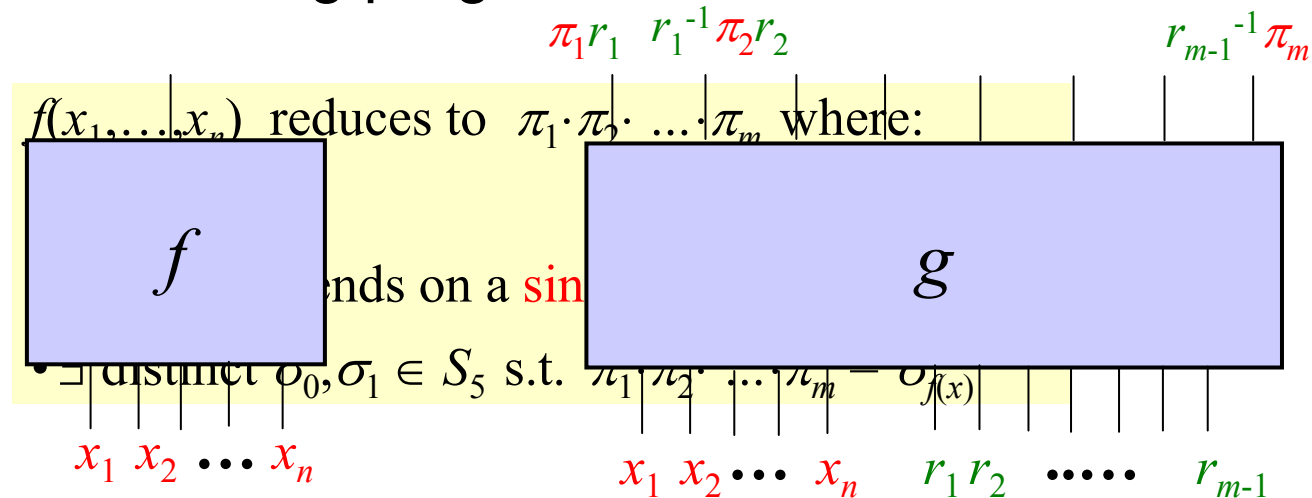
- $g(x, (r_1, \dots, r_{n-1})) =$

$$x_1 r_1 \quad r_1^{-1} x_2 r_2 \quad r_2^{-1} x_3 r_3 \quad \dots \quad r_{n-2}^{-1} x_{n-1} r_{n-1} \quad r_{n-1}^{-1} x_n$$

Example: iterated group product

Thm [Barrington 86]

Every boolean $f \in NC^1$ can be computed by a poly-length, width-5 branching program.



Encoding iterated group product

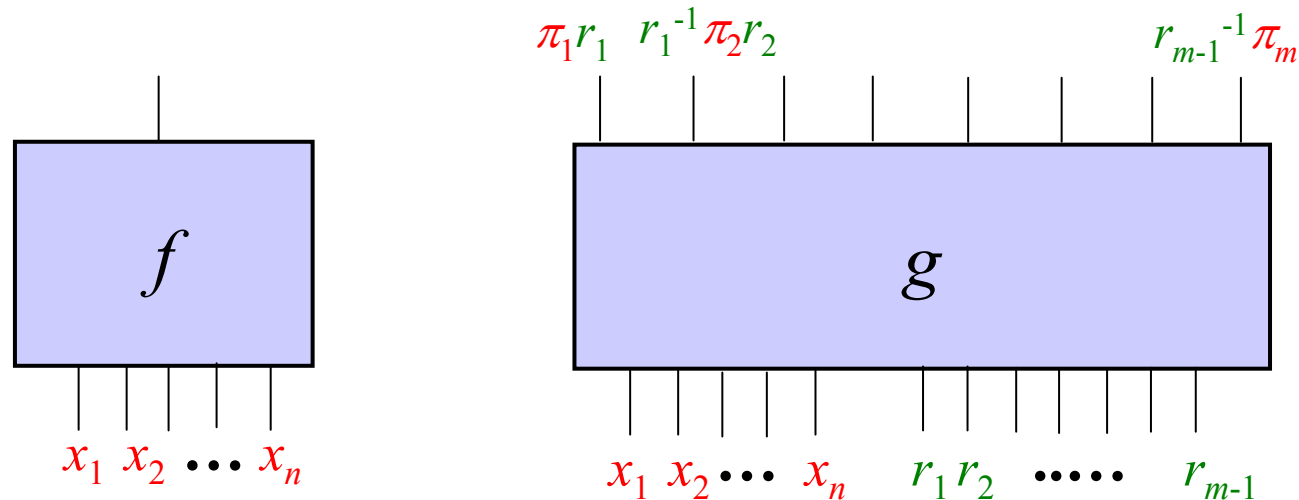
- Every output bit of g depends on just a single bit of x
- Efficient fully decomposable encoding for every $f \in NC^1$

Notions of Simplicity - III

- **Low degree:** $g(x,r)$ = vector of degree- d poly in x,r over F
 - aka “Randomizing Polynomials” [IK00,...]
 - Application: round-efficient MPC
- Motivating observation:
Low-degree functions are easy to distribute!
 - Round complexity of MPC protocols [BGW88, CCD88, CDM00, ...]
 - Semi-honest model
 - $t < n/d \rightarrow 2$ rounds
 - $t < n/2 \rightarrow$ multiplicative depth + 1 = $\lceil \log d \rceil + 1$ rounds
 - Malicious model
 - Optimal $t \rightarrow O(\log d)$ rounds

Examples

- What's wrong with previous examples?
 - Great degree in x ($\deg_x=1$), bad degree in r



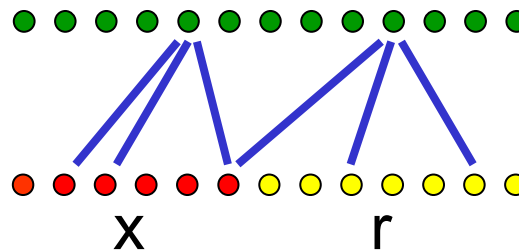
- Coming up:

- Degree-3 encoding for every f
- Efficient in size of branching program

$\in_R \mathbf{S}_5$

Notions of Simplicity - IV

- Small locality:



- Application: parallel cryptography!

[AIK04,AIK05,AIK07...]

- Coming up: encodings with locality 4
 - degree 3, fully decomposable
 - efficient in size of branching program

Parallel Cryptography

How low can we get?

poly-time

NC

log-space

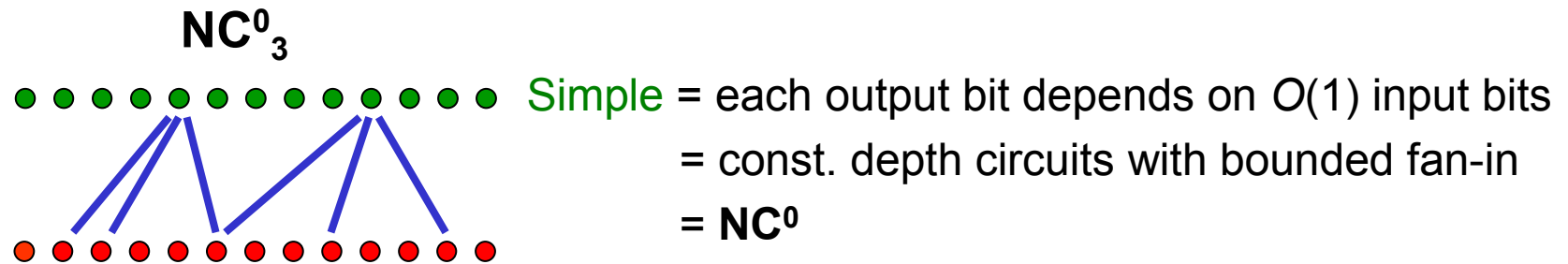
NC¹

AC⁰

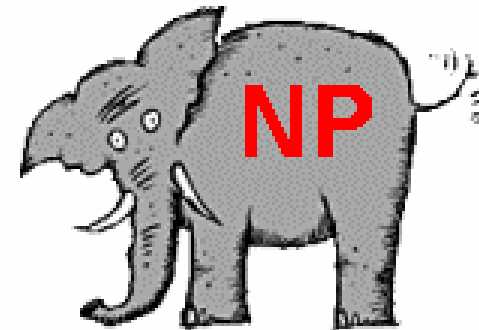
NC⁰

“Simplicity” of Cryptographic Primitives

- Can cryptographic primitives be computed by **very simple** functions?



- Currently the smallest creature in the complexity zoo



Cryptography in NC^0 ?

- Longstanding open question

Håstad 87

Impagliazzo Naor 89

Goldreich 00

Cryan Miltersen 01

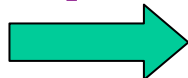
Krause Lucks 01

Mossel Shpilka Trevisan 03

- Real-life motivation: super-fast cryptographic hardware
- Tempting conjecture:

crypto hardness

[CM]: Yes

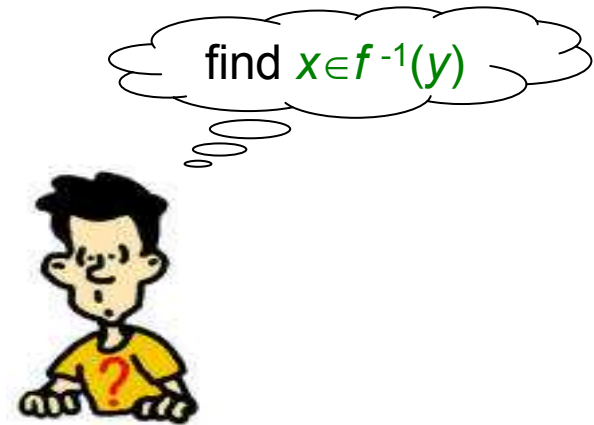
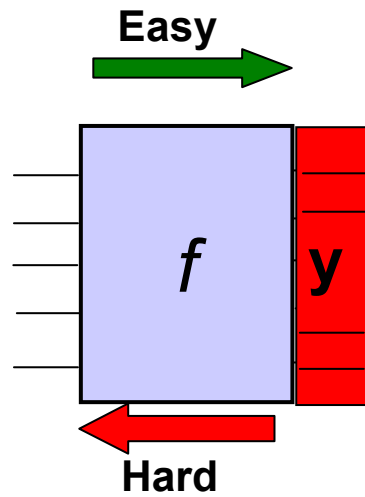


[G]: No

“complex” function

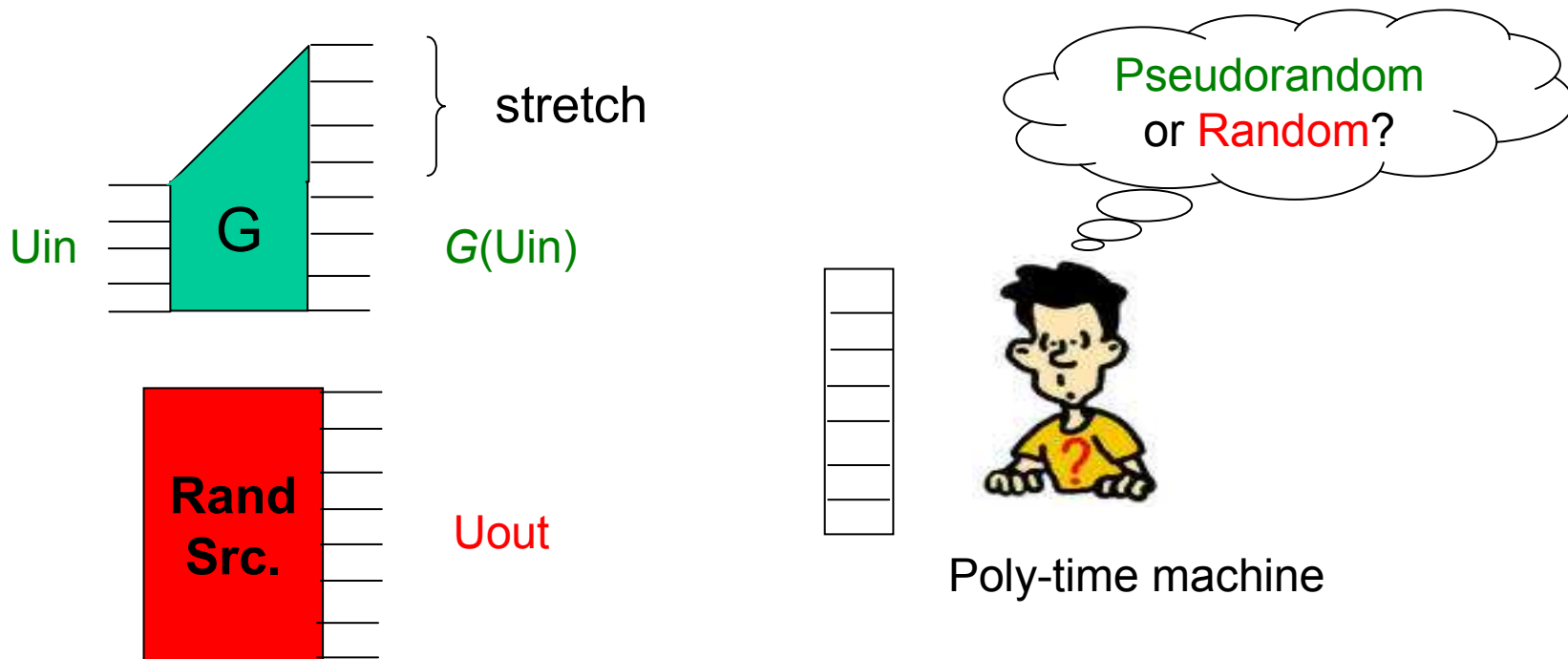
Basic Primitives: One-way Function (OWF)

OWF



Poly-time machine

Basic Primitives: Pseudorandom Generator (PRG)

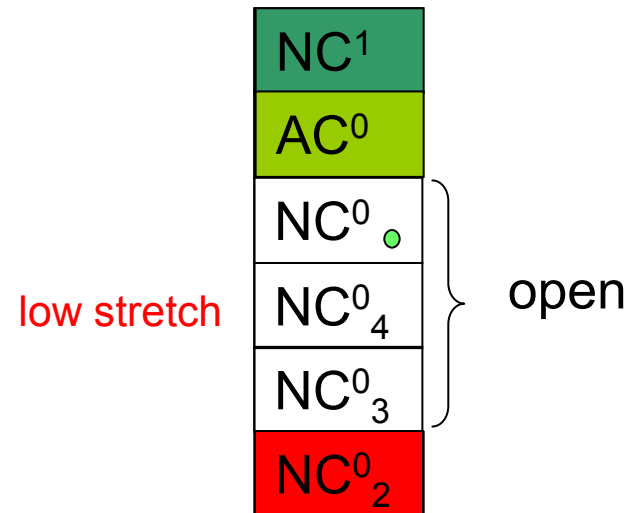
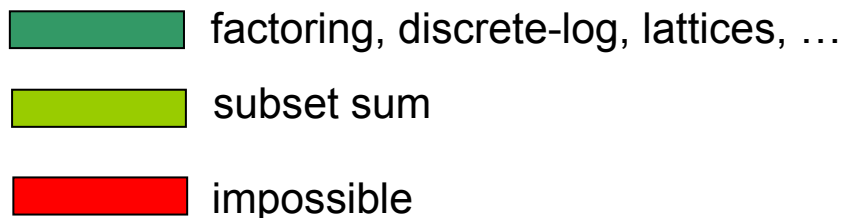


Def. PRG is minimal if stretch=1

Previous Work

- **Negative results**

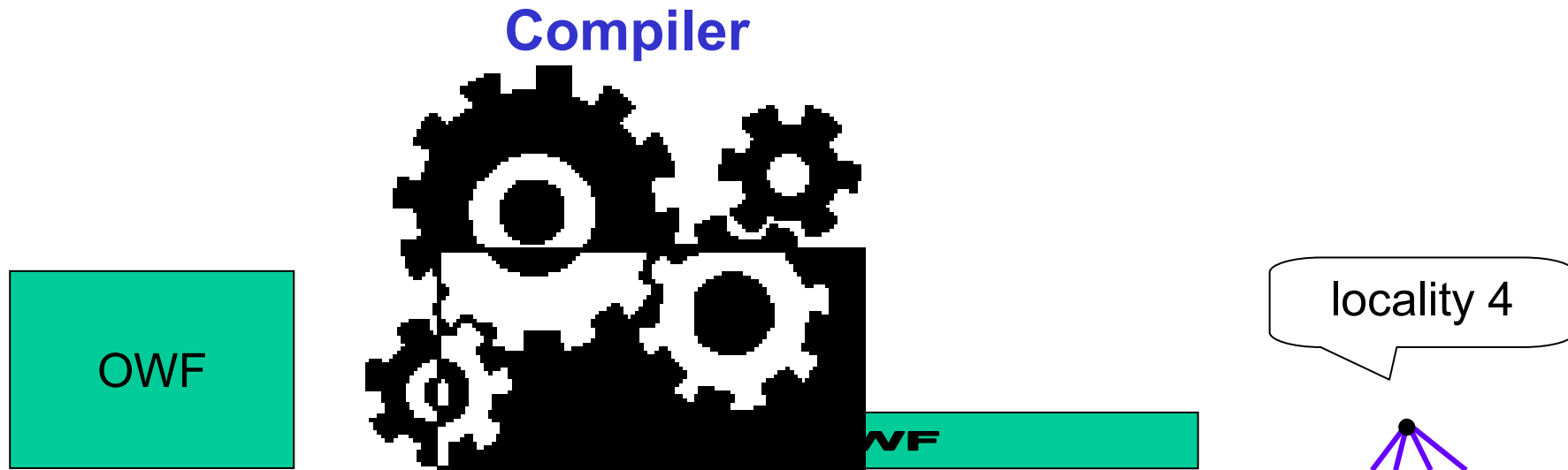
- No OWF/PRG in NC^0 [Goldreich 00, Crepeau-Matrises 01]
- No PRG in NC^1 from stretching $[NC^0, NC^0]$ [CWT1, MosselShpilkaTrevisan03]
- PRG (sub-lin stretch) in AC^0 from subset sum [Impagliazzo Naor 89]
- Permutation in NC^0 which is P-complete to invert [Håstad 87]
- Function in NC^0 which is NP-complete to invert [Agrawal Allender Rudich98]
- Heuristic construction of OWF/PRG in NC^0 [Goldreich 00 MST 03]



PRG / OWF

Our Approach

Compile primitives in a “relatively high” complexity class into ones in NC^0 .



Sufficient Assumptions for Crypto in NC^0

Caveats:

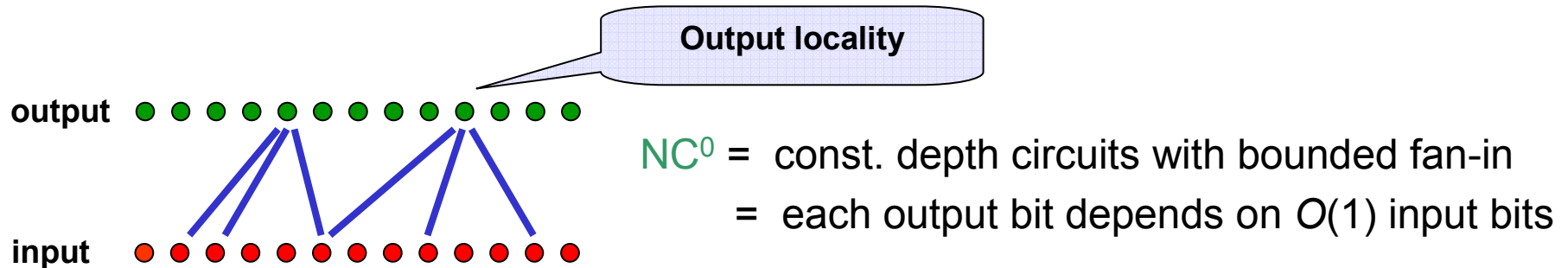
- We get PRG with sub-linear stretch
 - decryption verification not in NC^1
 - In NC^1 possible to decrypt/verify
 - ... commit in NC^0 with NC^1 [AMK05]
- OWF
PRG
Hash
Sym-Enc
PK-Enc
Signature
Commit
NIZK
OWF
PRG
Hash
Sym-Enc
PK-Enc
Signature
Commit
NIZK
- $\in NC^0$ [AND]

■ factoring, discrete-log/DDH, lattices, ...

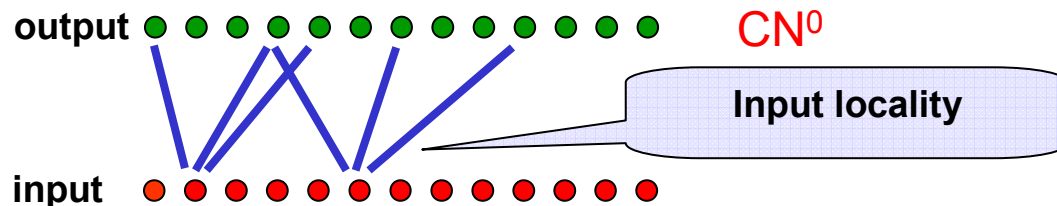
P									factoring
NC^1									factoring
NC^0_4									factoring
	OWF	PRG	Hash	Sym-Enc	PK-Enc	NI-Com	Sign	NIZK	

Cryptography with Constant Input Locality

Till now we considered only NC^0 functions...



Q: Can cryptographic primitives be realized by functions in which each input bit affects a constant number of output bits?



Outline

1. (Long) Introduction
2. Randomized Polynomials (w/applications to round-efficient MPC)
3. Randomized Encodings (w/applications to NC^0 Cryptography)
4. Constant Input Locality
5. Computational Randomized Encodings (w/applications)
6. NC^0 Linear Stretch PRG (w/applications)