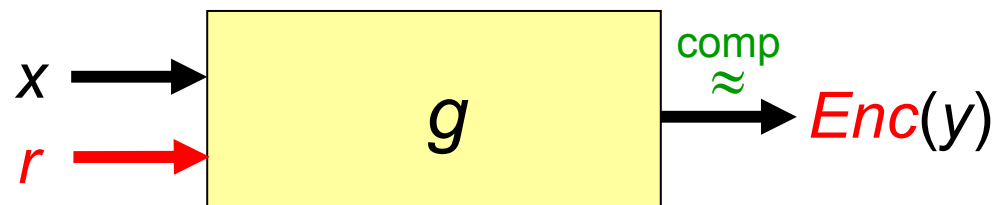


Outline

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)

Computationally Private Encodings

- **Known:** $f \in \text{NC}^1, \oplus\text{L}$ \rightarrow encoding in NC^0
- **Goal:** $f \in \text{P}$ \rightarrow encoding in NC^0
- **Idea:** relax encoding requirement



- Respects security of most primitives
- **Thm:** $f \in \text{P} \rightarrow$ computational encoding in NC_4^0 assuming “easy PRG” (min-PRG $\in \oplus\text{L}$)
 - ✓ “Easy PRG” can be based on **factoring**, **discrete-log**, **lattices**

App 1: Relaxed Assumptions for Crypto in NC^0

- Using **perfect** encoding:

OWF

Assuming “easy PRG”

PRG

Hash

Sym-Enc

PK-Enc

Signature

Commit

NIZK

~~exist~~



OWF

OWP

PRG

Hash

Sym-Enc

PK-Enc

Signature

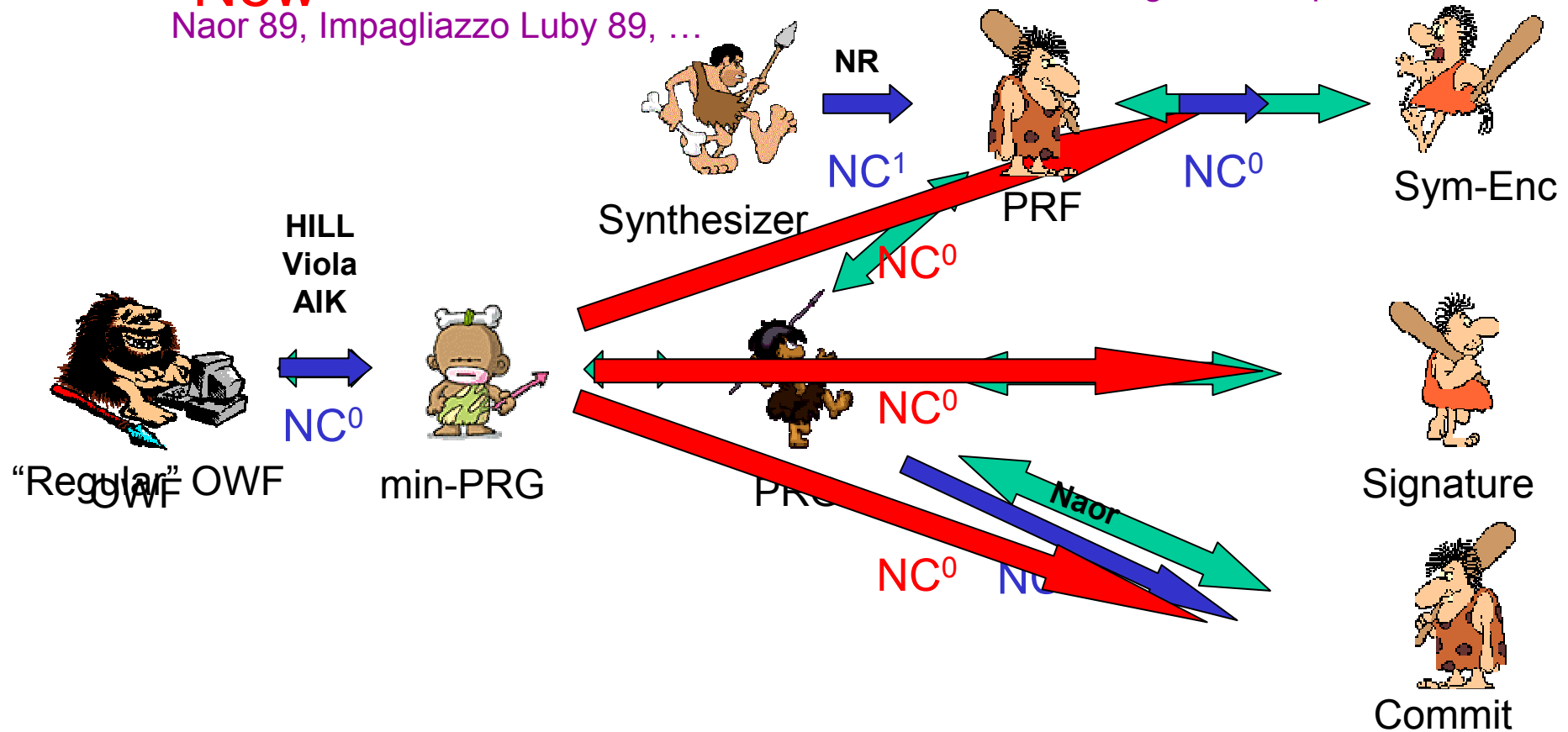
Commit

NIZK

$\in NC^0$

App 2: Parallel Reductions Between Primitives

- **What are equivalent under poly-time reductions**
- **Much less is known...**
- **New** Goldwasser Micali Rivest 84, Bellare Micali 88, Naor Yung 89, Rompel 90, Naor 89, Impagliazzo Luby 89, ...



App 3: Secure Multiparty Computation

In case you don't insist on unconditional security...

- Securely evaluating an arbitrary function f *efficiently* reduces to securely evaluating deg-3 polynomials

... assuming an “easy PRG”

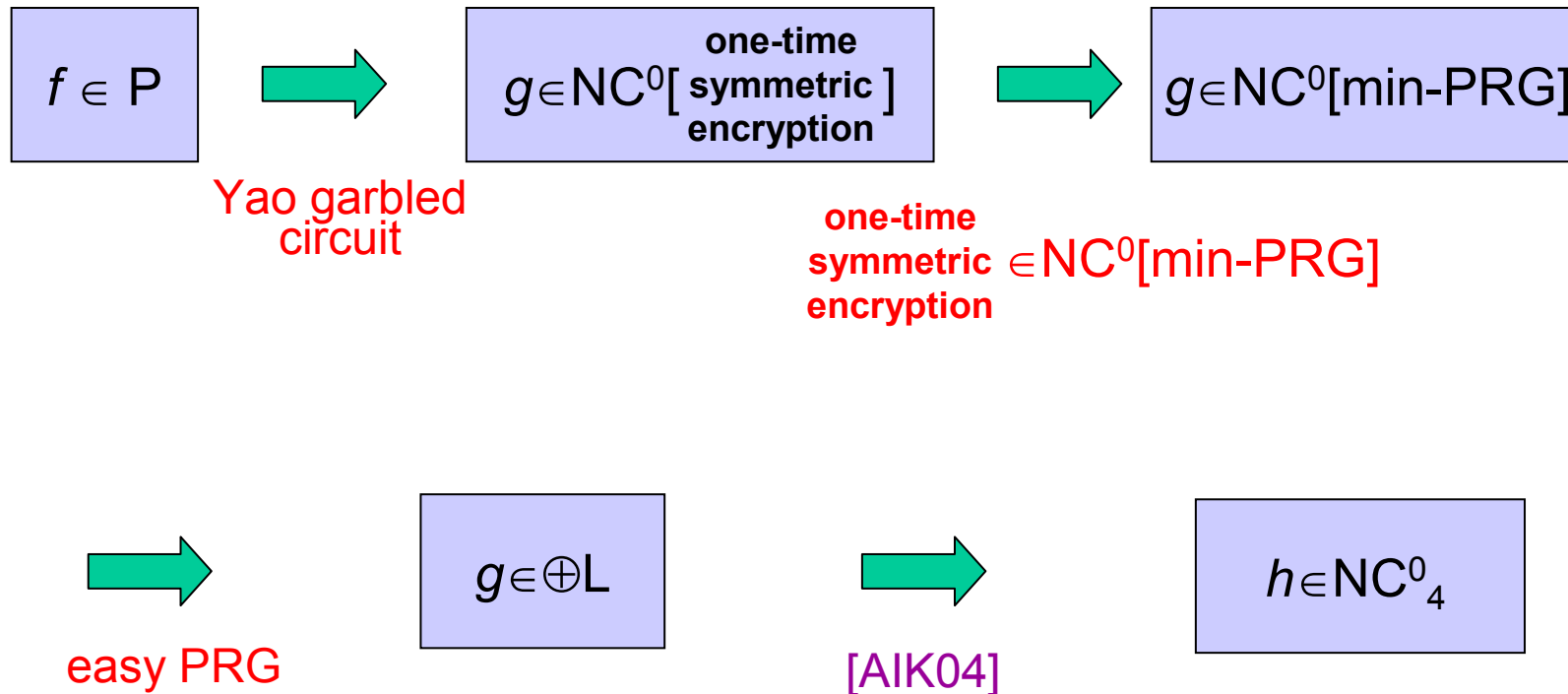
- In particular:

Basic MPC protocols (e.g., BGW88, CDM00) imply constant-round *computationally* secure MPC for every f .

Known assuming *any* PRG [BMR90,DI05]; however, current approach is simpler and can be made more efficient [DI06].

Proof Outline

Thm. “easy PRG” \Rightarrow encoding in NC^0 for all $f \in P$



Step 1: Min-PRG $\xrightarrow{NC^0}$ OTSE

Sol: Use naive construction in parallel to increase key size

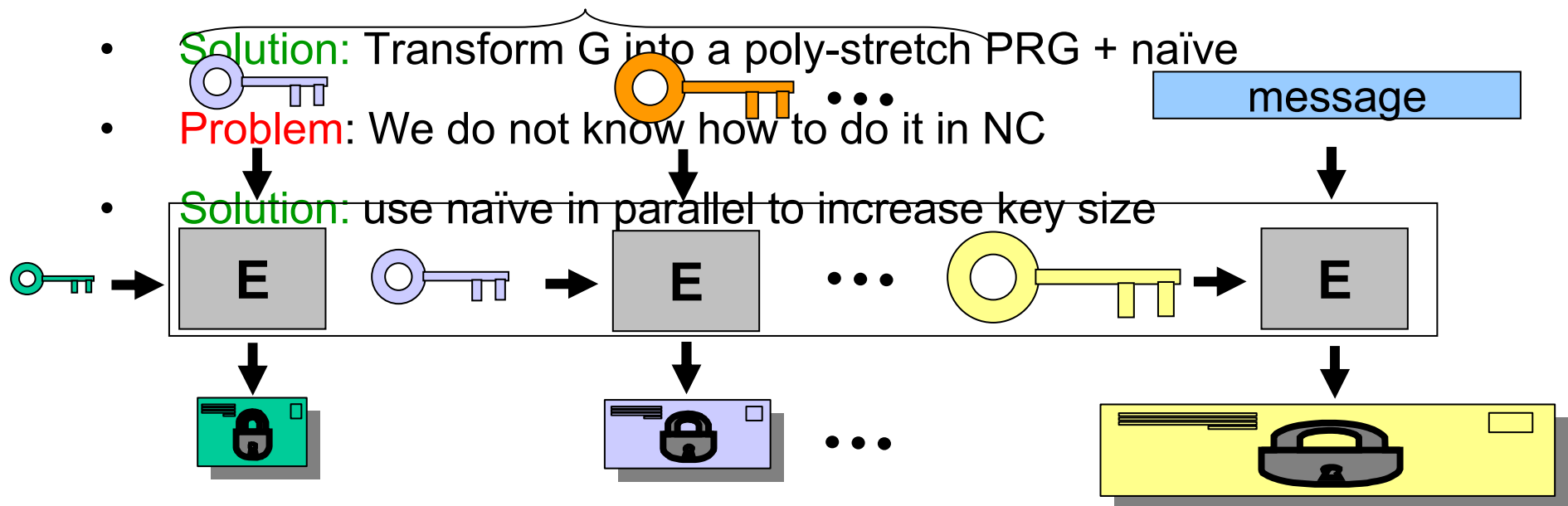
- **Naïve construction:** $E_{key}(m) = G(key) \oplus m$, $D_{key}(c) = G(key) \oplus c$

- **Problem:** Cannot encrypt long messages, $|m| \leq |key| + 1$

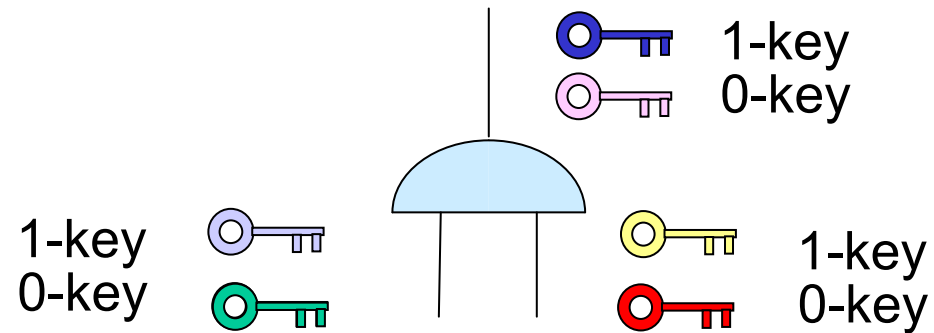
- **Solution:** Transform G into a poly-stretch PRG + naïve

- **Problem:** We do not know how to do it in NC

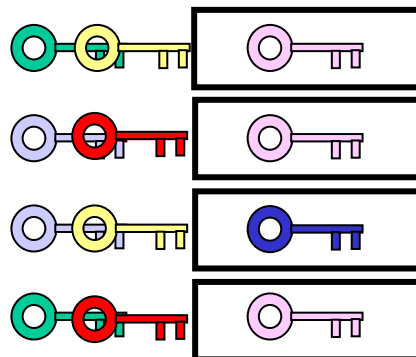
- **Solution:** use naïve in parallel to increase key size



Step 2: Garbled Circuit Construction



- Pair of random colored keys for each wire
- For each input wire, key corresponding to its value is revealed
- Color semantics of output wires are revealed
- Garbled gates:



Step 2: Garbled Circuit Construction

Implementing locks

- (one-time) symmetric encryption
 - computational privacy, works for any circuit
- one-time pads
 - information-theoretic privacy, efficient only for log-depth circuits

- Garbled gates.

