Program Obfuscation
and the Quest for Cryptography's Holy Grail

Vinod Vaikuntanathan

MIT CSAIL

Program Obfuscation

n. the action of making a program unintelligible, while preserving its input/output behavior.

Obfuscation

n. the action of making something obscure, unclear, or unintelligible.

Program 1

```c
#include                                          <math.h>
#include                                        <sys/time.h>
#include                                        <X11/Xlib.h>
#include                                       <X11/keysym.h>
                                               double L ,o ,P
                                              ,_=dt,T,Z,D=1,d,
                                              s[999],E,h= 8,I,
                                              J,K,w[999],M,m,O
                                             ,n[999],j=33e-3,i=
                                             1E3,r,t, u,v ,W,S=
                                             74.5,l=221,X=7.26,
                                             a,B,A=32.2,c, F,H;
                                             int N,q, C, y,p,U;
                                            Window z; char f[52]
                                         ; GC k; main(){ Display*e=
 XOpenDisplay( 0); z=RootWindow(e,0); for (XSetForeground(e,k=XCreateGC (e,z,0,0),BlackPixel(e,0))
; scanf("%lf%lf%lf",y +n,w+y, y+s)+1; y ++); XSelectInput(e,z= XCreateSimpleWindow(e,z,0,0,400,400,
0,0,WhitePixel(e,0) ),KeyPressMask); for(XMapWindow(e,z); ; T=sin(O)){ struct timeval G={ 0,dt*1e6}
; K= cos(j); N=1e4; M+= H*_; Z=D*K; F+=_*P; r=E*K; W=cos( O); m=K*W; H=K*T; O+=D*_*F/ K+d/K*E*_; B=
sin(j); a=B*T*D-E*W; XClearWindow(e,z); t=T*E+ D*B*W; j+=d*_*D-_*F*E; P=W*E*B-T*D; for (o+=(I=D*W+E
*T*B,E*d/K *B+v+B/K*F*D)*_; p<y; ){ T=p[s]+i; E=c-p[w]; D=n[p]-L; K=D*m-B*T-H*E; if(p [n]+w[ p]+p[s
]== 0|K <fabs(W=T*r-I*E +D*P) |fabs(D=t *D+Z *T-a *E)> K)N=1e4; else{ q=W/K *4E2+2e2; C= 2E2+4e2/ K
 *D; N-1E4&& XDrawLine(e ,z,k,N ,U,q,C); N=q; U=C; } ++p; } L+=_* (X*t +P*M+m*l); T=X*X+ l*l+M *M;
  XDrawString(e,z,k ,20,380,f,17); D=v/l*15; i+=(B *l-M*r -X*Z)*_; for(; XPending(e); u *=CS!=N){
                                        XEvent z; XNextEvent(e ,&z);
                                            ++*((N=XLookupKeysym
                                              (&z.xkey,0))-IT?
                                              N-LT? UP-N?& E:&
                                              J:& u: &h); --*(
                                              DN -N? N-DT ?N==
                                              RT?&u: & W:&h:&J
                                               ); } m=15*F/l;
                                               c+=(I=M/ l,l*H
                                               +I*M+a*X)*_; H
                                               =A*r+v*X-F*l+(
                                               E=.1+X*4.9/l,t
                                               =T*m/32-I*T/24
                                                )/S; K=F*M+(
                                                h* 1e4/l-(T+
                                                E*5*T*E)/3e2
                                                )/S-X*d-B*A;
                                                a=2.63 /l*d;
                                                X+=( d*l-T/S
                                                 *(.19*E +a
                                                 *.64+J/1e3
                                                 )-M* v +A*
                                                 Z)*_; l +=
                                                 K *_; W=d;
                                                 sprintf(f,
                                                 "%5d  %3d"
                                                 "%7d",p =l
                                                /1.7,(C=9E3+
                              O*57.3)%0550,(int)i); d+=T*(.45-14/l*
                             X-a*130-J* .14)*_/125e2+F*_*v; P=(T*(47
                             *I-m* 52+E*94 *D-t*.38+u*.21*E) /1e2+W*
                             179*v)/2312; select(p=0,0,0,0,&G); v-=(
                              W*F-T*(.63*m-I*.086+m*E*19-D*25-.11*u
                               )/107e2)*_; D=cos(o); E=sin(o); } }
```

Program 2

```c
#define _ F-->00 || F-OO--;
long F=00,OO=00;
main(){F_OO();printf("%1.3f\n", 4.*-F/OO/OO);}F_OO()
{
            _-_-_-_
       _-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_-_
       _-_-_-_-_-_-_-_-_
            _-_-_-_
}
```

Courtesy: IOCCC/Omer P.

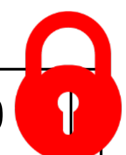
Answer: Run me!

Program Obfuscation

PROGRAMS w/ SECRETS:

- Cryptographic keys
- Licensing Info
- Backdoors
- The Algorithm Itself

Example: E-mail delegation



def	138805012AA98B7920FC103850
Se	89012408A292E00FF001659009
m	01659AA1606B692650F3893EE3
if	9030957BE927A6789C10846DD
re	10AA92DEADBEEF09179578134

Program Obfuscation in Crypto

"CRYPTO-COMPLETE":

Nearly all crypto is an easy corollary of program obfuscation.

Public Key Encryption (from Secret Key Encryption)

[Diffie-Hellman'76]

> Essentially what is required is a one-way compiler: one which takes an easily understood program written in a high level language and translates it into an incomprehensible program in some machine language. The compiler is one-

Public Encryption Algorithm =

Enc(SK,●)

Public Encryption Algorithm?

Enc(SK,●)

Dec(SK,●)

Dec(SK,●)

Secret-key Encryption

Public-key Encryption

Program Obfuscation in Crypto

"CRYPTO-COMPLETE":

Nearly all crypto is an easy corollary of program obfuscation.

Fully Homomorphic Encryption

[Rivest-Adleman-Dertouzos'78, Gentry'09, Brakerski-V'11]

On Input ciphertexts c_1, c_2 and OP:

m_1 = Dec(SK, c_1); m_2 = Dec(SK, c_2);

m_3 = m_1 OP m_2;

Return Enc(SK, m_3);

"CRYPTO-COMPLETE":

Nearly all crypto is an easy corollary of program obfuscation.

PPAD Hardness

Two-round MPC

One-way Functions

Public-key Encryption

"Pure" FHE

Time-lock Puzzles

Succinct RE

Deniable Encryption

Functional Encryption

Constrained PRFs

Non-interactive Key Exchange

Correlation-intractable fns

Traitor Tracing

NIWI

Software Watermarking

Compact Token-based Obfuscation

"Program Obfuscation"

TUTORIAL OUTLINE

Part 1. DEFINITIONS of program obfuscation

a. Virtual Black-Box OBF

b. Indistinguishability OBF (IO)

Part 2. APPLICATIONS of IO

a. Crypto Applications

b. A Complexity Application

c. Bootstrapping Theorems

Part 3. CONSTRUCTIONS of IO from simpler objects

Theorem: If 3-linear maps exist and local PRGs exist, so does IO.

Part 4. DE-IO-IZATION

Remove the need for IO in applications.

e.g., Traitor Tracing (on Wed)

Defining Program Obfuscation (Take 1)

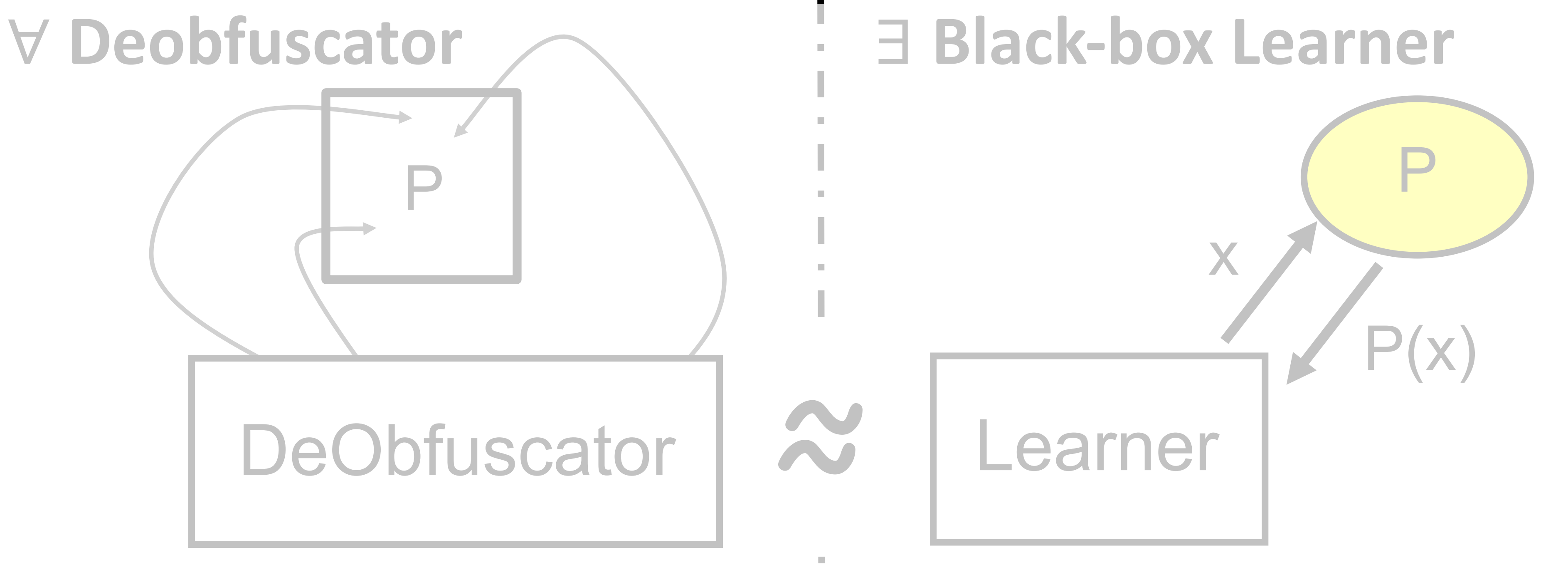
Virtual Black-Box (VBB) obfuscation

[Barak-Goldreich-Impagliazzo-Rudich-Sahai-Vadhan-Yang'01]

"$\mathcal{O}$(P) reveals no more info than black-box access to P".

BAD NEWS: There are "unobfuscatable" programs!

[Barak-GIRSVY'01, Goldwasser-Kalai'05]



Unobfuscatable Programs

THEOREM [BAD NEWS, BGIRSVY'01]:
$\forall \mathcal{O}$ $\exists$P **such that** $\mathcal{O}$ **completely fails to obfuscate** P.

Proof: "Programs that eat themselves"

Define a family of programs $\{P_{x,y}\}$ where *x* and *y* are n-bit strings, as follows:

$$P_{x,y}(b, \Pi) = \begin{cases} y & \text{if } b=0 \text{ and } \Pi = x \\ x, y & \text{if } b=1 \text{ and } \Pi(0,x) = y \\ 0 & \text{otherwise} \end{cases}$$

Unobfuscatable Programs

THEOREM [BAD NEWS, BGIRSVY'01]:
$\forall\mathcal{O}\ \exists$P **such that** $\mathcal{O}$ **completely fails to obfuscate** P.

Proof: "Programs that eat themselves"

Define a family of programs $\{P_{x,y}\}$ where *x* and *y* are n-bit strings, as follows:

$$P_{x,y}(b,\Pi) = \begin{cases} y & \text{if } b=0 \text{ and } \Pi = x \\ x, y & \text{if } b=1 \text{ and } \Pi(0,x) = y \\ 0 & \text{otherwise} \end{cases}$$

Define a family of programs $\{P_{x,y}\}$ where *x* and *y* are n-bit strings, as follows:

$$P_{x,y}(b,\Pi) = \begin{cases} \text{y} & \text{if b=0 and } \Pi = \text{x} \\ \text{x, y} & \text{if b=1 and } \Pi\text{(0,x) = y} \\ 0 & \text{otherwise} \end{cases}$$

1. Black-box access to P is useless:

For random x and y, cannot distinguish between black-box to $P_{x,y}$ versus black-box access to the all-zero function.

2. Can recover source from obfuscated code:

Given $P' = \mathcal{O}(P_{x,y})$, simply run $P'(1, P')$.

Defining Program Obfuscation (Take 2)

Virtual Black-Box (VBB) obfuscation

[Barak-Goldreich-Impagliazzo-Rudich-Sahai-Vadhan-Yang’01]

“$\mathcal{O}$(P) betrays no more info than black-box access to P”.

BAD NEWS: There are “unobfuscatable” programs!

[BGIRSVY’01, Goldwasser-Kalai’05]

“Indistinguishability obfuscation”: Much weaker.

[BGIRSVY’01, Goldwasser-Rothblum’05]

GOOD NEWS #1: No impossibility results and even candidate constructions. [Garg-Gentry-Halevi-Raykova-Sahai-Waters’13]

GOOD NEWS #2:

IO + Basic Crypto + Hard Work = Nearly All Applications.

[Sahai-Waters’14 and many followups]

Defining Program Obfuscation (Take 2)

Indistinguishability Obfuscation (IO) for Circuits:
[Barak-Goldreich-Impagliazzo-Rudich-Sahai-Vadhan-Yang'01]

A probabilistic poly-time algorithm $\mathcal{O}$ is an indistinguishability obfuscator if:

It is Correct:
For any circuit C, $\mathcal{O}$(C) is functionally the same as C.

It is Secure:
For any two functionally equivalent circuits C1 and C2 of the same size, $\mathcal{O}$(C1) is computationally indistinguishable from $\mathcal{O}$(C2).

An Example

$C_1(x,y)$: OUTPUT $(x+y)(x-y)$	$\equiv$	$C_2(x,y)$: OUTPUT x^2-y^2

$\Downarrow$ Indistinguishability obfuscation $\Downarrow$

$C_1(x,y)$: OUTPUT $(x+y)(x-y)$	$\approx_c$	$C_2(x,y)$: OUTPUT x^2-y^2

Indistinguishability Obfuscation:

Reveals the truth table, hides the *implementation.*

IO exists if P = NP

[BGIRSVY'01]

***Computationally inefficient* IO exists.**

Given a circuit C, output the lexicographically smallest equivalent circuit C'.

If P=NP, this strategy can be implemented efficiently.

(Even better, this is a *perfect* IO.)

Corollary:

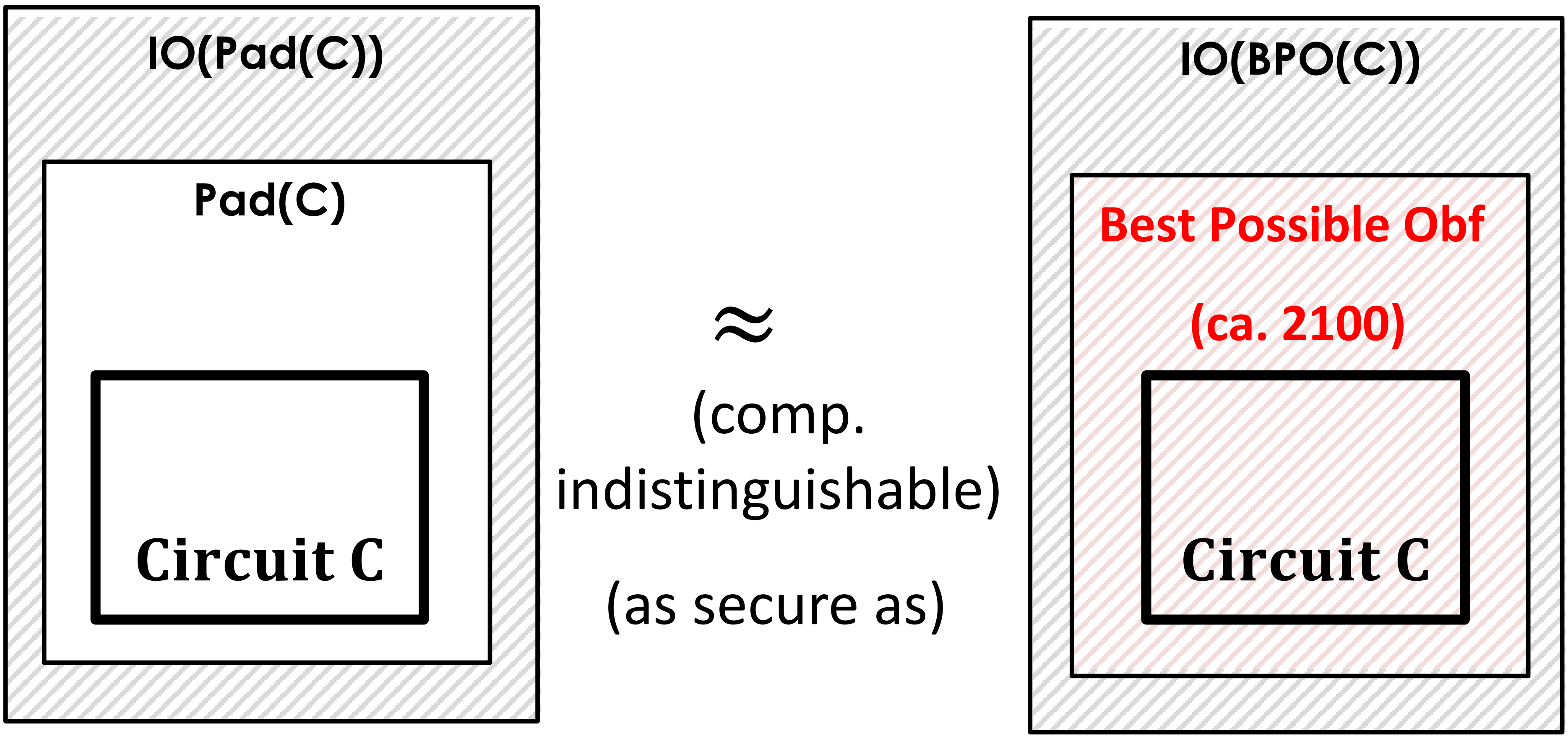
IO does not imply any crypto (even one-way functions).

Suppose IO $\Longrightarrow$ OWF.

Then, P = NP $\Longrightarrow$ $\exists$OWF, a contradiction.

IO is a "Best Possible" Obfuscation

[Barak-Goldreich-Impagliazzo-Rudich-Sahai-Vadhan-Yang'01, Goldwasser-Rothblum'17]



More Theorems on IO

If Perfect (even Statistical) IO exists, then PH collapses.

[Goldwasser-Rothblum'07]

IO is equivalent to VBB with an unbounded simulator.

"Mildly compressing" IO + "standard crypto" implies IO.

[Ananth-Jain'15, Bitansky-V.'15, Lin-Pass-Seth-Telang'16]

XIO is IO with two relaxations:

1. Obfuscator can run in $\mathrm{poly}(2^n)$ time.
2. Obfuscated circuit has size $2^{(1-\varepsilon)n}$ for some $\varepsilon > 0$.

“CRYPTO-COMPLETE”:

IO + Basic Hardness + Hard Work ⟹ Nearly all crypto.

Two-round MPC

PPAD Hardness

One-way Functions

Public-key Encryption

“Pure” FHE*

Time-lock Puzzles*

Succinct RE

Constrained PRFs

Deniable Encryption

Functional Encryption

Non-interactive Key Exchange

Correlation-intractable fns

Traitor Tracing

NIWI

Software Watermarking

Compact Token-based Obfuscation

IO + “basic hardness”

TUTORIAL OUTLINE

Part 1. DEFINITIONS

of program obfuscation

a. Virtual Black-Box OBF

b. Indistinguishability OBF (IO)

Part 2. APPLICATIONS of IO

a. Crypto Applications

b. A Complexity Application

c. Bootstrapping Theorems

Part 3. CONSTRUCTIONS

of IO from simpler objects

Theorem: If 3-linear maps exist, so does IO.

Part 4. DE-IO-IZATION

Remove the need for IO in applications.

e.g., Traitor Tracing (on Wed)

Application 1: One-way Functions

THEOREM **[Komargodski-Moran-Naor-Pass-Rosen-Yogev'14]**

If IO exists and $NP \nsubseteq i.o\text{-}coRP$, one-way functions exist.

One-way Function CONSTRUCTION:

G(r) = $\mathcal{O}$(Z; r) where Z is the Zero circuit (Z(x) = 0 for all x)

Suppose there is an inverter Inv.

If F is UNSAT, then Inv "inverts" $\mathcal{O}$(F; r).

// Outputs r' such that $\mathcal{O}$(Z; r') = $\mathcal{O}$(F; r)

If F is SAT, then Inv ***cannot*** "invert" $\mathcal{O}$(F; r).

// since the sets $\{\mathcal{O}(F; r)\}_r$ and $\{\mathcal{O}(Z; r)\}_r$ are disjoi

<u>Satisfiability Algorithm, on input a formula F:</u>

If Inv inverts $\mathcal{O}$(F; r), output UNSAT else output SAT.

Application 2: Public-key Encryption

THEOREM [Garg-Gentry-Sahai-Waters'13, Sahai-Waters'14]

If IO and OWF exist, so does public-key encryption.

Public-key Encryption CONSTRUCTION:

Let G: $\{0,1\}^n \to \{0,1\}^{2n}$ be a cryptographic PRG.

Secret key = s $\leftarrow_R$ $\{0,1\}^n$ and Public key = G(s)

Enc(PK, m) $\leftarrow_R$ $\mathcal{O}(C_{PK,m})$ where

$$C_{PK,m}(x) = \begin{cases} m & \text{if } G(x) = PK \\ \perp & \text{otherwise} \end{cases}$$

EXPT 0: Adv gets PK and ciphertext $\mathcal{O}(C_{PK,m})$.

$\approx_{\boldsymbol{PRG}}$

EXPT 1: Adv gets $\widetilde{PK}$ and ciphertext $\mathcal{O}(C_{\widetilde{PK},m})$

where $\widetilde{PK}$ is uniformly random.

$\approx_{\boldsymbol{IO}}$

(note: w.h.p. $\widetilde{PK}$ lives outside the image of G)

EXPT 2: Adv gets $\widetilde{PK}$ and ciphertext $\mathcal{O}(Z)$

where the circuit Z always outputs $\perp$.

$\blacksquare$

THEOREM [Komargodski-Moran-Naor-Pass-Rosen-Yogev'14]

If IO exists and $NP \nsubseteq i.o\text{-}coRP$, one-way functions exist.

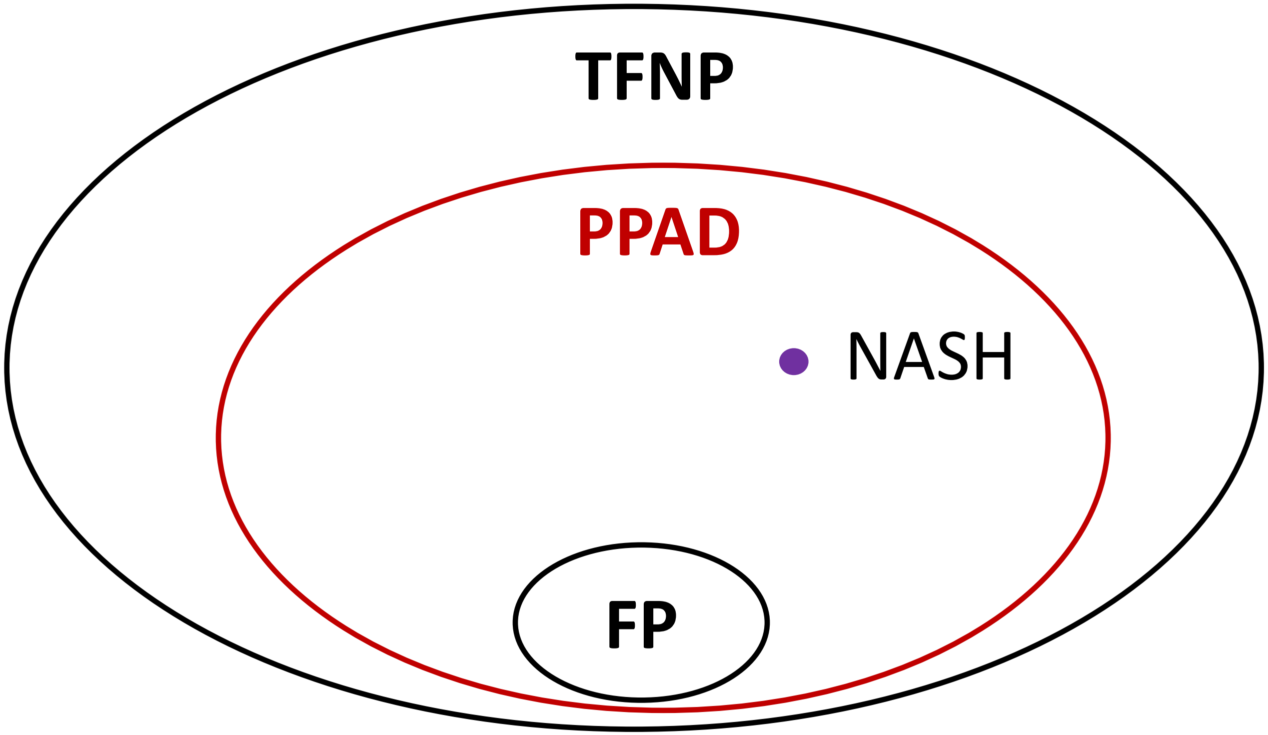
THEOREM [Garg-Gentry-Sahai-Waters'13, Sahai-Waters'14]

If IO and OWF exist, so does public-key encryption.

COMMON THEME:

IO "lifts" hardness into *useful* hardness.

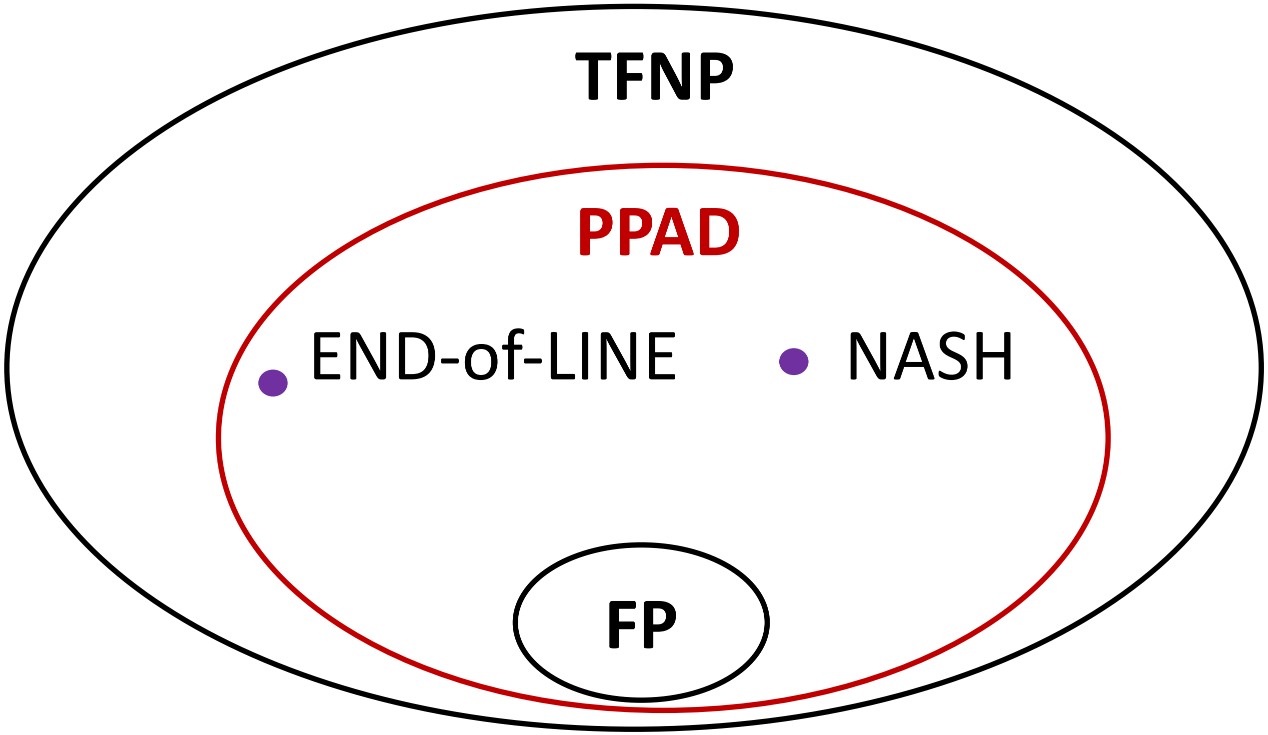
Application 3: PPAD-Hardness



PPAD [Papadimitriou'94]: Totality is proved via "a parity argument in directed graphs"

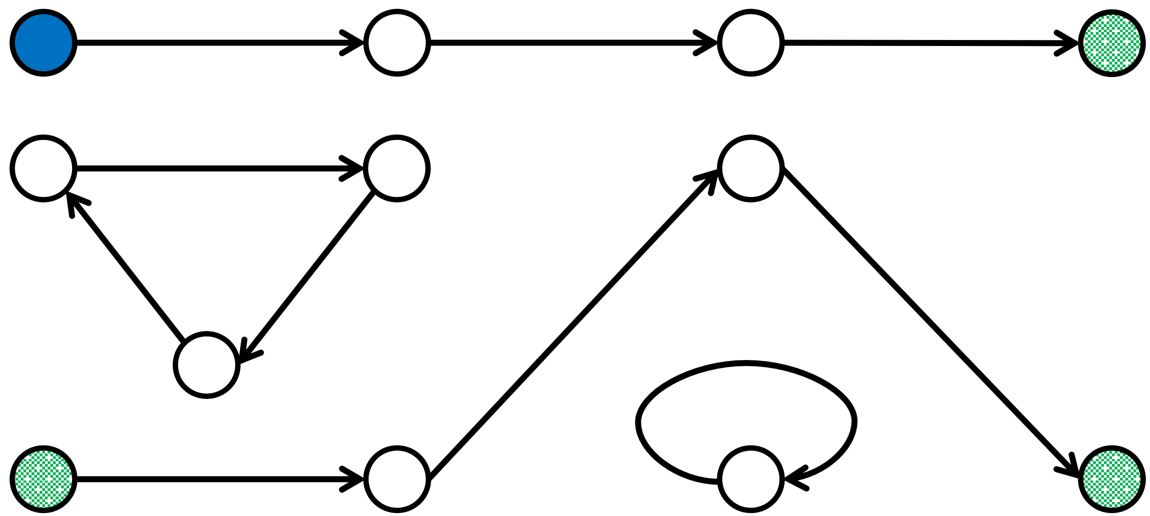
NASH is complete for PPAD [DGP'05, CD'05].

Application 3: PPAD-Hardness

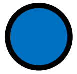


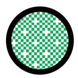
Canonical complete problem: **END-of-LINE** [Pap'94]

The END-of-LINE Problem



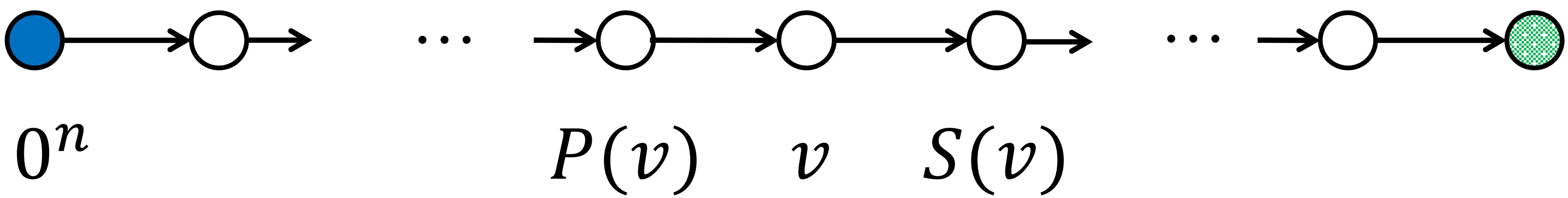
Input: A graph with in/out degree ≤ 1

A source: 

Output: Another source/sink: 

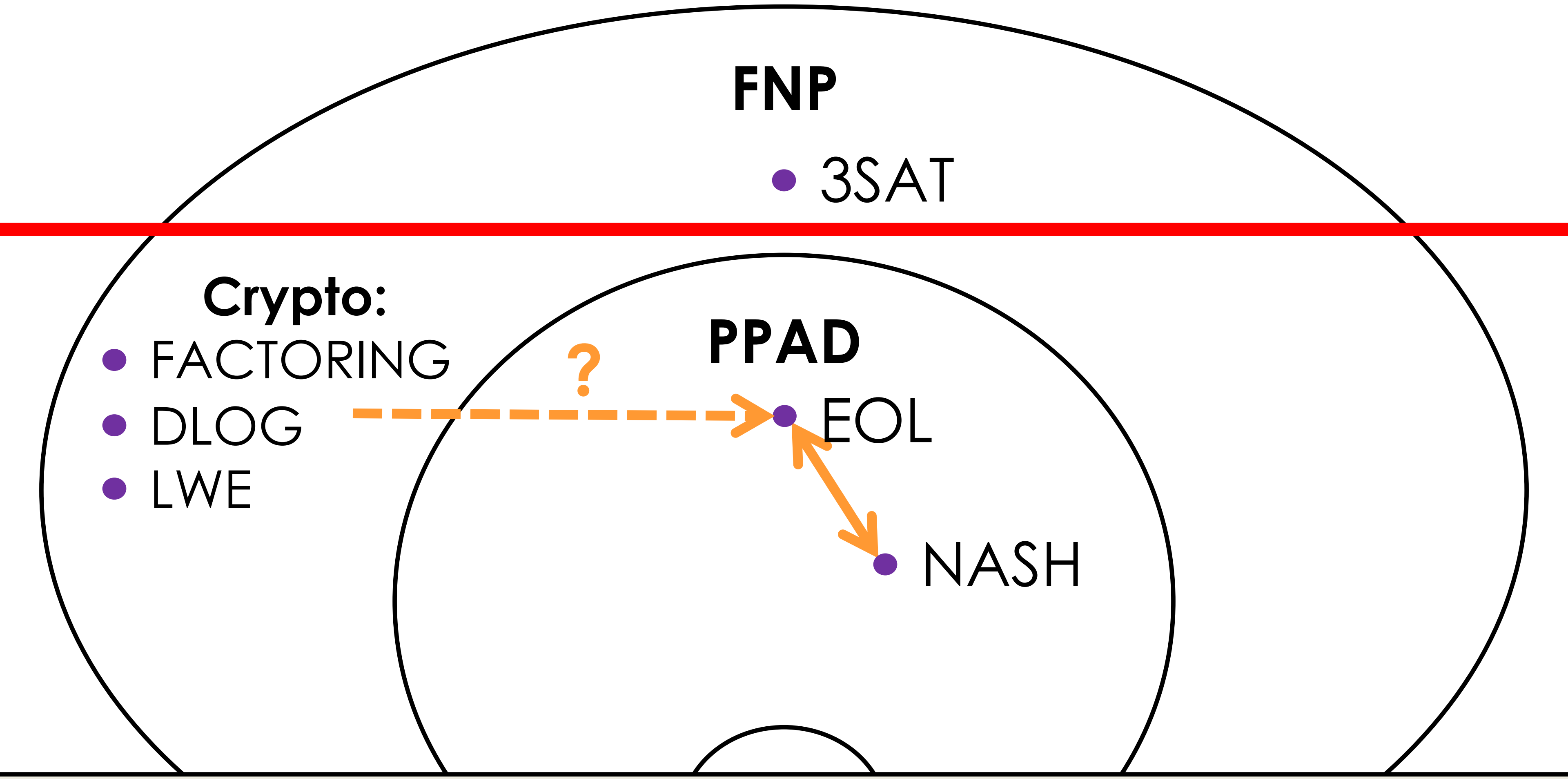
The END-of-LINE Problem

Exponential size graph:



Nodes are in $\{0,1\}^n$

Edges defined by programs $S, P: \{0,1\}^n \rightarrow \{0,1\}^n$



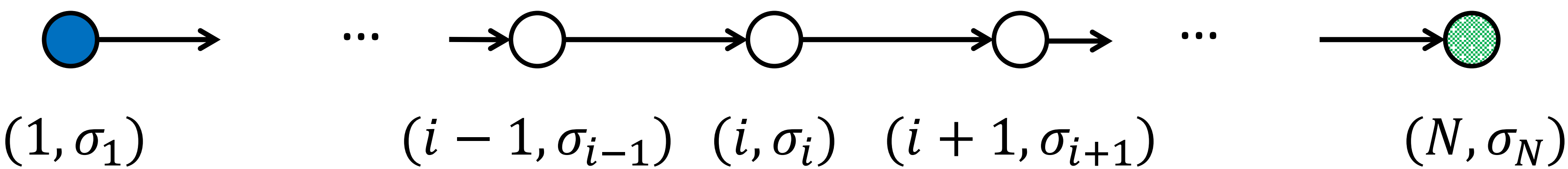
THEOREM [Bitansky-Paneth-Rosen’15]

If IO and OWF exist, END-of-LINE is (average-case) hard.

(Previously Abbott-Kane-Valiant’05 from Super-VBB)

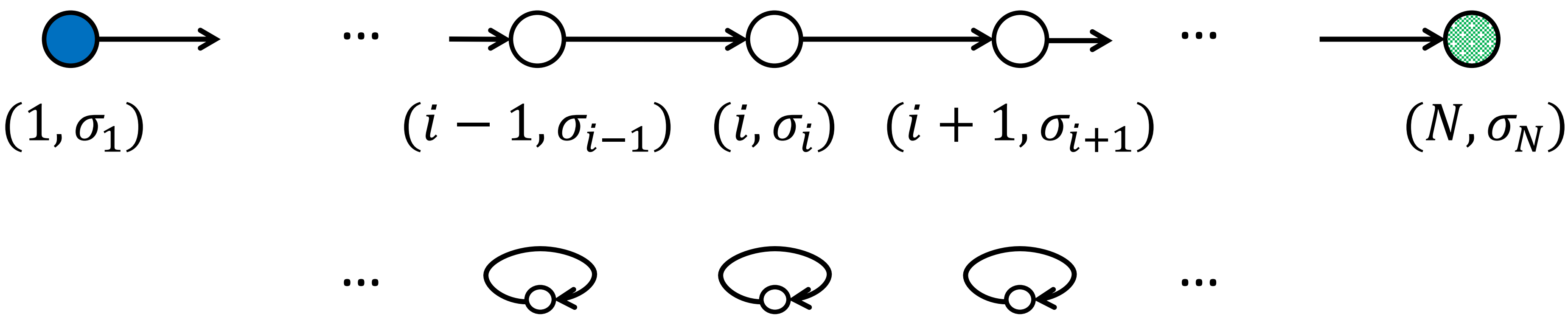
Constructing the Hard EOL Instance

Using a pseudorandom **function** f_k, construct a graph



where $\sigma_i = f_k(i)$.

Slide Courtesy: Omer Paneth



S:

$\boldsymbol{S_k(i, \sigma)}$:

if $(\mathrm{i}, \sigma) = (N, \sigma_N)$:
    return "*sink*"

If $(\mathrm{i}, \sigma) = (i, \sigma_i)$:
    return $(i+1, \sigma_{i+1})$

else:
    return (i, σ)

P:

$\boldsymbol{P_k(i, \sigma)}$:

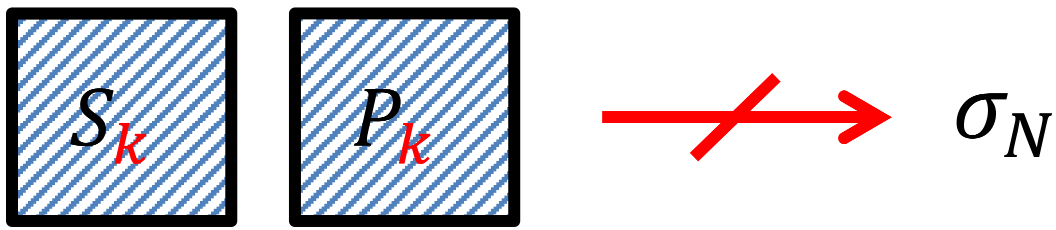
if $(\mathrm{i}, \sigma) = (1, \sigma_1)$:
    return "*source*"

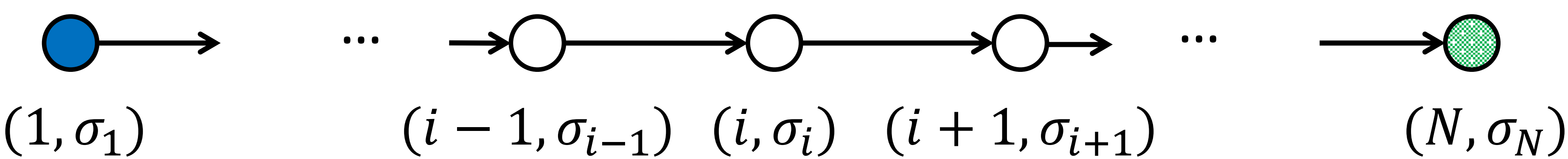
If $(\mathrm{i}, \sigma) = (i, \sigma_i)$:
    return $(i-1, \sigma_{i-1})$

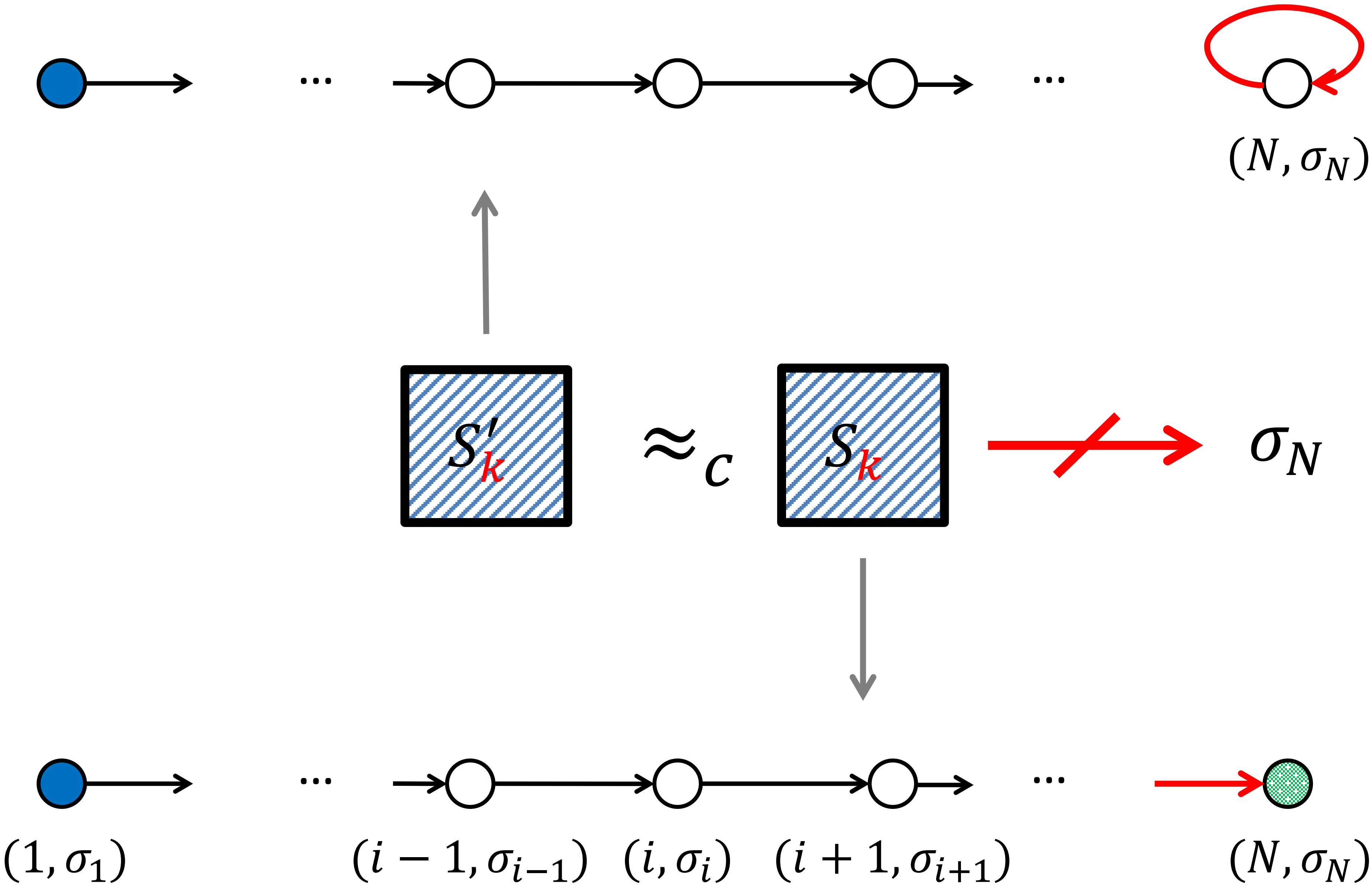
else:
    return (i, σ)

Slide Courtesy: Omer Paneth

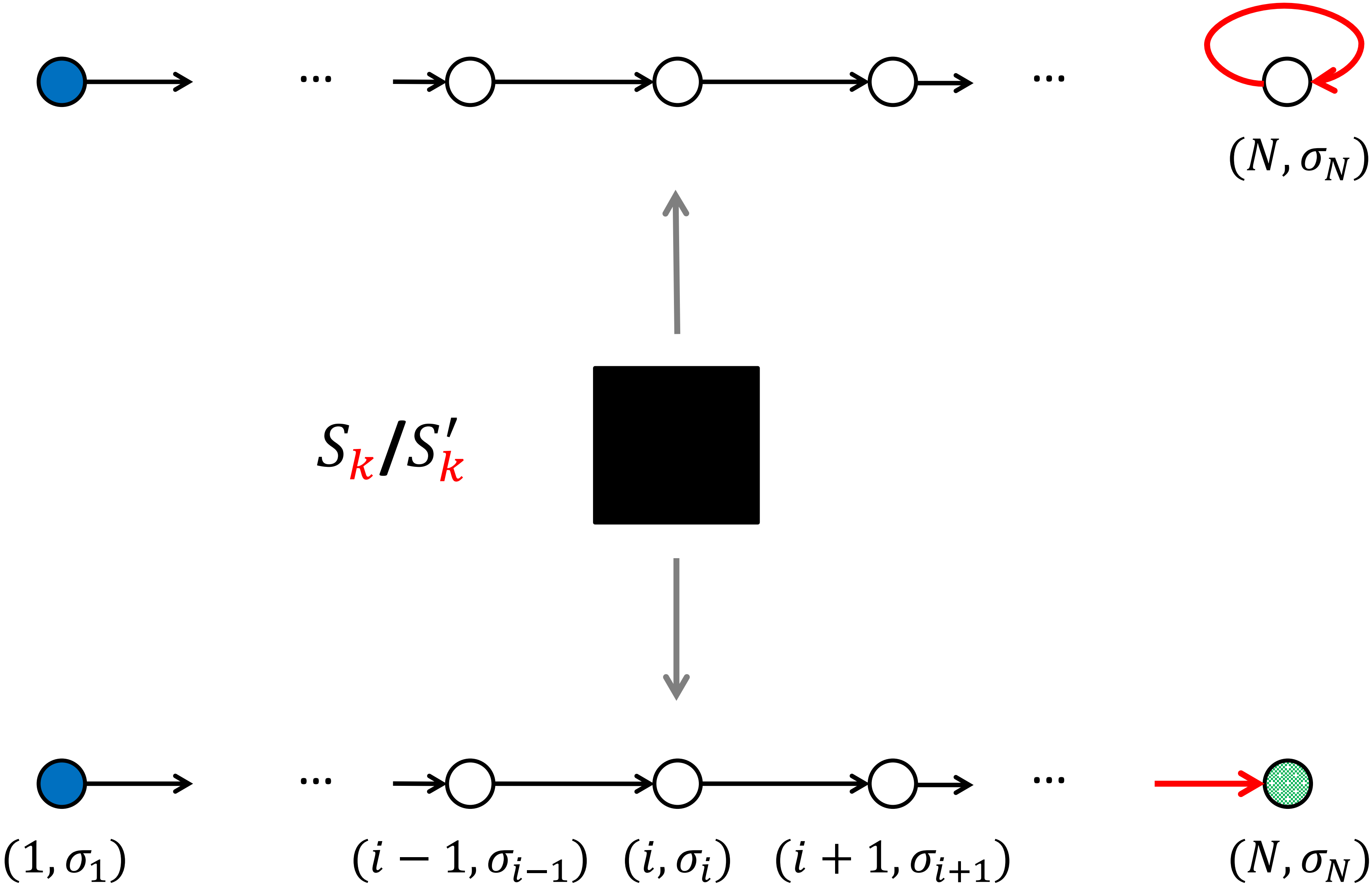
Need To Prove



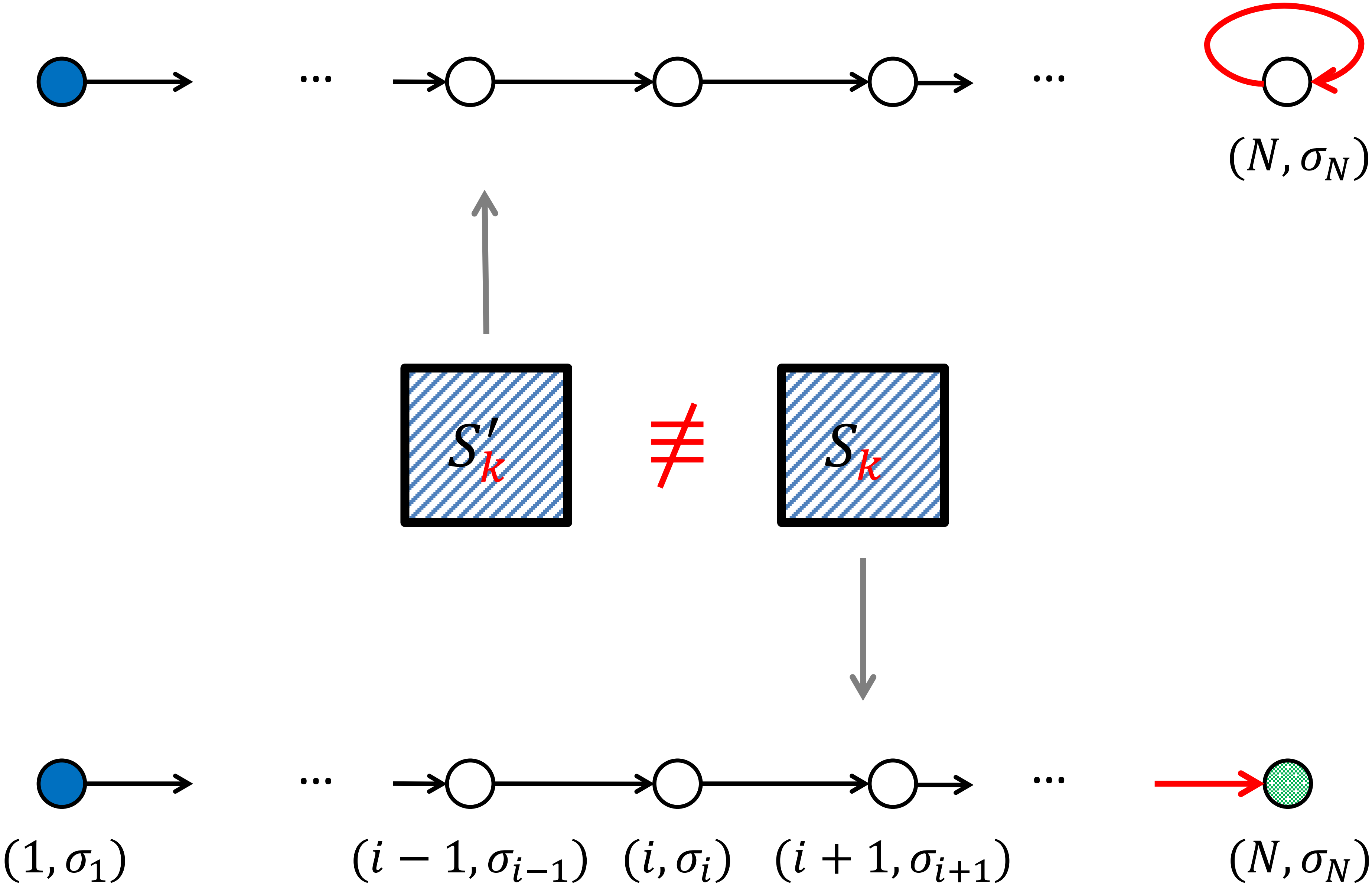




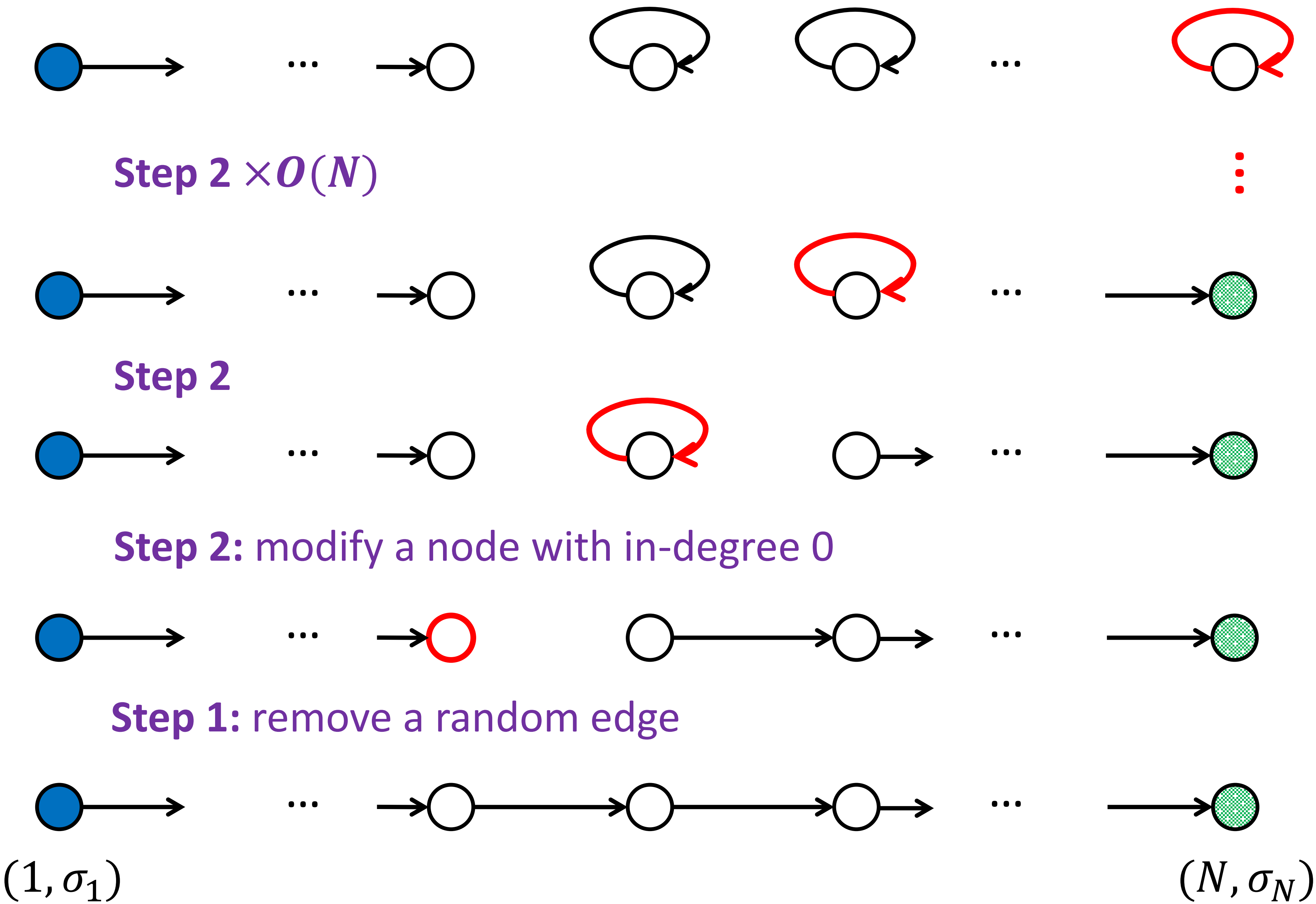
(N, σ_N)
S'_k
$\approx_c$
S_k
σ_N
$(1, \sigma_1)$
$(i-1, \sigma_{i-1})$
(i, σ_i)
$(i+1, \sigma_{i+1})$
(N, σ_N)



(N, σ_N)
S_k/S'_k
$(1, \sigma_1)$
$(i-1, \sigma_{i-1})$
(i, σ_i)
$(i+1, \sigma_{i+1})$
(N, σ_N)



(N, σ_N)
S'_k
$\not\equiv$
S_k
$(1, \sigma_1)$
$(i-1, \sigma_{i-1})$
(i, σ_i)
$(i+1, \sigma_{i+1})$
(N, σ_N)



Step 2 $\times O(N)$
Step 2
Step 2: modify a node with in-degree 0
Step 1: remove a random edge
$(1, \sigma_1)$
(N, σ_N)

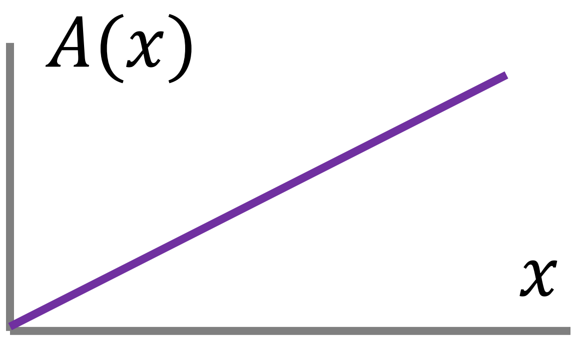
A Useful Lemma

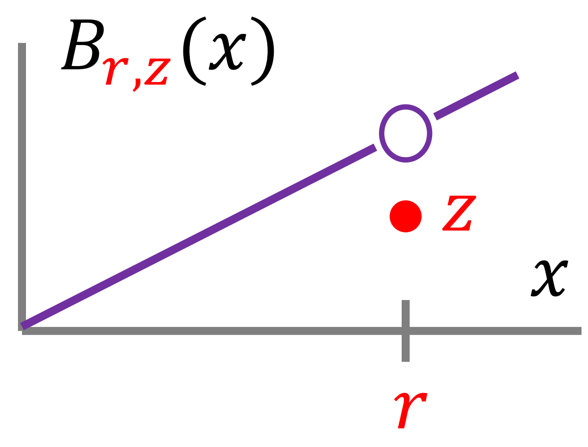
$\boldsymbol{A}$

$\boldsymbol{B_{r,z}(x)}$:

if $x = r$: return z

else: return $A(x)$





A Useful Lemma

$\boldsymbol{A}$	$\boldsymbol{B_{r,z}(x)}$:	
	if $x = r$:	return z
	else:	return $A(x)$

For a random r and for all z:

$\boldsymbol{A}$	$\approx_c$	$\boldsymbol{B_{r,z}(x)}$:	
		if $x = r$:	return z
		else:	return $A(x)$

Proof of Lemma (using ideas from [Sahai-Waters14])

$\boldsymbol{A}$

$\boldsymbol{B_{r,z}(x)}$:

if $x = r$: return z

else: return $A(x)$

$\approx_c$ using IO

Also using an Injective,

length doubling PRG:

$$g: \{0,1\}^n \to \{0,1\}^{2n}$$

$\boldsymbol{B^*_{s=g(r),z}(x)}$:

if $g(x) = s$: return z

else: return $A(x)$

Proof of Lemma

$\boldsymbol{A}$

$\boldsymbol{B}_{r,z}(\boldsymbol{x})$:

if $x = r$: return z

else: return $A(x)$

using IO $\approx_c$

using g

$\approx_c$ using IO

$\boldsymbol{B}^*_{s \leftarrow U, z}(\boldsymbol{x})$:

if $g(x) = s$: return z

else: return $A(x)$

$\approx_c$

$\boldsymbol{B}^*_{s = g(r), z}(\boldsymbol{x})$:

if $g(x) = s$: return z

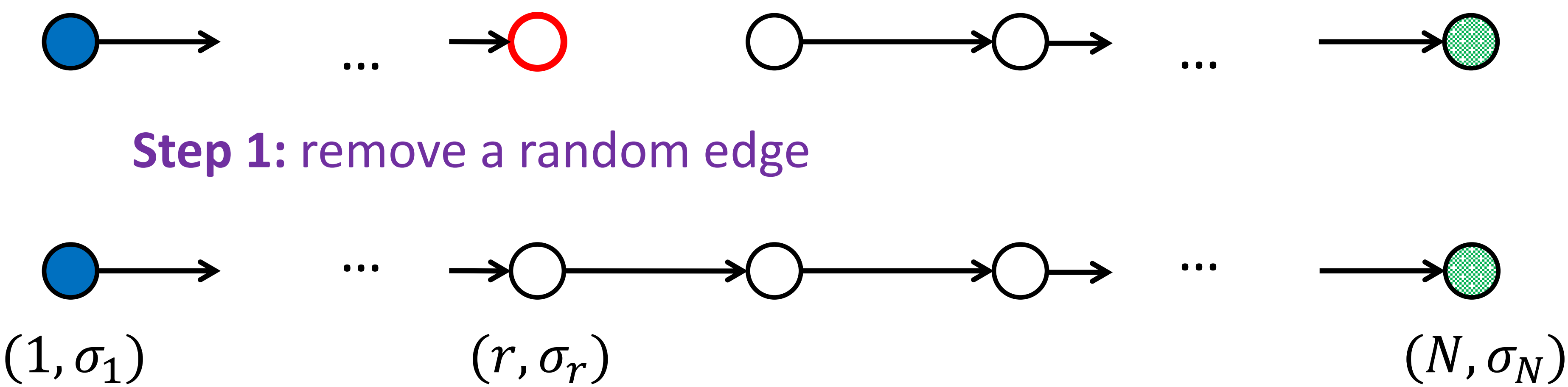
else: return $A(x)$

Step 1 - Proof

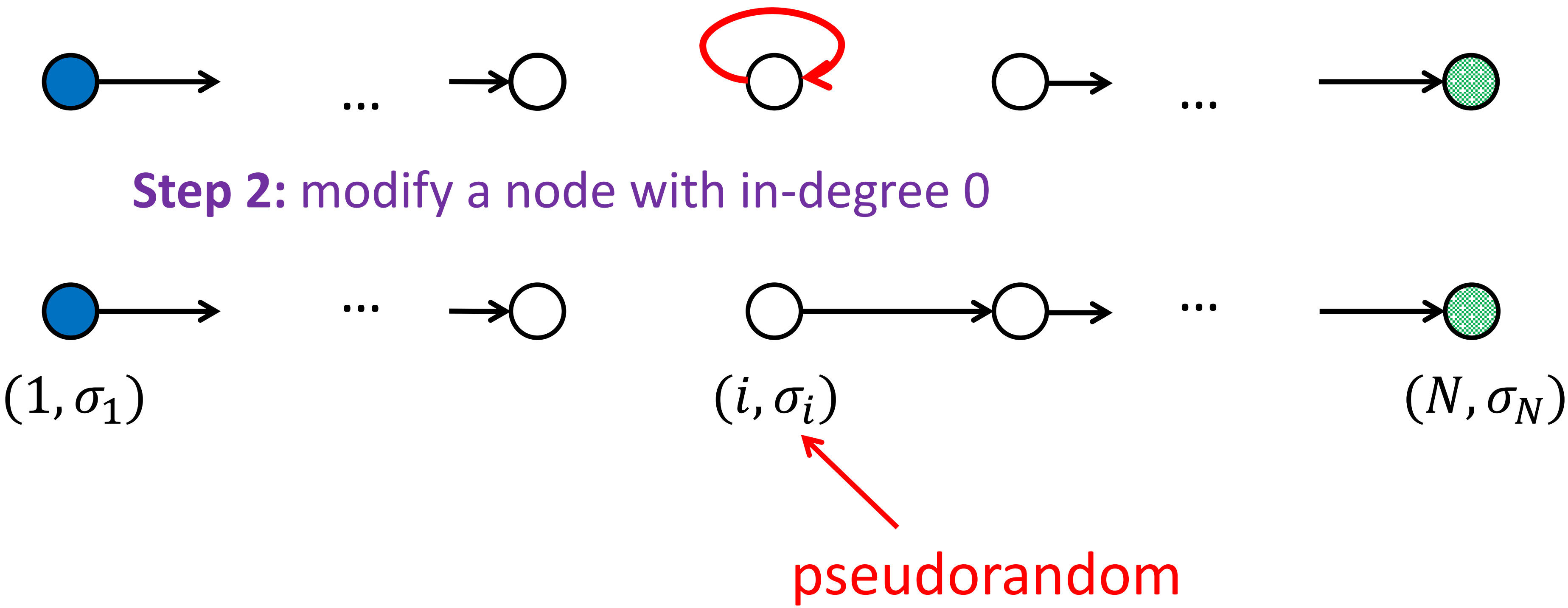
$\boldsymbol{S_k(i, \sigma)}$ $\approx_c$ $\boldsymbol{S'_{k,r}(i, \sigma)}$:

if $i = r$: return $\perp$

else: return $\boldsymbol{S_k(i, \sigma)}$



Step 2 - Proof



...
...
Step 2: modify a node with in-degree 0
...
...
$(1, \sigma_1)$
(i, σ_i)
(N, σ_N)
pseudorandom

Interlude: Pseudorandom Functions (PRFs)

Family of poly-time computable functions $\{F_K\}$ such that no poly-time oracle alg. can distinguish between oracle access to F_K vs. oracle access to a truly random function.

<u>Theorem</u> [Goldreich-Goldwasser-Micali'84 + Hastad-Impagliazzo-Levin-Luby'89]
If one-way functions exist, so do PRFs.

Useful Tool: Punctured PRFs

Can create a "punctured key" $K\{x\}$ which

– Allows anyone to compute $F_K(y)$ for $y \neq x$, but
– Hides $F_K(x)$

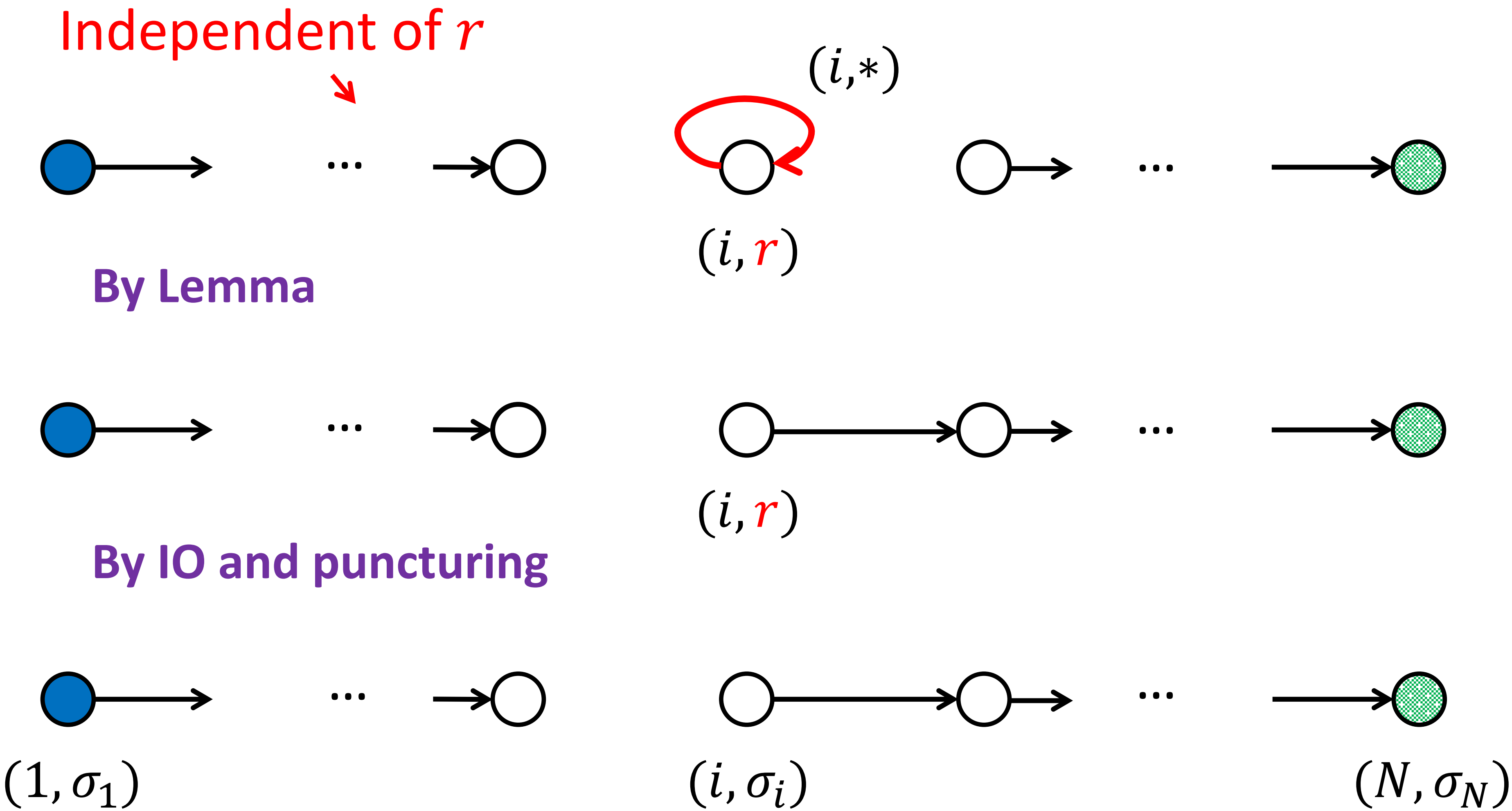
THEOREM [Boyle-Goldwasser-Ivan'13,Boneh-Waters'13,Kiayias-Papadopoulos-Triandopoulos-Zacharias'13]

If one-way functions exist, so do punctured PRFs.

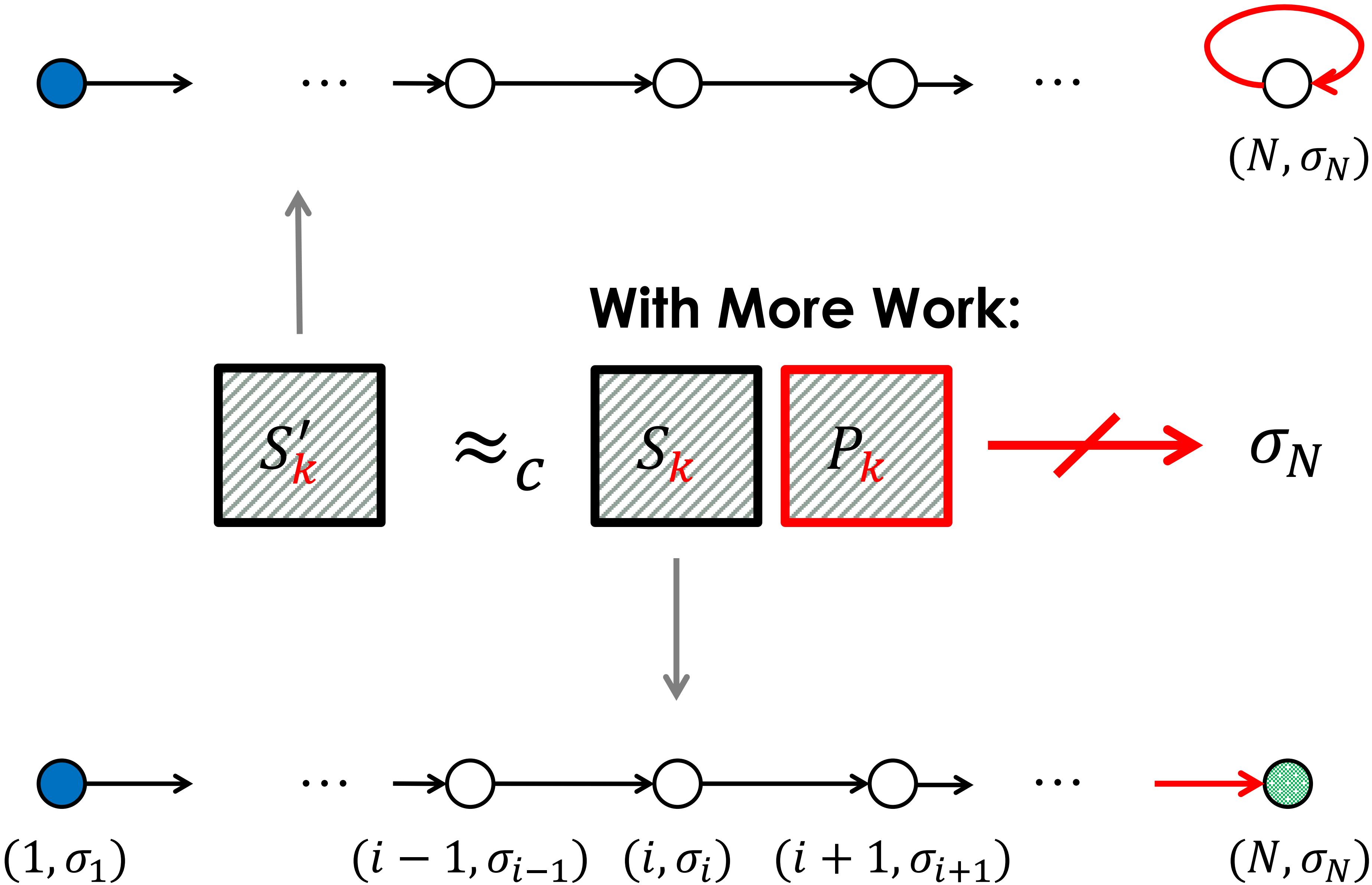
<u>An Observation:</u>

Punctured PRFs are "mildly obfuscatable" already.

Step 2 - Proof

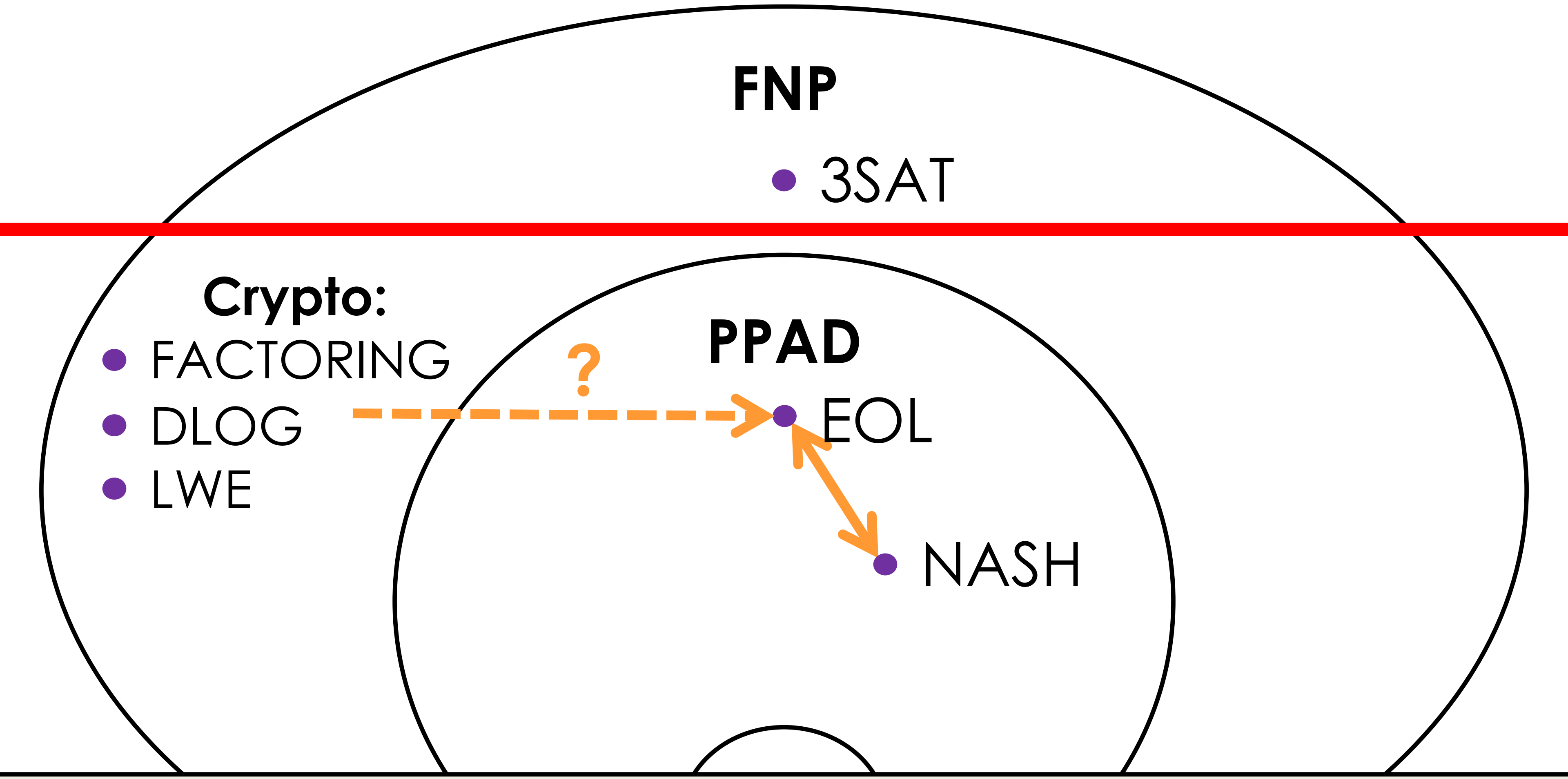


Independent of r
$(i,*)$
(i,r)
By Lemma
(i,r)
By IO and puncturing
$(1,\sigma_1)$
(i,σ_i)
(N,σ_N)



$\cdots$
$\cdots$
(N, σ_N)
With More Work:
$S'_k \approx_c S_k \, P_k \not\rightarrow \sigma_N$
$\cdots$
$\cdots$
$(1, \sigma_1)$
$(i-1, \sigma_{i-1})$
(i, σ_i)
$(i+1, \sigma_{i+1})$
(N, σ_N)

Slide Courtesy: Omer Paneth



THEOREM [Bitansky-Paneth-Rosen'15]

If IO and OWF exist, END-of-LINE is (average-case) hard.

(Previously Abbott-Kane-Valiant'05 from Super-VBB)

TUTORIAL OUTLINE

Part 1. DEFINITIONS

of program obfuscation

a. Virtual Black-Box OBF

b. Indistinguishability OBF (IO)

Part 2. APPLICATIONS of IO

a. Crypto Applications

b. A Complexity Application

c. Bootstrapping Theorems

Part 3. CONSTRUCTIONS

of IO from simpler objects

Theorem: If 3-linear maps exist, so does IO.

Part 4. DE-IO-IZATION

Remove the need for IO in applications.

e.g., Traitor Tracing (on Wed)

IO Bootstrapping Theorems

1. **From Simple Circuits to All Circuits.**
   IO for a circuit class C implies IO for P assuming either:
   - Fully homomorphic encryption with decryption in C
     [Garg-Gentry-Halevi-Raykova-Sahai-Waters'13]

     OR

   - Sub-exponentially secure PRFs computable in C
     [Applebaum'15, Canetti-Lin-Tessaro-V.'15]

2. **From Circuits to Turing Machines and RAM Machines.**
   IO for circuits implies IO TMs and RAMs assuming that sub-exponentially secure PRGs exist.
   [Canetti-Holmgren-Jain-V.'15, Bitansky-Garg-Lin-Pass-Telang'15, Koppula-Lewko-Waters'15, Canetti-Holmgren'16]

From Simple Circuits to All Circuits

THEOREM [Canetti-Lin-Tessaro-V.'15]

If (subexp. secure) IO for NC1 exists and PRFs computable in NC1 exist, so does IO for P.

KEY TOOL: RANDOMIZED ENCODINGS [Ishai-Kushilevitz'98, Yao'86]

A randomized encoding **RE** is a probabilistic algorithm:

- takes a pair $(\boldsymbol{C}, \boldsymbol{x})$ and outputs a pair $\left(\widehat{\boldsymbol{C}}, \widehat{\boldsymbol{x}}\right)$.
- Given $\widehat{\boldsymbol{C}}$ and $\widehat{\boldsymbol{x}}$, one can compute **C(x)**.
- Given **C(x)**, can simulate the distribution of $\left(\widehat{\boldsymbol{C}}, \widehat{\boldsymbol{x}}\right)$.
- **RE** can be computed in parallel (same depth as a PRF).

From Simple Circuits to All Circuits

THEOREM [Canetti-Lin-Tessaro-V.'15]
If (subexp. secure) IO for NC1 exists and PRFs computable in NC1 exist, so does IO for P.

CONSTRUCTION IDEA:

"Don't compute **C(x).** Compute **RE(C,x).**"

$\mathcal{O}$(C) =

$\boldsymbol{P_{C,K}(x)}$
Generate randomness $r = F_K(x)$
Output $\mathrm{RE}(C, x; r)$.

Observe: P is a "low-depth" circuit if F_K is "low-depth".

TUTORIAL OUTLINE

Part 1. DEFINITIONS of program obfuscation

a. Virtual Black-Box OBF

b. Indistinguishability OBF (IO)

Part 2. APPLICATIONS of IO

a. Crypto Applications

b. A Complexity Application

c. Bootstrapping Theorems

Part 3. CONSTRUCTIONS of IO from simpler objects

Theorem: If 3-linear maps exist, so does IO.

Part 4. DE-IO-IZATION

Remove the need for IO in applications.

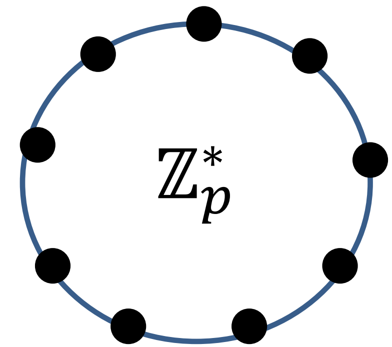
e.g., Traitor Tracing (on Wed)

Crypto and New Sources of Hardness

PUBLIC KEY ENCRYPTION

Discrete Logarithms.

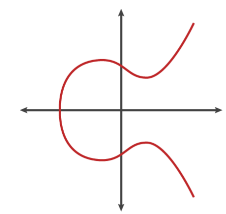
Hardness of Factoring.



Diffie-Hellman

IDENTITY-BASED ENCRYPTION

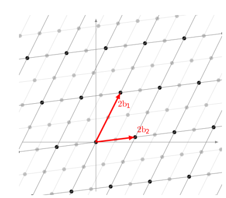
Elliptic Curves and Bilinear Maps.



[Joux, Boneh-Franklin]

FULLY HOMOMORPHIC ENCRYPTION

Integer Lattices.



[Gentry, Brakerski-V]

INDISTINGUISHABILITY OBFUSCATION

?

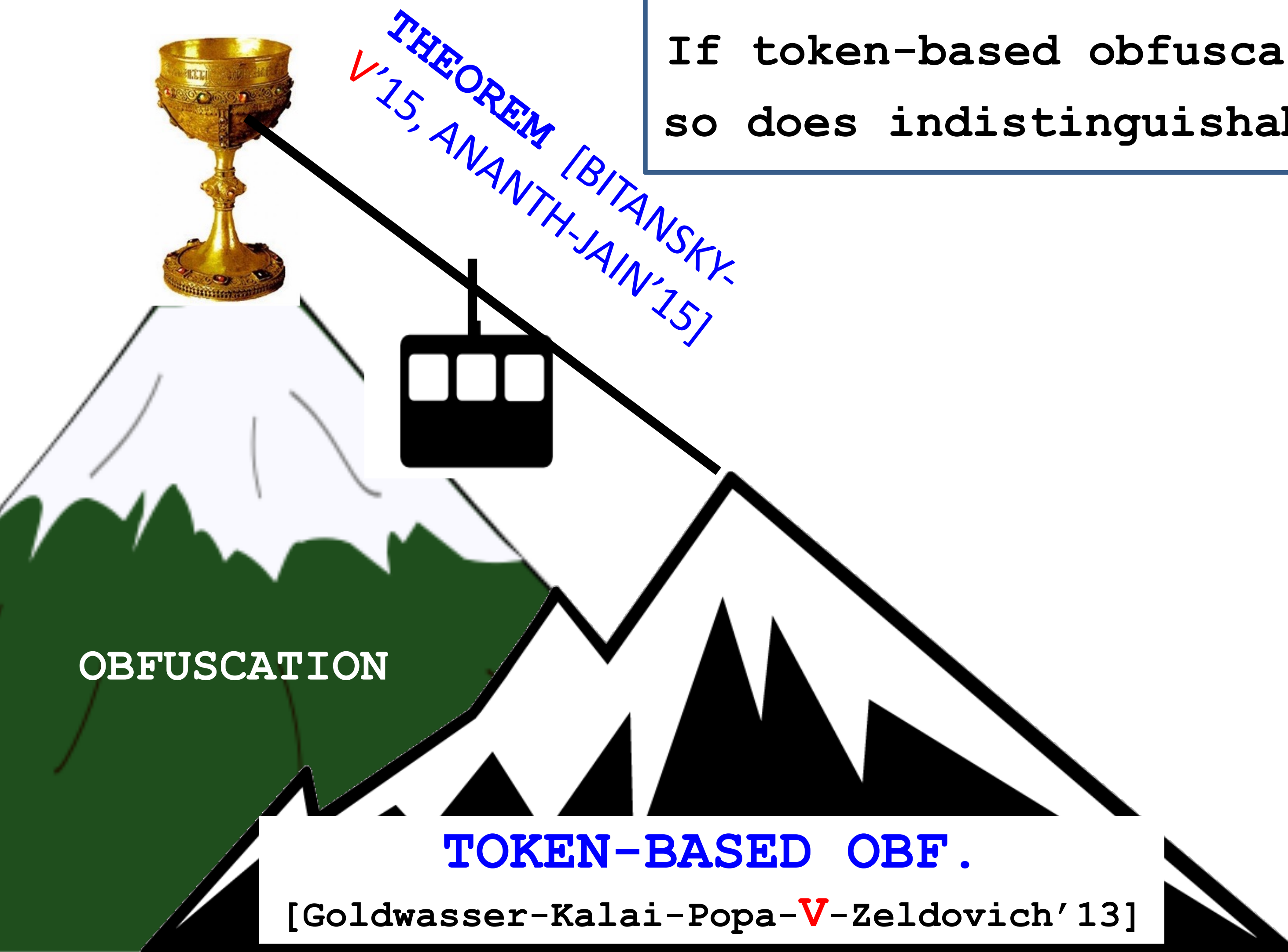
Constructing Program Obfuscators

UPSHOT: We now have candidate constructions secure against all known attacks + generalizations, but no absolute proofs of security.

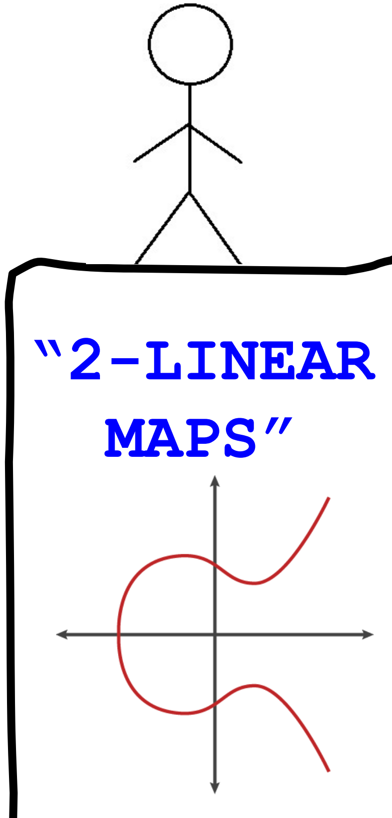
Constructing Program Obfuscators

THEOREM 1:

If token-based obfuscation exists, so does indistinguishability obf.



Constructing Program Obfuscators



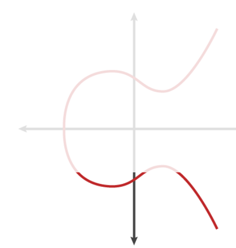
1, 2- and 3-Linear Maps

1-Linear Map: Need Group G where

Diffie-Hellman

$$g^x \cdot g^y = g^{x+y} \quad \text{BUT} \quad g^x, g^y \xrightarrow{hard} g^{xy}$$

2-Linear Map: Need Groups G, G' where



$$g^x \circ g^y = g^{xy} \quad \text{BUT} \quad g^x, g^y, g^z \xrightarrow{hard} g^{xyz}$$

[Joux, Boneh-Franklin]

3-Linear Map: Need Groups G, G' where

?

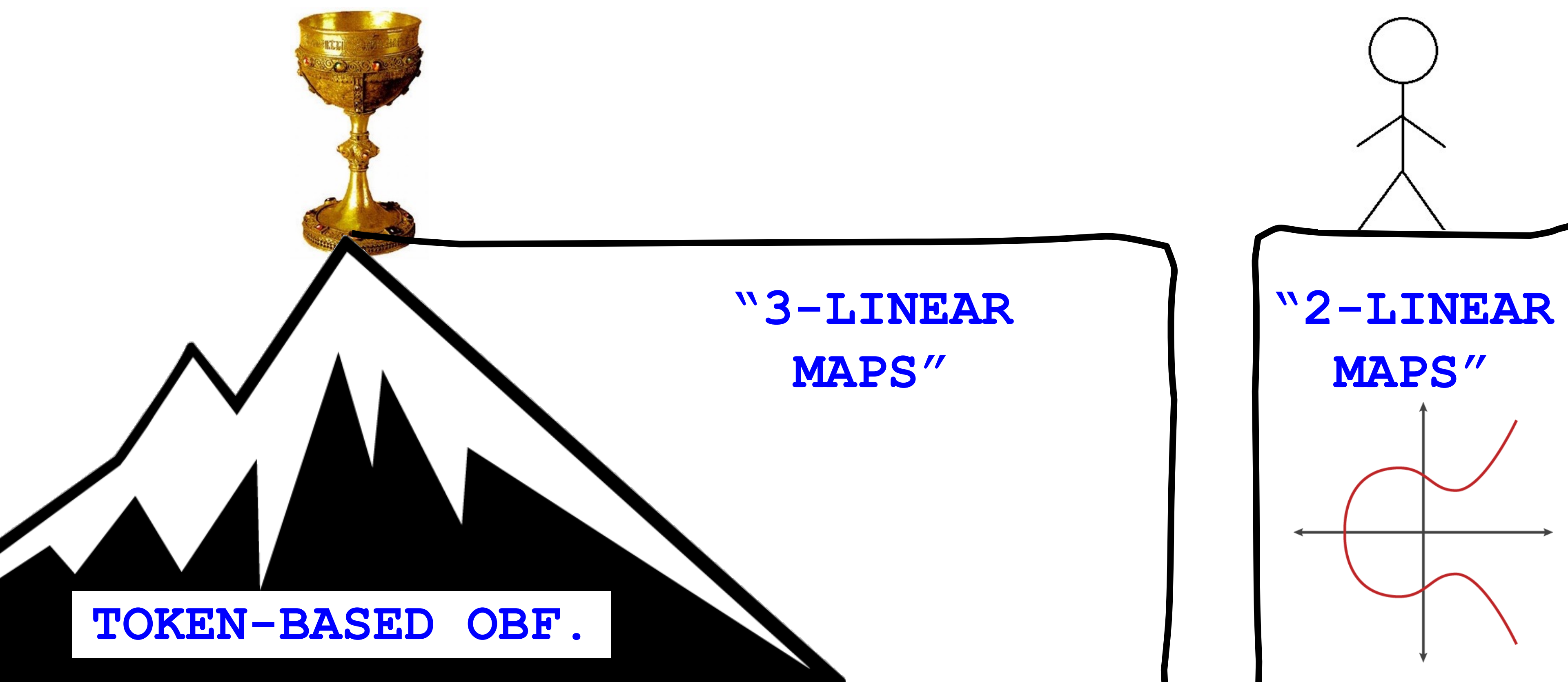
$$g^x \circ g^y \circ g^z = g^{xyz} \quad \text{BUT} \quad g^x, g^y, g^z, g^w \xrightarrow{hard} g^{xyzw}$$

See [Huang'18] for a candidate

Constructing Program Obfuscators

THEOREM 2 [Lin-V'16, Lin'17, Ananth-Sahai'17, Lin-Tessaro'17]

If 3-linear maps exist*, so does token-based obf., and therefore, indistinguishability obf.



CONSTRUCTION OUTLINE

IO for P

[Bitansky-V15, Ananth-Jain'15,
Ananth-Jain-Sahai'16, Lin-Pass-Seth-Telang'16]

$\Uparrow$

Token-based Obfuscation for P

[Lin16, Lin-V16, Lin17, Ananth-Sahai'17,
Lin-Tessaro'17]

$\Uparrow$

+ "Local PRG"

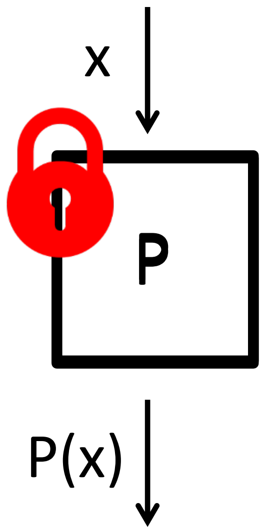
Token-based Obfuscation for NC^0

[Lin16, Lin-V16, Lin17, Ananth-Sahai'17,
Lin-Tessaro'17]

$\Uparrow$

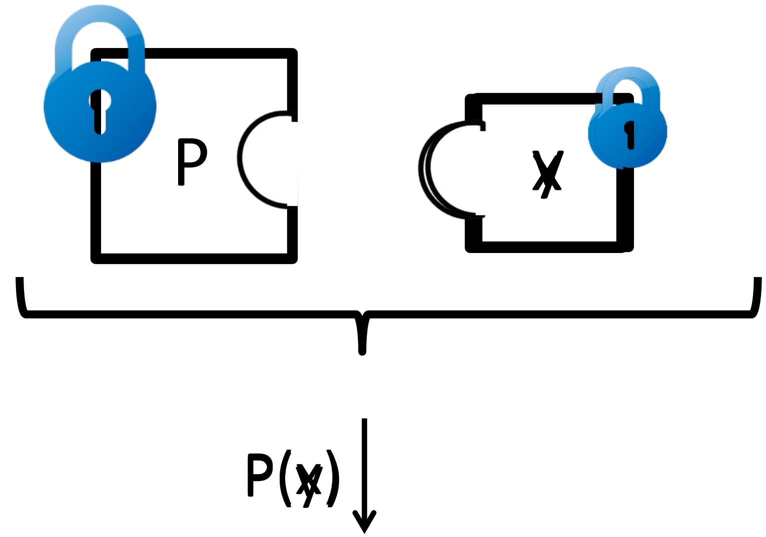
3-Linear Maps

Obfuscation



Given $\mathcal{O}$(P), can compute P(x) for any x.

Token-Based Obfuscation (TBO)



TB$\mathcal{O}$(P) is useless by itself.

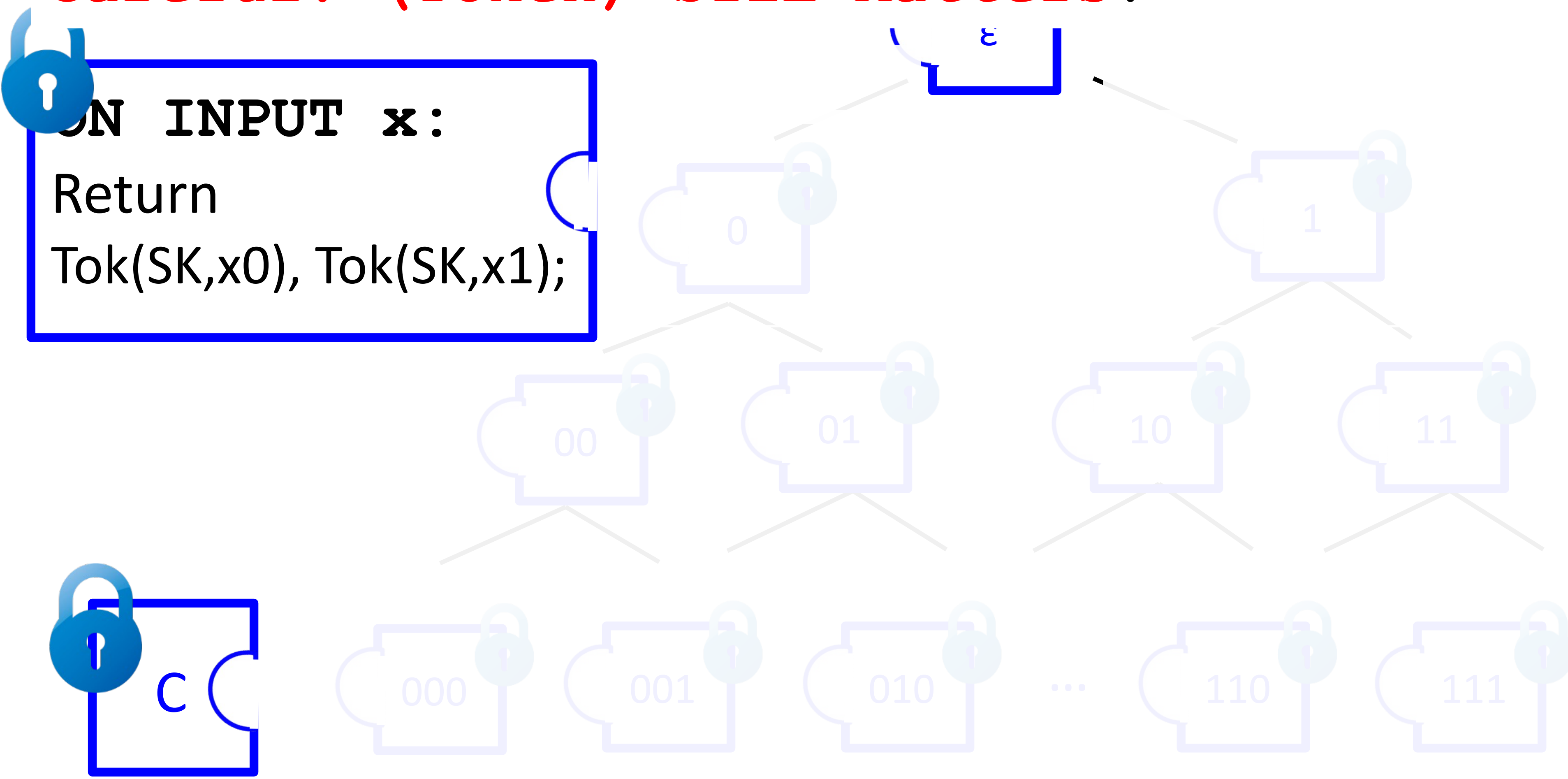
Given TB$\mathcal{O}$(P) and Tok(x), can compute P(x).

From Token-Based to Obfuscation

[Bitansky-V 15, Ananth-Jain 15]

KEY IDEA: Self-Replicating Programs (Tokens)

Careful: (Token) SIZE Matters!



From Token-Based to Obfuscation

KEY IDEA: **Self-Replicating Programs** (Tokens)

Careful: Token SIZE Matters!

[Goldwasser-Kalai-Popa-V-Zeldovich'13] uses standard crypto assumptions (Learning with Errors). However, their token size doubles every level of the tree!

CONSTRUCTION OUTLINE

IO for P

[Bitansky-V15, Ananth-Jain'15,
Ananth-Jain-Sahai'16, Lin-Pass-Seth-Telang'16]

$\Uparrow$

Token-based Obfuscation for P

[Lin16, Lin-V16, Lin17, Ananth-Sahai'17,
Lin-Tessaro'17]

$\Uparrow$

+ Local PRG

Token-based Obfuscation for NC^0

[Lin16, Lin-V16, Lin17, Ananth-Sahai'17,
Lin-Tessaro'17]

$\Uparrow$

3-Linear Maps

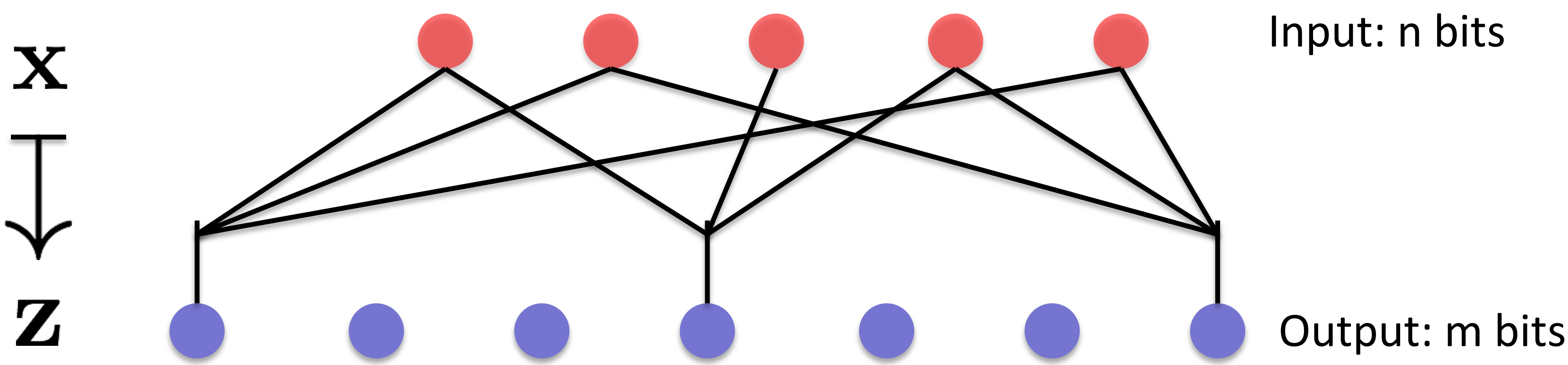
Local Pseudorandom Generators
[Goldreich'00]

n: input length (in bits)
m: output length (in bits)
L: locality
$G_{H,P}$:the PRG

Specified by:

a) a sequence of m L-tuples $H_1,\ldots,H_m$ and

b) a predicate P: $\{0,1\}^L \to \{0,1\}$.



$$G_{H,P}(x) = P(x_{H_1}), P(x_{H_2}), \ldots, P(x_{H_m})$$

$$\text{Security: } G(U_n) \approx_c U_m$$

Token-based Obf: From NC0 to P

Lemma: If there exists a TBO for degree-L functions and there exists a locality-L PRG, then TBO for P (and thus, IO) exists.

"Proof": Similar to bootstrapping obfuscation

✓ Use Randomized encodings for P.
[Applebaum-Ishai-Kushilevitz'00, Yao'86]

✓ **No need for a PRF**. Instead, use a local PRG

✓ Benefit: Can start from TBO for NC0 (instead of NC1).

CONSTRUCTION OUTLINE

IO for P

[Bitansky-V15, Ananth-Jain'15,
Ananth-Jain-Sahai'16, Lin-Pass-Seth-Telang'16]

Token-based Obfuscation for P

[Lin16, Lin-V16, Lin17, Ananth-Sahai'17,
Lin-Tessaro'17]

+ Local PRG

Token-based Obfuscation for NC^0

[Lin16, Lin-V16, Lin17, Ananth-Sahai'17,
Lin-Tessaro'17]

3-Linear Maps

Token-Based Obfuscation for NC0: A Caricature

Lemma: For any constant L, there exists a TBO for degree-L functions (in particular, NC0) assuming **L-linear maps exist.**

Sketch:

✓ Obfuscation of x = $(x_1, \ldots, x_n)$ is $(g^{x_1}, \ldots, g^{x_n})$

✓ Given secret key, want to compute degree-L functions "in the exponent".

✓ Prior works show that O(L)-linear maps are *sufficient*.

✓ Lin-Tessaro show that L-linear maps are *sufficient*.

(L,q)-Blockwise Local PRGs

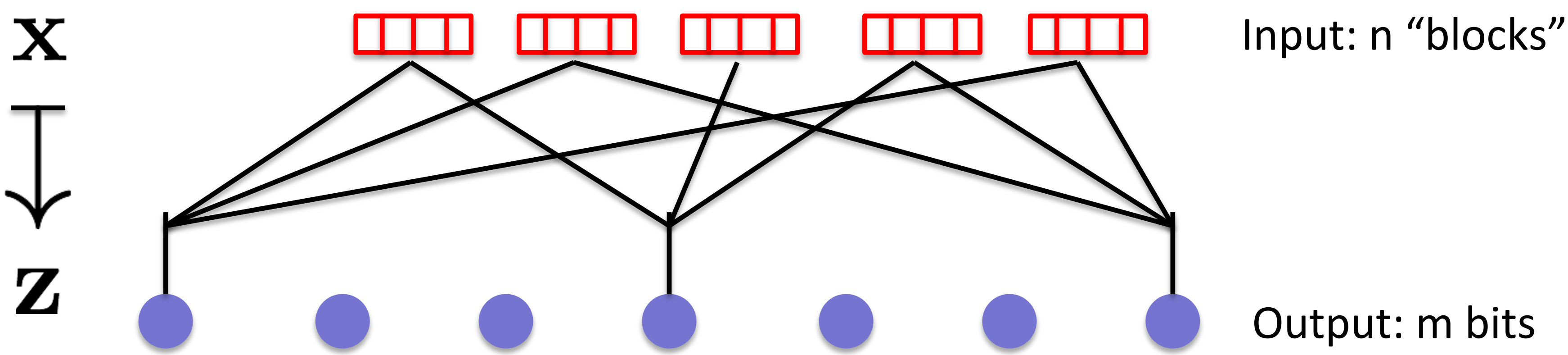
[Lin-Tessaro'17]

Specified by:

a) a sequence of m L-tuples $H_1,\ldots,H_m$ and

b) a predicate P: $[q]^L \to \{0,1\}$.

n: input length (in blocks)
m: output length (in bits)
L: locality
q: alphabet size
$G_{H,P}$: the PRG



$$G_{H,P}(x) = P(x_{H_1}), P(x_{H_2}), \ldots, P(x_{H_m})$$

**(We could additionally have different predicates P_i for each output bit. We focus on the single predicate case in this talk.)

Generalizing: The Lin-Tessaro Theorem

Theorem (informal): There exists an IO scheme, assuming:

a) **L-linear maps (**with the SXDH assumption); and

b) **Blockwise-Locality L PRGs** with polynomial stretch (and subexponential security)

[Lin and Tessaro, CRYPTO 2017]

Case L = 3: There exists an IO scheme, assuming:

a) **3-linear maps**; and

b) "Blockwise 3-local" PRGs expanding n blocks to $\tilde{\Omega}(n^{1+\epsilon})$ bits ✓

with sub-exponential security.

Generalizing: The Lin-Tessaro Theorem

Th… ormal): There exists an IO scheme, assuming:

…ap… the SXDH assumption); and

…Locality …RGs with polynomial stretch (and subexponential security)

[Lin and Tessaro, CRYPTO 2017]

?

Case L = 2: There exists an IO scheme, assuming:

a) **Bilinear maps**; and ✓

b) “(2,q)-blockwise local” PRGs expanding n blocks to $\tilde{\Omega}(nq^{3+\epsilon})$ bits

with sub-exponential security.

Case L = 2: There exists an IO scheme, assuming:

a) **Bilinear maps** with the SXDH assumption; and ✓

b) "(2,q)-blockwise local" PRGs expanding n blocks to $\tilde{\Omega}(nq^{3+\epsilon})$ bits

with sub-exponential security.

Polynomial Time Attacks on Blockwise 2-local PRGs

[Lombardi-V'17, Barak-Brakerski-Komargodski-Kothari'17]

Therefore, the [LT17] construction gets stuck at 3-linear maps.

TUTORIAL OUTLINE

Part 1. DEFINITIONS of program obfuscation

a. Virtual Black-Box OBF

b. Indistinguishability OBF (IO)

Part 2. APPLICATIONS of IO

a. Crypto Applications

b. A Complexity Application

c. Bootstrapping Theorems

Part 3. CONSTRUCTIONS of IO from simpler objects

Theorem: If 3-linear maps exist, so does IO.

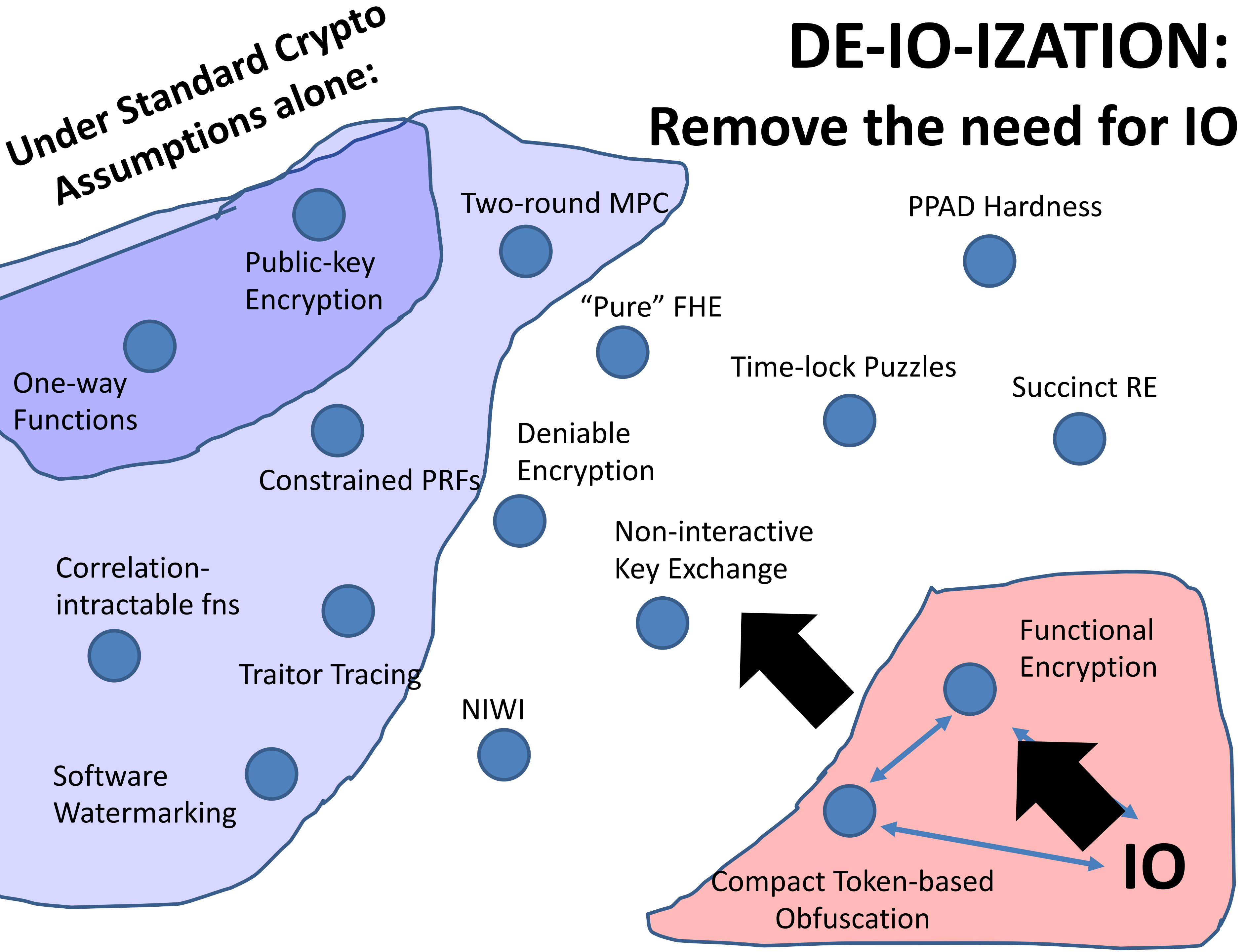
Part 4. DE-IO-IZATION

Remove the need for IO in applications.

e.g., Traitor Tracing (on Wed)

DE-IO-IZATION:

Remove the need for IO



"IO-Inspired" Results

IO-based Constructions teach us new techniques.

(quite often, non-black-box techniques)

- (Anonymous) ID-based Encryption from 1-linear maps.

  (Previously, required 2-linear maps.)

  [Garg-Dottling'17, '18, Brakerski-Lombardi-Segev-V.'18]

- 2-round Multiparty Computation from OT.

  (Previously, required IO or learning with errors.)

  [Garg-Srinivasan'18, Benhamouda-Lin'18]

SUMMARY

Part 1. DEFINITIONS of program obfuscation

a. Virtual Black-Box OBF

b. <u>Indistinguishability OBF (IO)</u>

Part 2. APPLICATIONS of IO

a. Crypto Applications

b. A Complexity Application

c. Bootstrapping Theorems

Part 3. CONSTRUCTIONS of IO from simpler objects

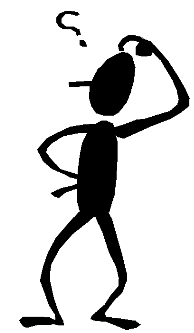
Theorem: If 3-linear maps exist, so does IO.

Part 4. DE-IO-IZATION

Remove the need for IO in applications.

e.g., Traitor Tracing (on Wed)

Thank you!



The Quest Continues…

MANY OTHER RESULTS:

Obfuscating simple programs

Obfuscation with the aid of secure hardware

Achieving applications without obfuscation

Functional Encryption

[Sahai-Waters'05, Boneh-Sahai-Waters'12]

Given *encryption* of string x

and *secret key* for function f

Thou shalt be able to compute *f(x),*

but nothing else.

P.S.: the size of Enc(x) should be $O_\lambda(|x|)$.

From NC0 to NC1 (Lemma 2)

Lemma 2: If there exists a functional encryption for degree-L functions and there exists a locality-L PRG, then functional encryption for NC1 (and thus, IO) exists.

"Proof":

- ✓ Use AIK Randomized encodings for NC1.
  [Applebaum-Ishai-Kushilevitz'04]
- ✓ **AIK Principle**: Instead of computing a complex function $F(x)$, compute a simpler randomized function $\hat{F}(x, r)$. ($\hat{F}$ is in NC0).
- ✓ Problem: $|r|$ proportional to the circuit size of F and $\gg |x|$.
- ✓ Solution: use local PRG to generate r. $\square$

Connection between Local PRGs and IO

[Lin'16, Lin-**V**'16, Lin'17, Ananth-Sahai'17]

Theorem: There exists an IO scheme, assuming:

a) **L-linear maps** with the SXDH assumption

b) **Locality L PRGs** with *any* polynomial stretch (and subexponential security)

c) **Subexponentially secure Learning with Errors** (ignored from now on)

[Lin, CRYPTO 2017]

?

Connection between Local PRGs and IO

[Lin'16, Lin-V'16, Lin'17, Ananth-Sahai'17]

Lin's Theorem = Lemma 1 + Lemma 2

Lemma 1: For any constant L, there exists a functional encryption for degree-L functions (in particular, NC0) assuming **L-linear maps exist.**

Lemma 2: If there exists a functional encryption for degree-L functions and there exists a locality-L PRG, then functional encryption for NC1 (and thus, IO) exists.