APPLICATIONS OF (INDISTINGUISHABILITY) OBFUSCATION

Craig Gentry, IBM Research

May 20, 2015

Cryptography Boot Camp, Simons Institute

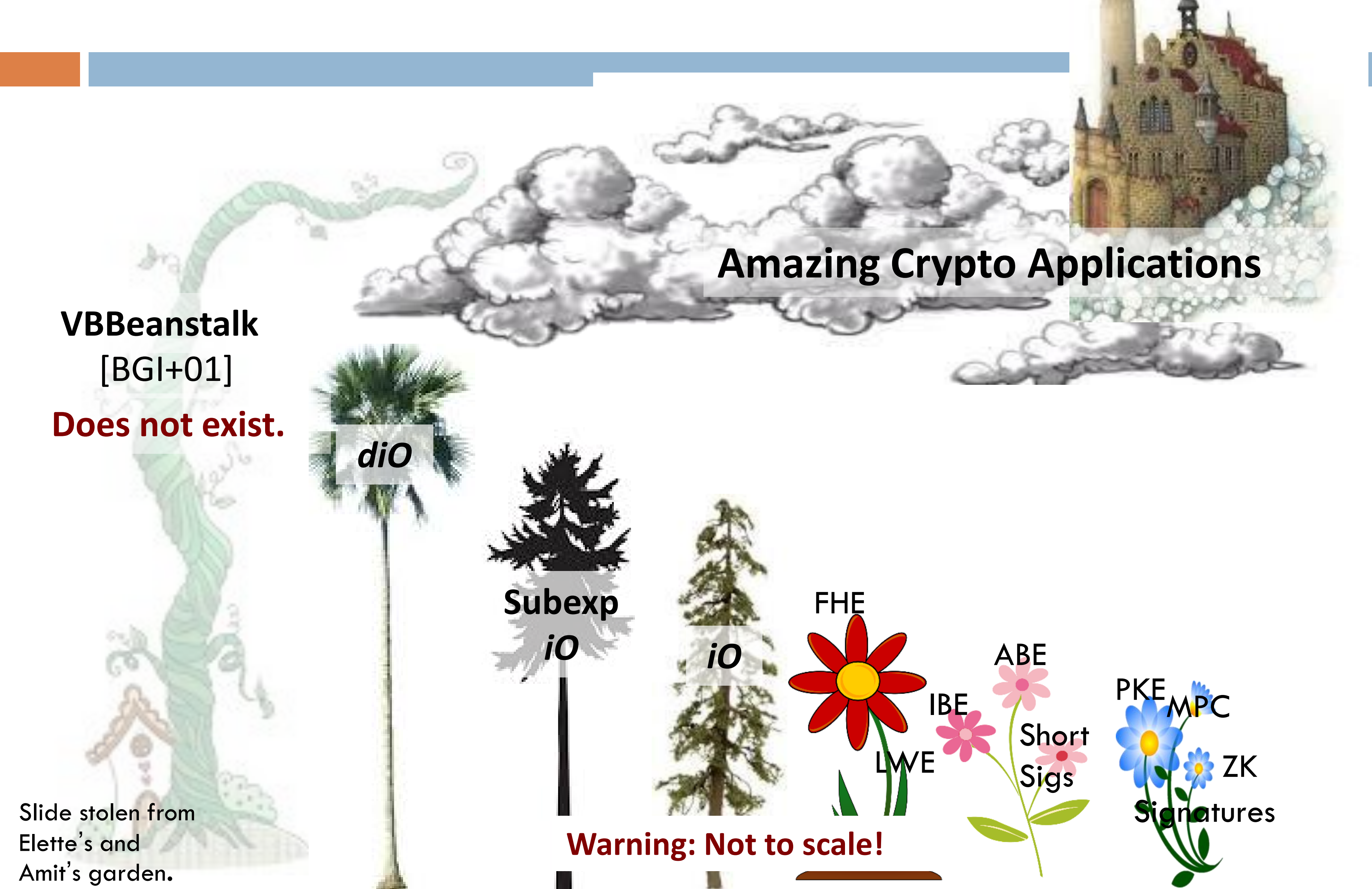
Definition of iO [B+01]

- An indistinguishability obfuscator is a PPT algorithm iO that takes a program P as input and is:
  - Efficient: Description/runtime of iO(P) are poly-related to P.
  - Functionality-Preserving: The string iO(P) describes a program with the same input-output behavior as P.
  - Pseudo-Canonicalizing: For any PPT adversary A and any programs P_1 and P_2 of equal complexity and functionality:

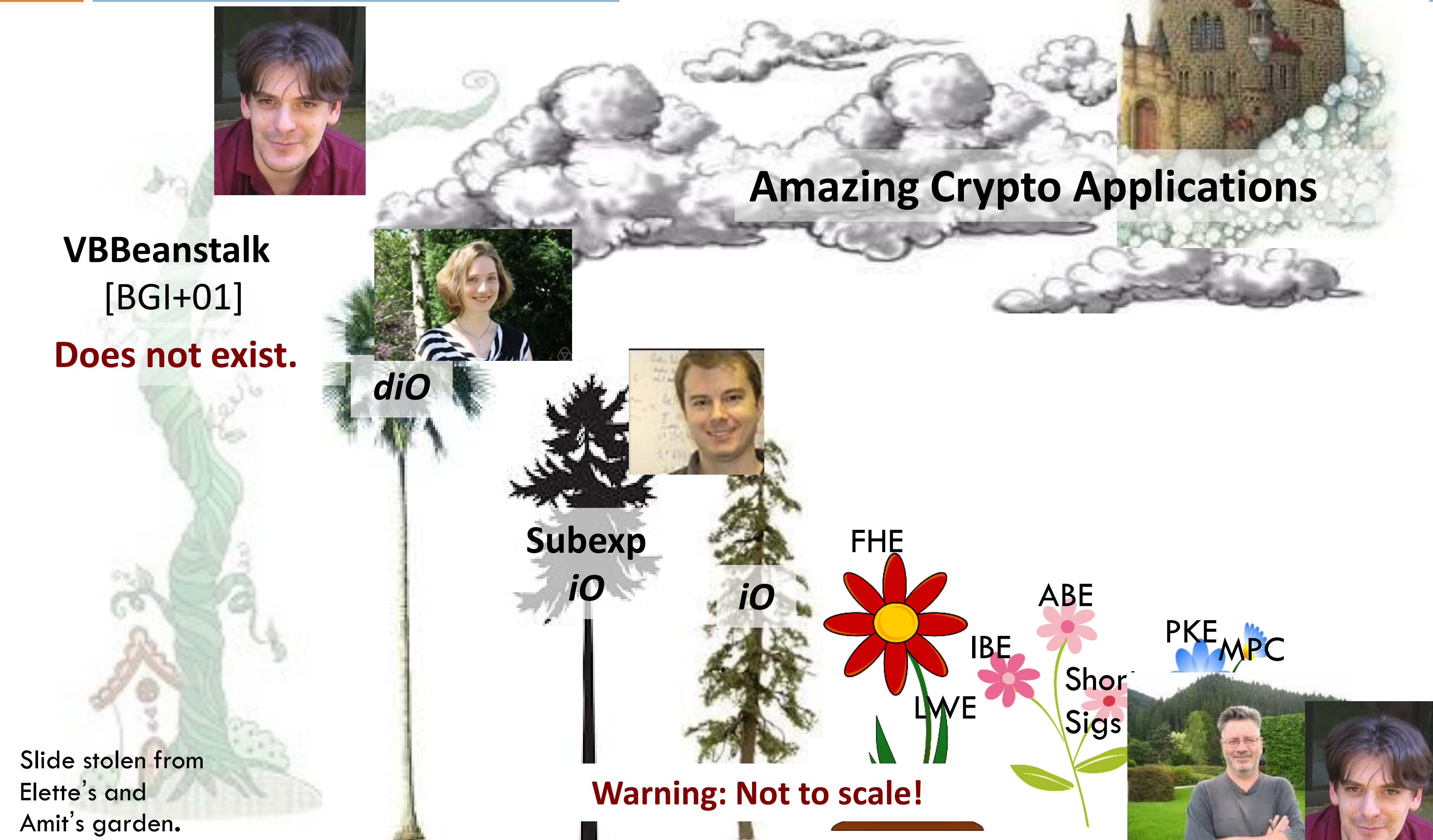
    $$\left| \Pr[A(iO(P_1))=1] - \Pr[A(iO(P_2))=1] \right| \text{ is negligible.}$$

  - In English: If two programs have same input-output behavior, the adversary cannot distinguish which was obfuscated.
- Circuits: Usually our model of computation.

Plant of the Talk



Plant of the Talk



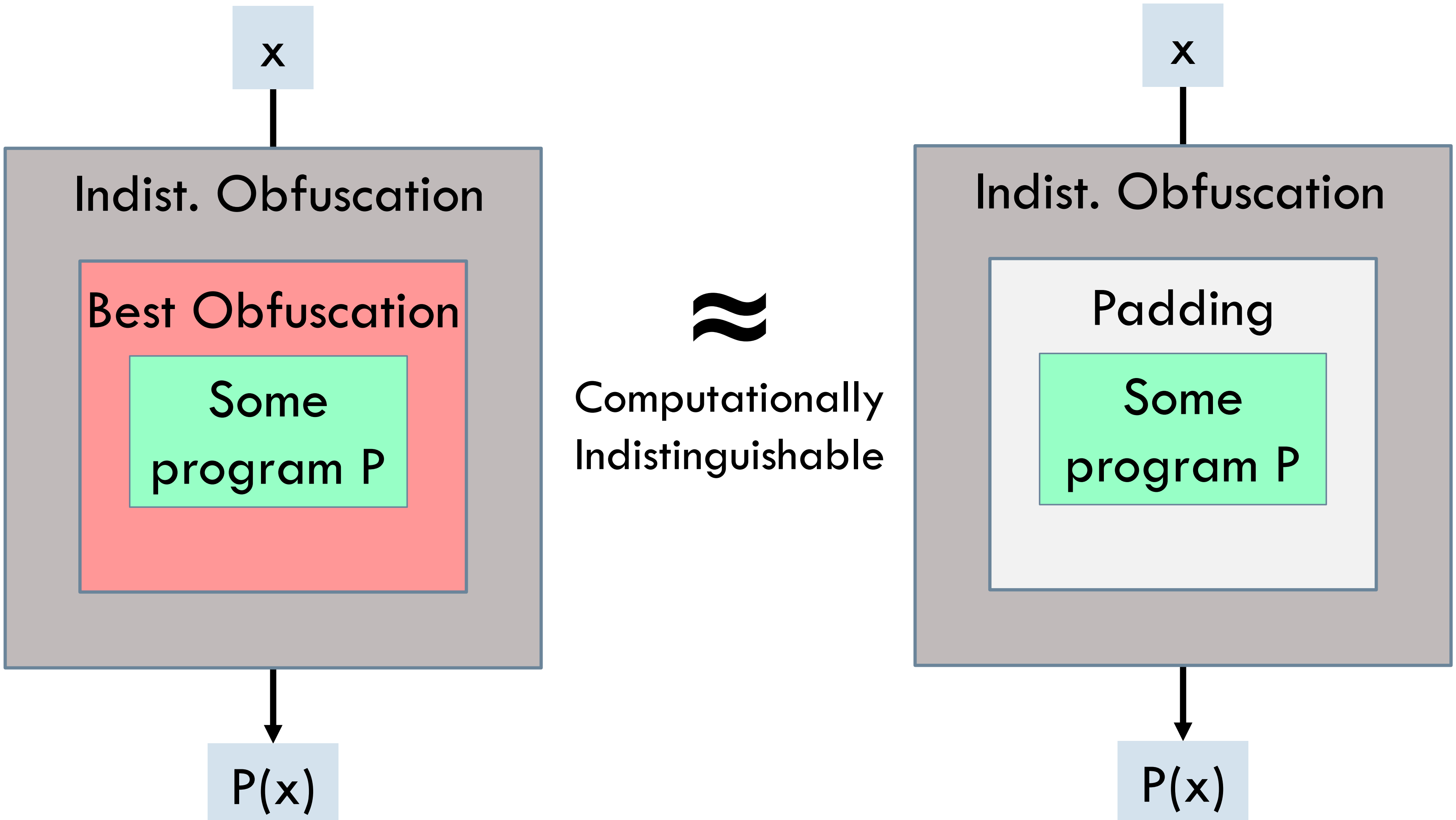
Slide stolen from Elette's and Amit's garden.

Simple Applications of iO

Best-Possible Obfuscation

An indistinguishability obfuscator is “as good" as any other obfuscator that exists. [GR07]

Best-Possible Obfuscation

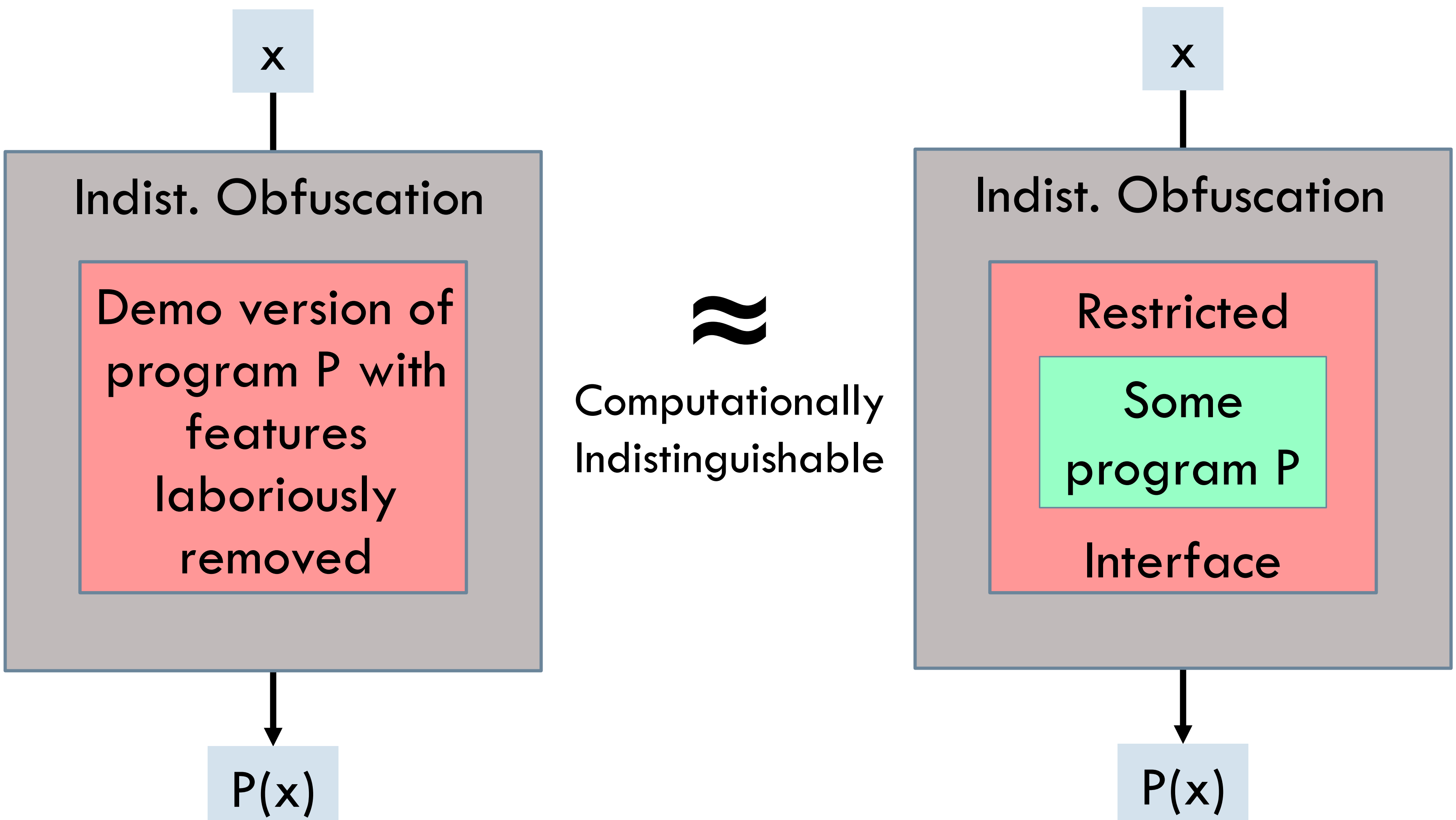


Restricted-Use Software

- Setting: Software developer wants to:
  - Publish demo version with features removed
  - Construct multiple tiers of product at different prices
  - Give an untrusted partner a “dumbed-down” version that only works for relevant tasks

- The problem: Removing features is difficult.
  - Laborious, introduces bugs
  - End product may still reveal more than intended

Restricted-Use Software from iO



Witness Encryption [R89,GK05,GGSW13]

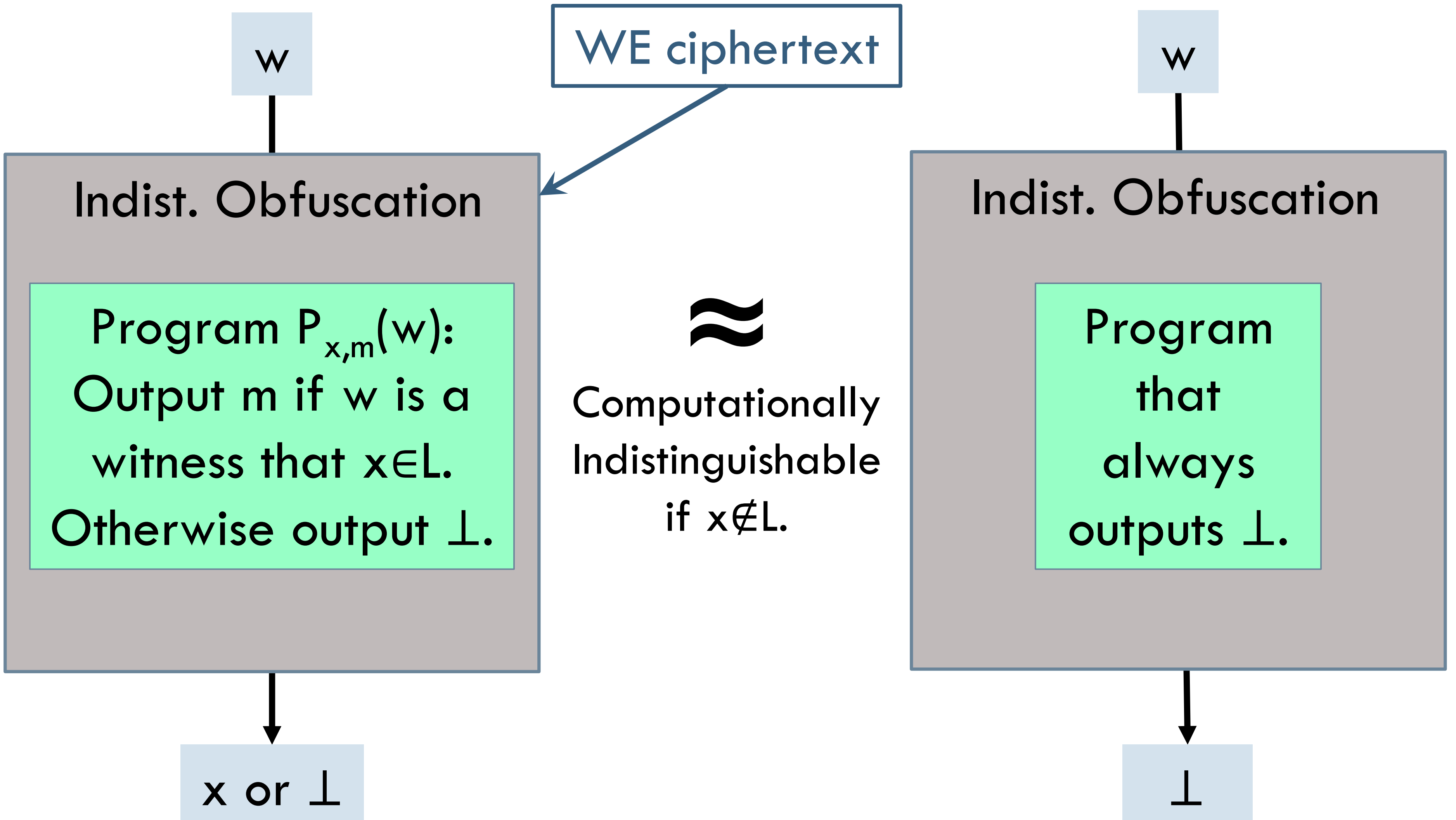
- Goal: Encrypt m so only someone with proof of Riemann Hypothesis can decrypt.

- Procedures:
  - Encryption: c $\leftarrow$ WEnc(x;m) encrypts m relative to statement x.
  - Decryption: $\{m,\perp\} \leftarrow$ WDec(w;c) works if w is a witness for $x \in L$.

- Secret key?: No "secret key" per se.

- Security: $\mathrm{WEnc}(x;m_0) \approx \mathrm{WEnc}(x;m_1)$ when $x \notin L$.

Witness Encryption from iO [GGHRSW13]



Relative vs. Absolute Guarantees

- Apps above have weak "relative" security guarantees:
  - BPO: Obfuscation is as good as best-possible obfuscation
  - Restricted-use software: As good as restricted interface.
  - WE: No guarantees when $x \in L$.

- How to get absolute guarantees?
  - Make an absolute assumption – e.g., existence of OWFs.
  - But surely iO already implies OWFs…?

iO and OWFs

- iO → OWFs?
  - No, if P = NP.
  - iO exists if P = NP: Obfuscate program by outputting lexicographically first program with same functionality.
- iO → OWFs if NP $\not\subseteq$ BPP [KMNPRY14].
  - Candidate OWF: f(x) = iO(Z;x) where Z is unsatisfiable.
  - Replace challenge with y_1 = iO(C_1;x_1) for unsatisfiable C_1. By iO, adversary cannot distinguish, and will still invert.
  - Replace challenge with y_2 = iO(C_2;x_2) for satisfiable C_2. Adversary cannot invert, since $\nexists$x such that f(x) = iO(C_2;x_2).
  - Adversary's success/failure tells us whether C is satisfiable.

Simple App: WE+OWF → PKE [GGSW13]

Super-fast KeyGen!

- KeyGen: PRG : $\{0,1\}^n \to \{0,1\}^{2n}$.
  - Secret key: $s^* \in \{0,1\}^n$. Public key: $t^* = \mathrm{PRG}(s^*)$.
- Encryption:
  - Let x_{t^*} be the statement "$\exists$ s such that $t^* = \mathrm{PRG}(s)$".
  - $c \leftarrow \mathrm{WEnc}(x_{t^*};m)$.
- Decryption: $m \leftarrow \mathrm{WDec}(s^*;c)$.

Hyb_0 (real world): $c = \mathrm{WEnc}(x_{t^*} ; m_0)$.

PRG security

Hyb_1 ($t \leftarrow \{0,1\}^{2n}$): $c = \mathrm{WEnc}(x_t ; m_0)$.

WE security

Hyb_2 ($m_0 \to m_1$): $c = \mathrm{WEnc}(x_t ; m_1)$.

PRG security

Hyb_3 (real world): $c = \mathrm{WEnc}(x_{t^*} ; m_1)$.

Other Apps of WE (+ Simple Primitives)

- Identity-based encryption
- Attribute-based encryption for circuits
- Secret sharing for monotone NP access structures [KNY14]
- …

Hiding Secrets in Software with iO

Two main techniques:

1. Shell games with secrets

2. Punctured programs

1st Technique:
Shell Games with Secrets

Shell Games with Secrets

Thought Experiment

Functionally Equivalent

Indist. Obfuscation

Program P_1 uses secret sk_1 and does not contain sk_2.

$\approx$

Computationally Indistinguishable

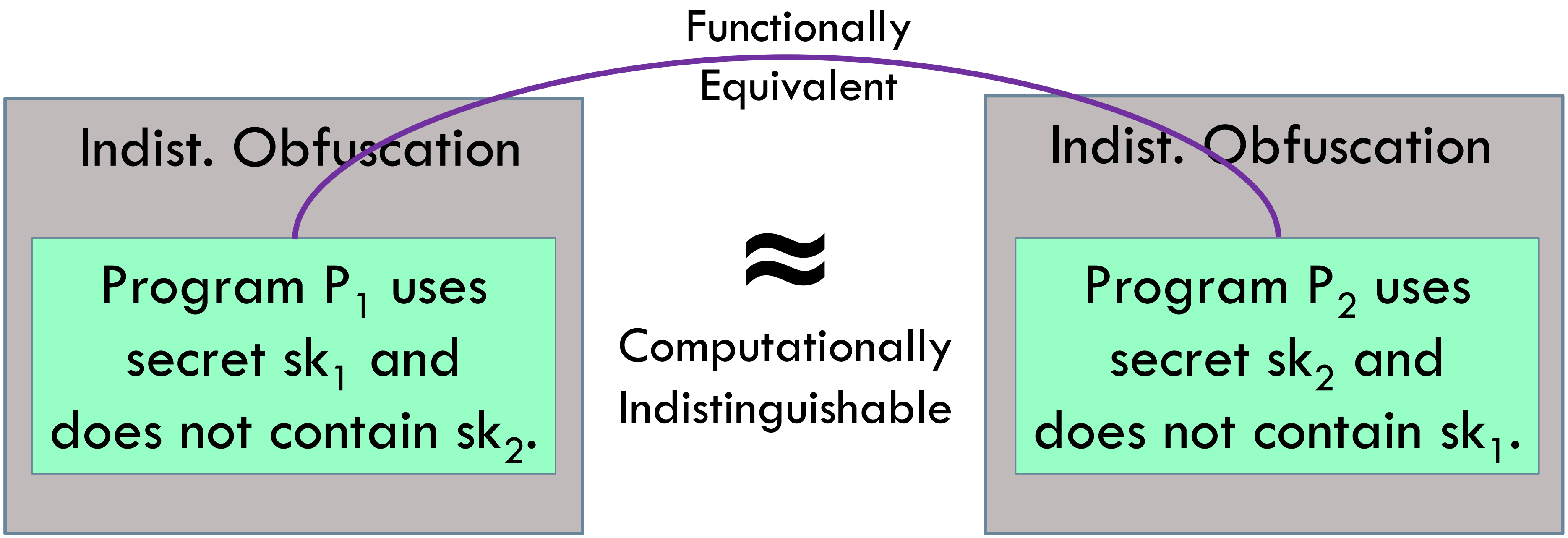
Indist. Obfuscation

Program P_2 uses secret sk_2 and does not contain sk_1.

Does either obfuscation reveal sk_1 or sk_2?

Shell Games with Secrets

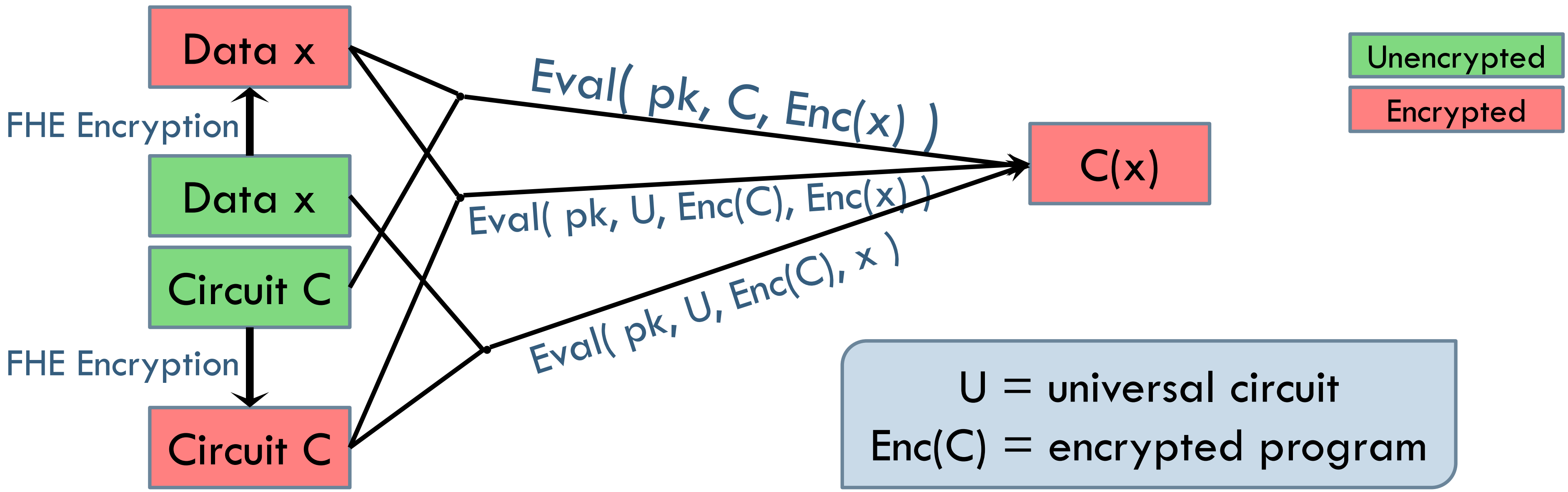
Thought Experiment



- P_1 hides sk_2 and P_2 hides sk_1.
- But $iO(P_1) \approx iO(P_2)$.
- So, $iO(P_1)$ hides sk_1 and $iO(P_2)$ hides sk_2.
- "Two-key technique" used many times before ([NY90], …).

Shell Game Application: iO for Circuits from (iO for NC^1) + (FHE with decryption in NC^1)

Reminder about FHE [RAD78, Gen09, …]



U = universal circuit
Enc(C) = encrypted program

Current FHE schemes have decryption procedures that can be computed by shallow (NC^1) circuits.

Shell Game Application: iO for Circuits from (iO for NC^1) + (FHE with decryption in NC^1)

[GGHRSW13]

Obfuscation of General Circuit C

Circuit C encrypted under FHE key pk_1

Circuit C encrypted under FHE key pk_2

Input x, two ciphertexts c_1, c_2, and proof π

iO for NC^1

Program CondDec:

Hardwired: sk_1 and $\{e_i = \mathrm{Enc}(pk_i, c)\}$

Verify proof π that

$c_1 = \mathrm{Eval}(pk_1, U, e_1, x)$,

$c_2 = \mathrm{Eval}(pk_2, U, e_2, x)$.

If true, output $\mathrm{Dec}(sk_1, c_1)$; else output $\perp$.

C(x) or $\perp$

Shell Game Application: iO for Circuits from (iO for NC^1) + (FHE with decryption in NC^1)

x, c_1, c_2, π

iO for NC^1

<u>CondDec$_{sk1}$:</u>

Verify proof π that

$c_1 = \mathrm{Eval}(pk_1, U, e_1, x)$,

$c_2 = \mathrm{Eval}(pk_2, U, e_2, x)$.

If true, output $\mathrm{Dec}(sk_1, c_1)$.

Else output $\perp$.

$C(x)$ or $\perp$

$\approx$

x, c_1, c_2, π

iO for NC^1

<u>CondDec$_{sk2}$:</u>

Verify proof π that

$c_1 = \mathrm{Eval}(pk_1, U, e_1, x)$,

$c_2 = \mathrm{Eval}(pk_2, U, e_2, x)$.

If true, output $\mathrm{Dec}(sk_2, c_2)$.

Else output $\perp$.

$C(x)$ or $\perp$

- $\mathrm{CondDec}_{sk1}$ and $\mathrm{CondDec}_{sk2}$ have same input-output behavior.
- So, their obfuscations are indistinguishable, and hide sk_1 and sk_2.

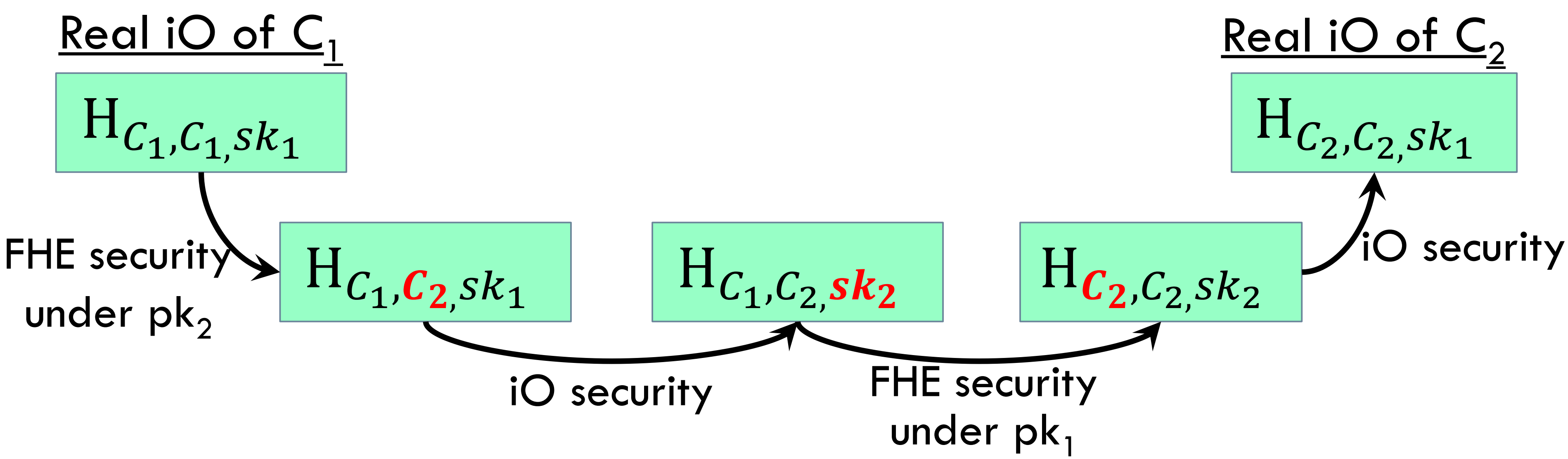
Shell Game Application: iO for Circuits

from (iO for NC^1) + (FHE with decryption in NC^1)

Security Proof for iO Scheme:

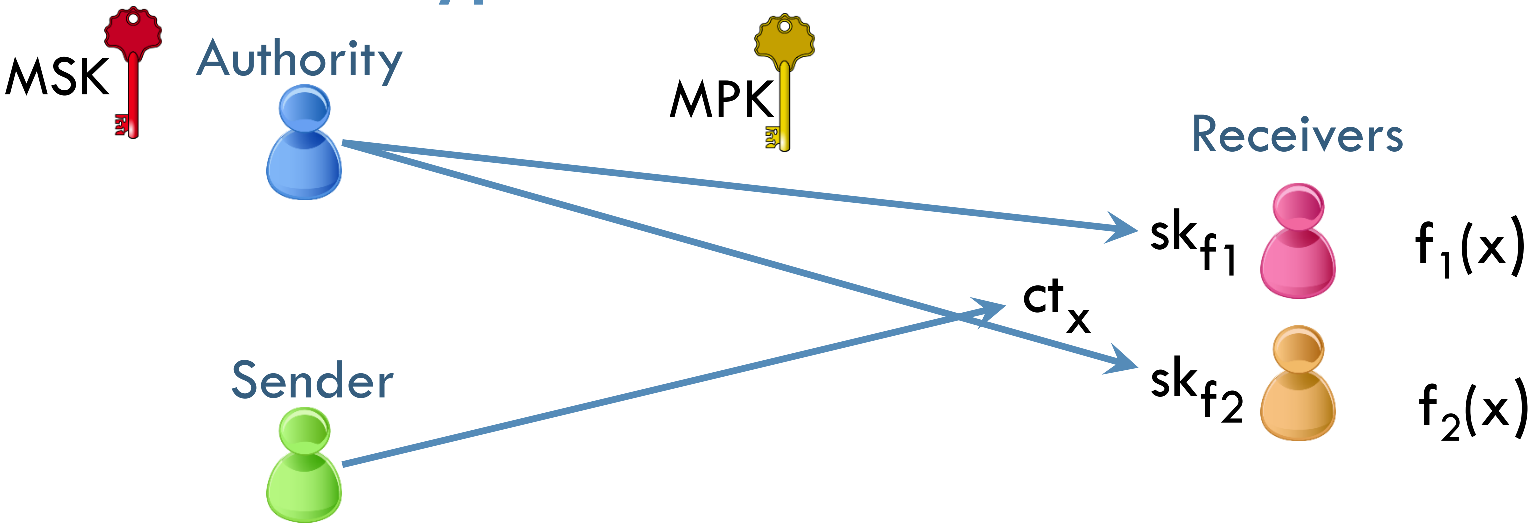
- Suppose circuits C_1, C_2 have same functionality.
- Hybrids: In $\mathrm{H}_{C_{b_1},C_{b_2},sk_{b_3}}$ the obfuscation consists of:

  $e_1 = \mathrm{Enc}(pk_1, C_{b_1})$, $e_2 = \mathrm{Enc}(pk_2, C_{b_2})$, $iO(\mathrm{CondDec}_{sk_{b_3}})$



Shell Game Application: Functional Encryption for Circuits

Functional Encryption [S84, SW05, BSW11, …]

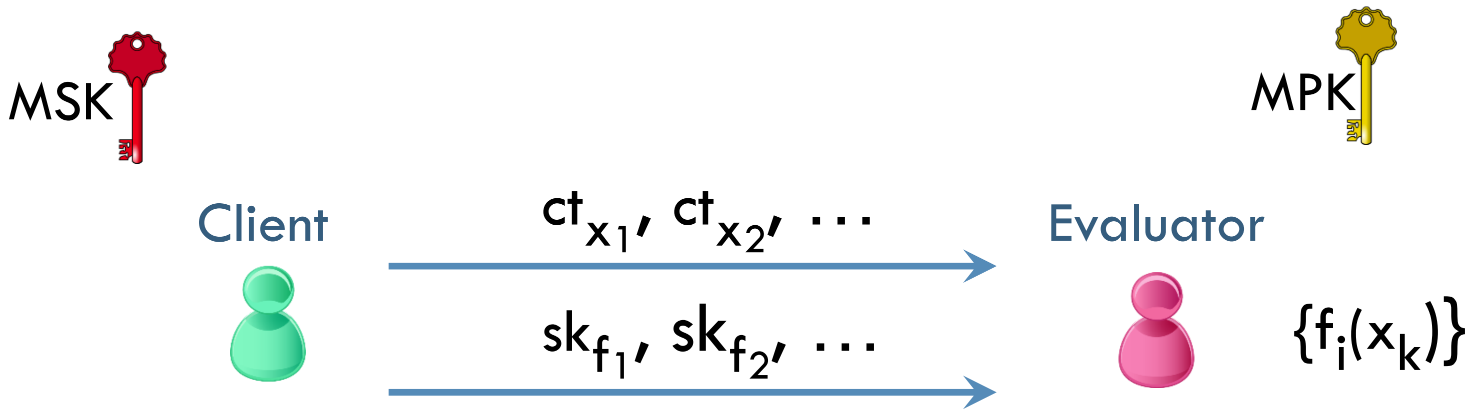


Syntax

- (MPK, MSK) $\leftarrow$ FE.Setup(1^λ)
- $sk_f \leftarrow$ FE.KeyGen(MSK, f)
- $ct_x \leftarrow$ FE.Enc(MPK, x)
- $f(x) \leftarrow$ FE.Dec(sk_f, ct_x)

Shell Game Application: Functional Encryption for Circuits

Functional Encryption [S84, SW05, BSW11, …]



Syntax

- (MPK, MSK) ← FE.Setup(1^λ)
- sk_f ← FE.KeyGen(MSK, f)
- ct_x ← FE.Enc(MPK, x)
- f(x) ← FE.Dec(sk_f , ct_x)

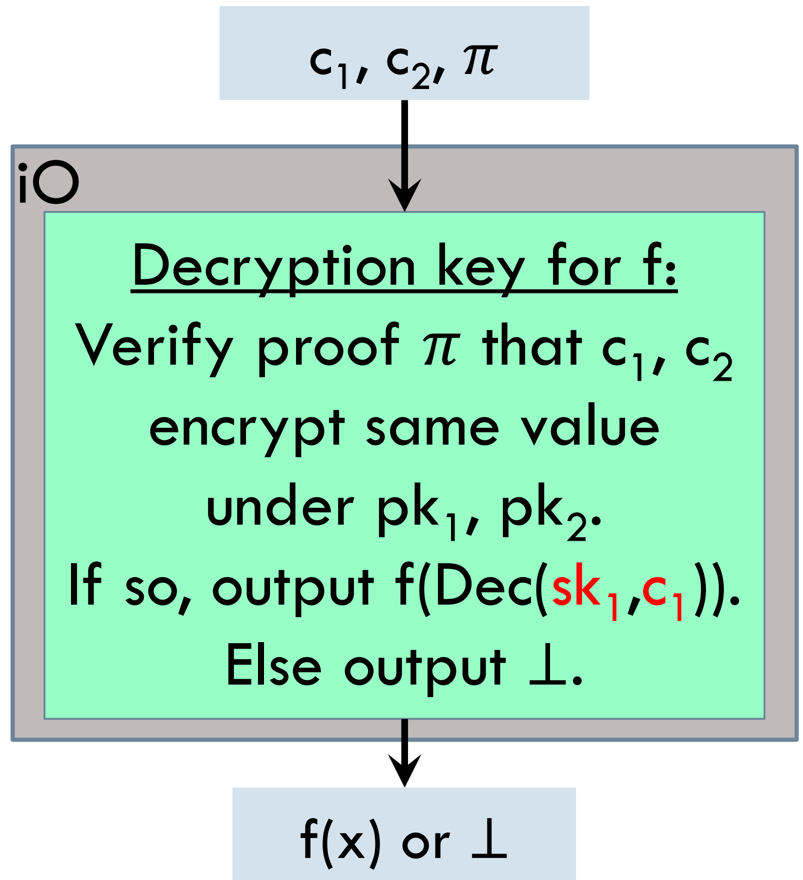
(Selective) Security Game

- Adversary selects x_1, x_2.
- Challenger sends MPK, challenge ct.
- Key queries: for f_i such that $f_i(x_1) = f_i(x_2)$.
- Adversary guesses.

Shell Game Application: Functional Encryption for Circuits

Functional Encryption from IO [GGHRSW13]

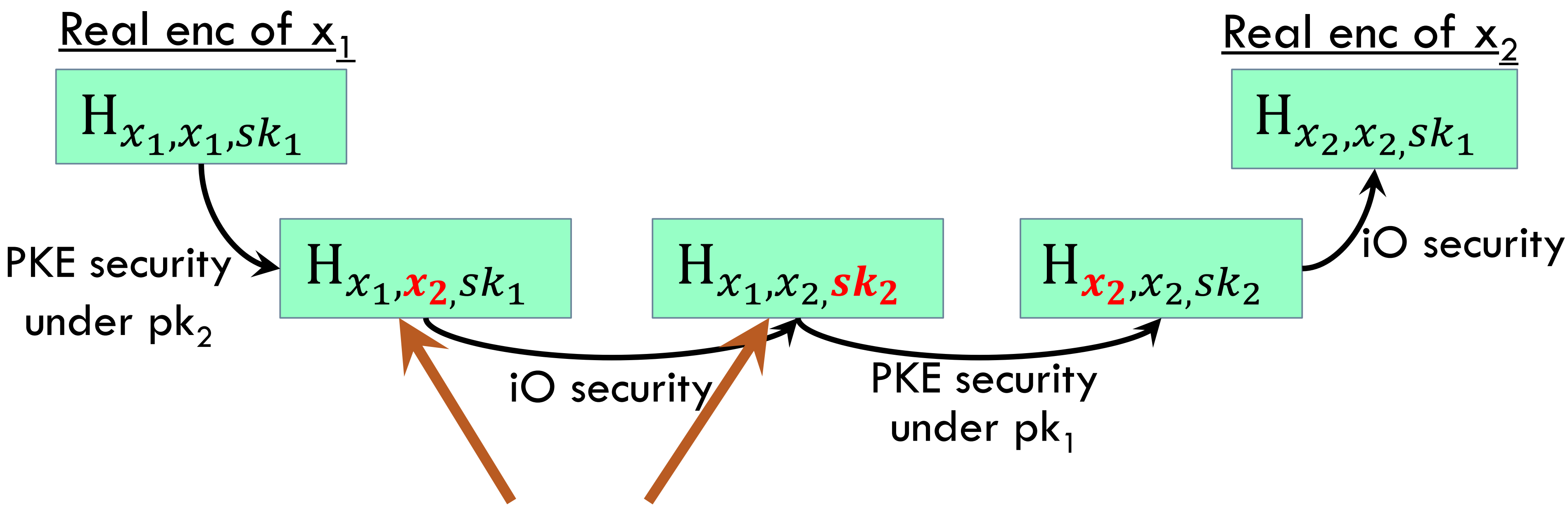
- FE.Setup: Generate:
  - PKE key-pairs (pk_1, sk_1), (pk_2, sk_2).
  - CRS for stat. sim. sound NIZK proof.
- FE.Enc(MPK,x): Generate:
  - $c_1 \leftarrow Enc(pk_1, x; r_1)$,
  - $c_2 \leftarrow Enc(pk_2, x; r_2)$,
  - NIZK proof π that c_1, c_2 encrypt same value: that $\exists\ r_1, r_2$ s.t. $c_1=Enc(pk_1,x;r_1)$, $c_2=Enc(pk_2,x;r_2)$.



Shell Game Application: Functional Encryption for Circuits

Security Proof for FE Scheme:

- Hybrids: In $\mathrm{H}_{x_{b_1}, x_{b_2}, sk_{b_3}}$ the challenge ciphertext is $c_1^*,\ c_2^*, \pi^*$, where c_1^* encrypts $x_{b_1},\ c_2^*$ encrypts x_{b_2}, and user keys decrypt under sk_{b_3}.

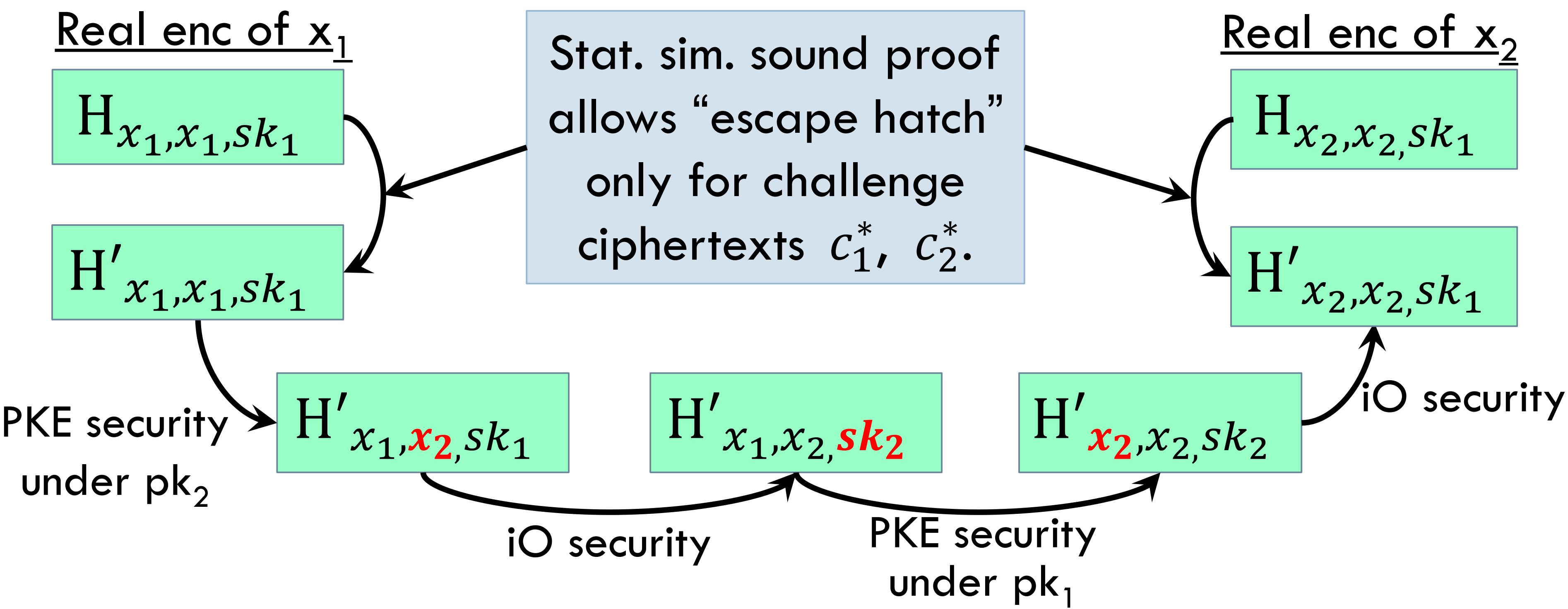


Oops!! NIZK proof cannot work in these hybrids!!!

Shell Game Application: Functional Encryption for Circuits

Security Proof for FE Scheme:

- Hybrids: In $\mathrm{H}_{x_{b_1},x_{b_2},sk_{b_3}}$ the challenge ciphertext is $c_1^*,\ c_2^*, \pi^*$, where c_1^* encrypts x_{b_1}, c_2^* encrypts x_{b_2}, and user keys decrypt under sk_{b_3}.



2nd Technique: Punctured Programs

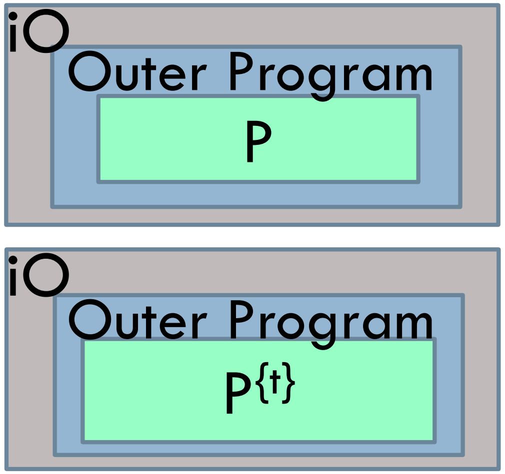
Punctured Programming [SW13]

Definition: $P^{\{t\}}$ is program P punctured at input t.

$$x \longrightarrow \boxed{P^{\{t\}}} \longrightarrow \begin{cases} P(x) & \text{if } x \neq t \\ \perp & \text{if } x = t \end{cases}$$

Punctured Programming Strategy:

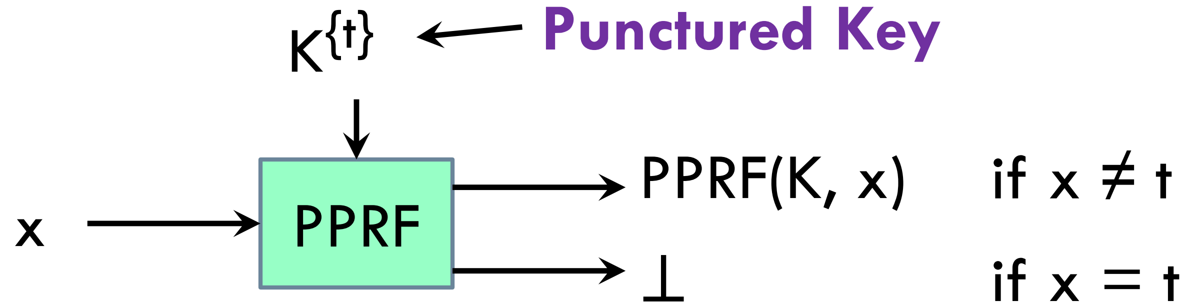
- Show iOs of two programs are indistinguishable.
- Show adversary needs P(t) to win game.
- Show that $P^{\{t\}}$ keeps P(t) secret.



"The idea of the technique is to alter a program (which is to be obfuscated) by surgically removing a key element of the program, without which the adversary cannot win the security game it must play, but in a way that does not alter the functionality of the program."

Punctured PRFs

Definition:



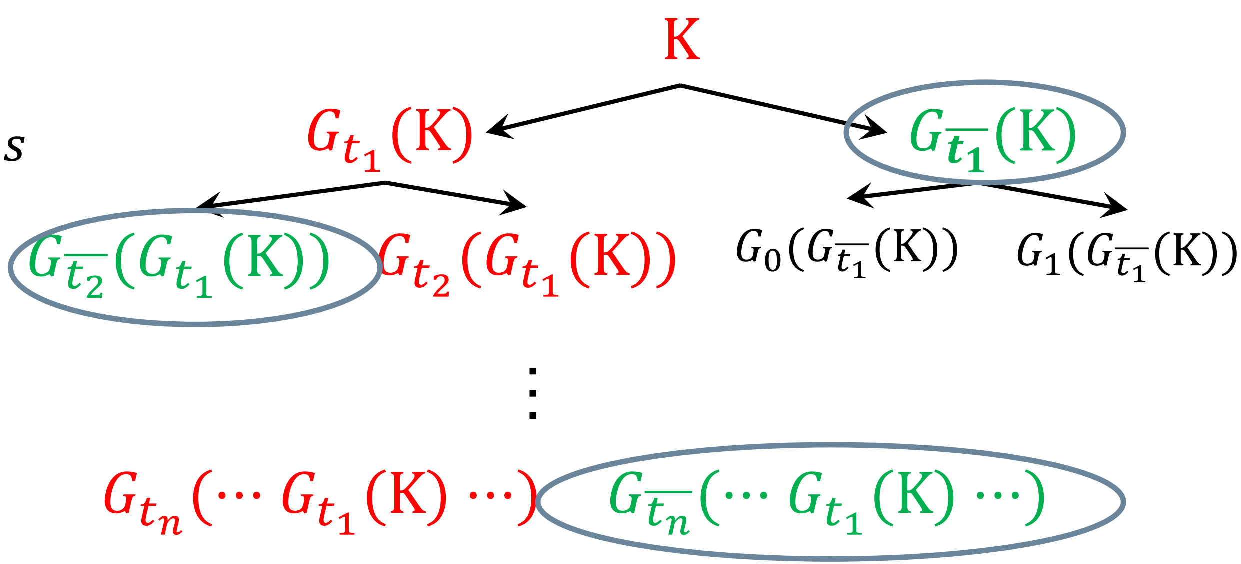
Security: PPRF(K, t) is pseudorandom given $K^{\{t\}}$ and t.

From GGM:

PRG $G : \{0,1\}^s \rightarrow \{0,1\}^{2s}$

$G(r) = G_0(r) \parallel G_1(r)$

$\mathrm{PRF}(K, x)$
$= G_{x_n}(\cdots G_{x_1}(\mathrm{K}) \cdots)$

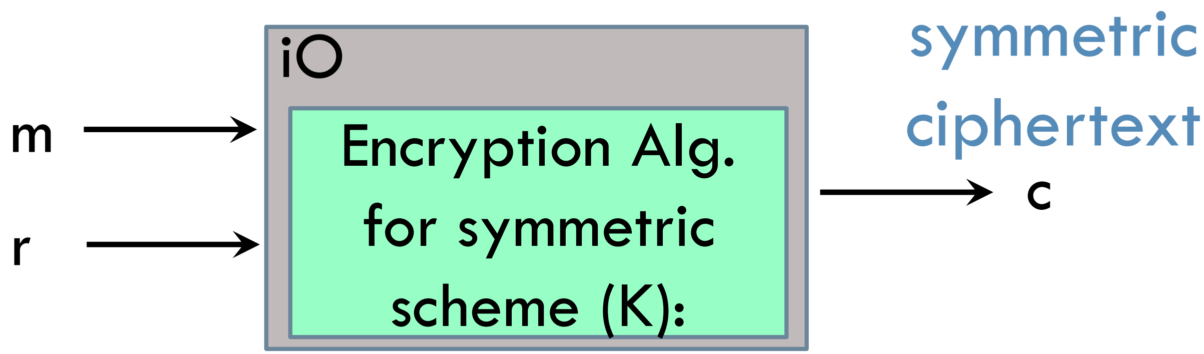


PKE Using Punctured PRFs [SW13]

Secret Key:

Key K for symmetric encryption

Public Key:



Diffie-Hellman '76: Get PKE by obfuscating encryption:

"If the [encryption] program were to be made purposefully confusing through the addition of unneeded variables and statements then determining an inverse algorithm could be made very difficult."

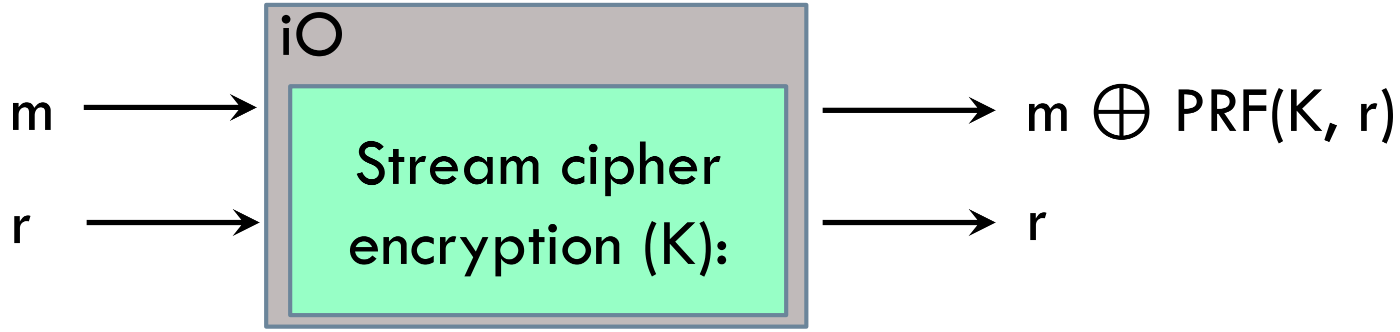
PKE Using Punctured PRFs [SW13]

Naïve Attempt

Secret Key:

Key K for PRF

Public Key:



Problem: Stream cipher encryption is its own inverse!

PKE Using Punctured PRFs [SW13]

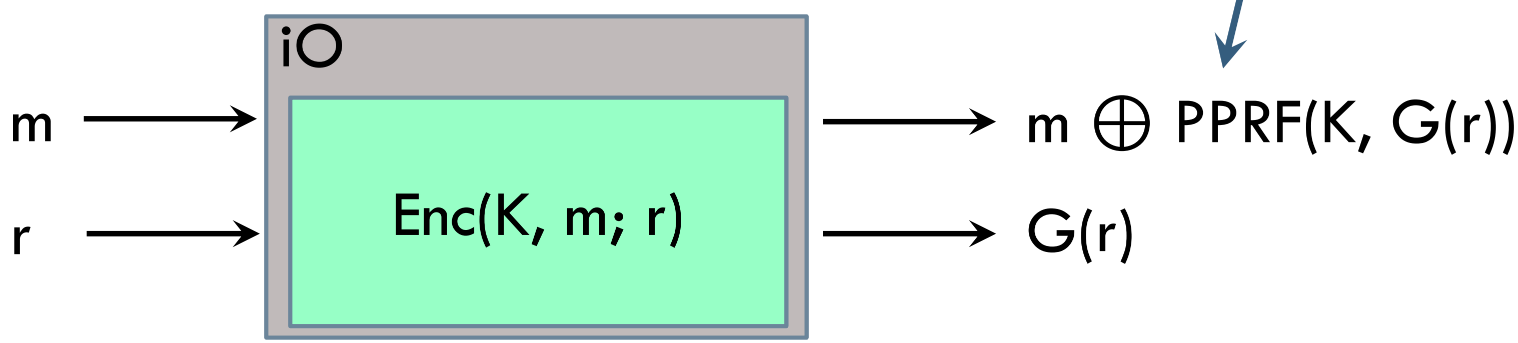
Actual Scheme

Super-fast Decryption!

Secret Key:

Key K for PRF

Public Key: Let $G : \{0,1\}^s \rightarrow \{0,1\}^{2s}$ be a PRG.



Challenge ciphertext: $c^* = (t\ ,\ m \oplus \mathrm{PPRF}(K, t)),\ t = \mathrm{PRG}(r)$

Make t uniform in $\{0,1\}^{2s}$. (PRG security)

PKE Using Punctured PRFs [SW13]

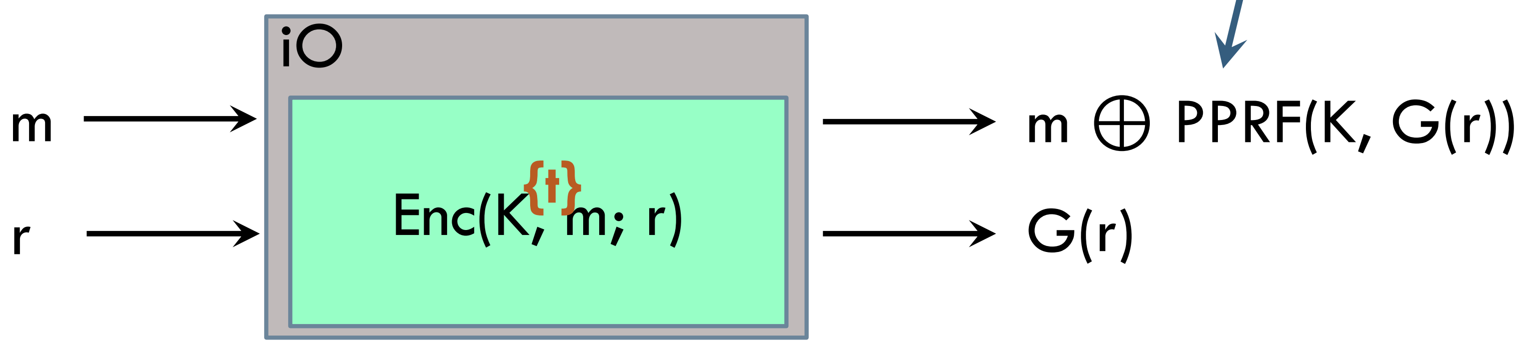
Actual Scheme

Super-fast Decryption!

Secret Key:

Key K for PRF

Public Key: Let $G : \{0,1\}^s \rightarrow \{0,1\}^{2s}$ be a PRG.



Challenge ciphertext: $c^* = (t\ ,\ m \oplus \mathrm{PPRF}(K, t))$, t uniform in $\{0,1\}^{2s}$.

Use $\mathrm{PPRF}(K^{\{t\}}, \cdot)$ instead of $\mathrm{PPRF}(K, \cdot)$ inside Enc.

(iO security, since **t is almost certainly not in range of G.**)

PKE Using Punctured PRFs [SW13]

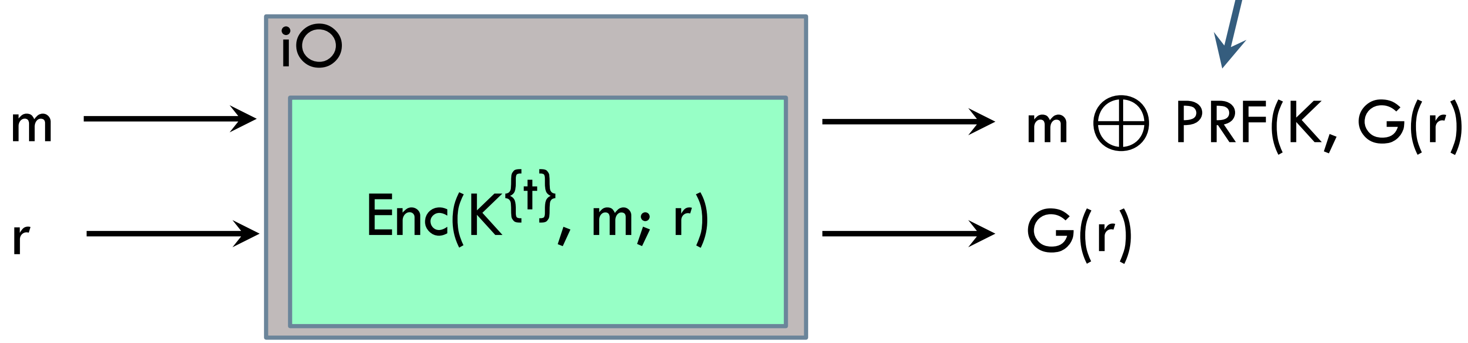
Actual Scheme

Super-fast Decryption!

Secret Key:

Key K for PRF

Public Key: Let $G : \{0,1\}^s \rightarrow \{0,1\}^{2s}$ be a PRG.



Challenge ciphertext: $c^* = (t\ ,\ m \oplus \mathrm{PPRF}(K, t))$, t uniform in $\{0,1\}^{2s}$.

Replace PPRF(K, t) with random value u. (Punctured PRF security)

PKE Using Punctured PRFs [SW13]

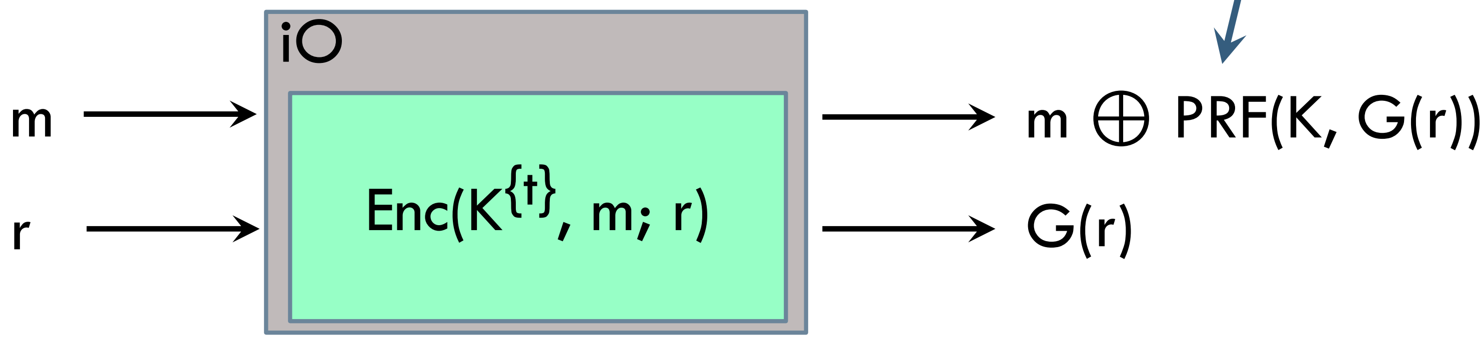
Actual Scheme

Super-fast Decryption!

Secret Key:

Key K for PRF

Public Key: Let $G : \{0,1\}^s \rightarrow \{0,1\}^{2s}$ be a PRG.



Challenge ciphertext: $c^* = (t\ ,\ m \oplus u)$, t uniform, u uniform.

Message is perfectly hidden.

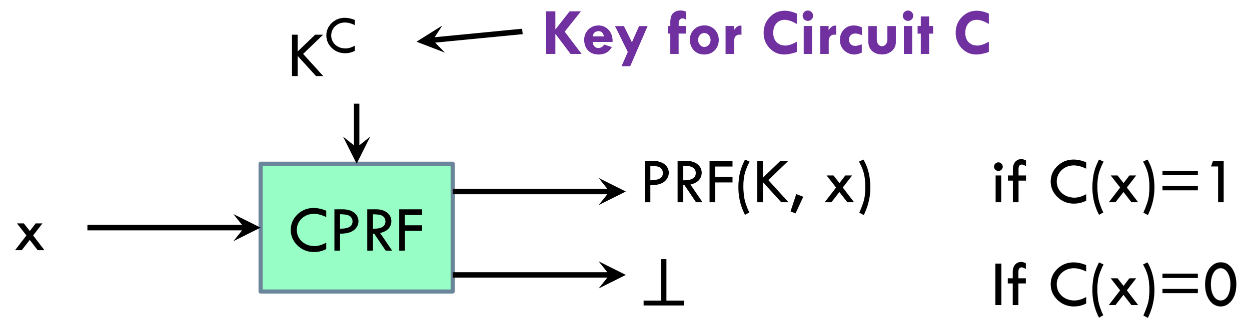
Another Useful Trick: Complexity Leveraging

- Construct "selectively secure" scheme.
  - Adversary forced to pre-commit to input $t \in \{0,1\}^k$ to "attack".
  - Successful attack on t breaks iO or PPRF (or whatever).
  - $\varepsilon_{\text{selective}}(\lambda) \le \varepsilon_{\text{iO}}(\lambda) + \varepsilon_{\text{PPRF}}(\lambda)$.
- Go from selective security to adaptive security
  - Challenger randomly guesses t that adversary will target.
  - Probability that adaptive adversary wins and happens to pick t is $\varepsilon_{\text{adaptive}}(\lambda)/2^k \le \varepsilon_{\text{selective}}(\lambda) \le \varepsilon_{\text{iO}}(\lambda) + \varepsilon_{\text{PPRF}}(\lambda)$.
  - So $\varepsilon_{\text{adaptive}}(\lambda) \le 2^k (\varepsilon_{\text{iO}}(\lambda) + \varepsilon_{\text{PPRF}}(\lambda))$.
  - Choose λ = poly(k) so that $\varepsilon_{\text{adaptive}}(\lambda)$ is negligible.

Security at one input boosted to security at many inputs.

Constrained PRF

Definition:



Security: PRF(K, x) is pseudorandom for all unsatisfying x.

Problem: How can we puncture the key at an exponential number of points?

Constrained PRF

Construction:

PPRF $\{0,1\}^n \rightarrow \{0,1\}^m$

iO

If C(x) = 0, output ⊥.
Else output PPRF(K, x).

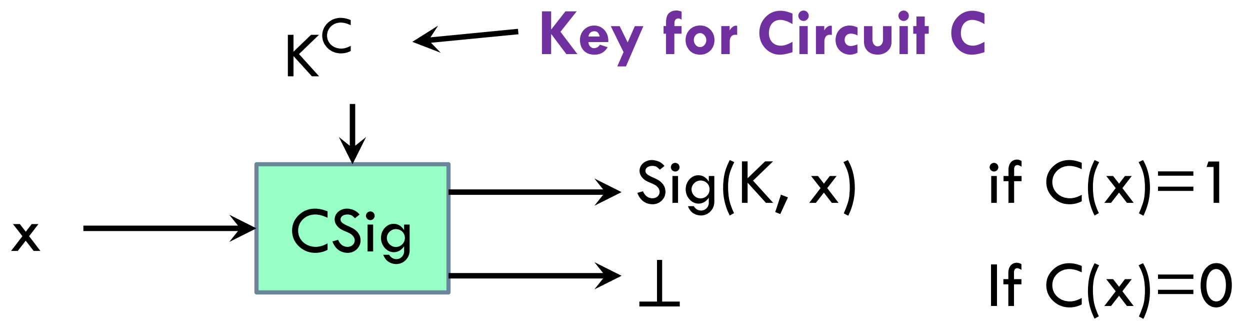
Security: Sample random t such that C(t)=0.

iO		iO
If C(x) = 0, output ⊥. Else output PPRF(K, x).	$\approx$	If C(x) = 0, output ⊥. Else output PPRF($K^{\{t\}}$, x).

- Adversary outputs PPRF(t) with probability > $\varepsilon_{CPRF}/2^n - \varepsilon_{iO}$.
- Complexity leveraging: iO and PPRF need sub-exp security.

Constrained Signature Scheme

Definition:



(Plus a verification algorithm Ver.)

For unsatisfying x, we get usual signature scheme security.

Applications:

- Mobile agents: Agent's signature looks just like Principal's (on messages that it is permitted to sign).
- Delegation: Signature on x is an argument that C(x)=1.

Constrained Signature Scheme

Construction:

PPRF $\{0,1\}^n \to \{0,1\}^m$. Let f be a public OWF. $|x| = k$.

Constrained signing	Verification
iO: If $C(x) = 0$, output $\perp$. Else output PPRF(K, x).	iO: Output f(PPRF(K, x)).

Security: Pick random t such that C(t)=0. Hybrids:

- Use $K^{\{t\}}$ and hardwire f(PPRF(K,t)) into Ver (iO security).
- Change f(PPRF(K,t)) to f(v) in Ver for random v (PPRF security).
- Adv forges v = Sig(t) with prob $> \varepsilon_{CSig}/2^k - 2\varepsilon_{IO} - \varepsilon_{PPRF}$.
  - OWF needs $> 2^k$ security $\to$ m > k $\to$ CSigs longer than messages.

Non-Deterministic Constrained PRFs/Sigs

Witness PRFs: [Zhandry '14] Get PRF on x if input satisfies R(x,w)=1

Applications:

- Multiparty NIKE without trust setup
  - Similar to Boneh-Zhandry protocol, but without iO
- Reusable WE and ABE with short cts (independent of relation size)
- Reusable secret sharing for NP with shorter shares
- Fully distributed broadcast encryption
- Maybe not enough for to achieve some things achievable via iO, like Boneh-Zhandry's traitor tracing protocol.

NIZK args for NP: [SW13] Get sig on x if input satisfies R(x,w)=1

A Few Constrained PRF/Sig papers

- Boneh, Waters: "Constrained Pseudorandom Functions and Their Applications". Asiacrypt 2013.
- Boyle, Goldwasser, Ivan: "Functional Signatures and Pseudorandom Functions". PKC 2014.
- Kiayias, Papadopoulos, Triandopoulos: "Delegatable Pseudorandom Functions and Applications". CCS 2013.
- Bellare, Fuchsbauer: "Policy-Based Signatures". PKC 2014.
- Backes, Meiser, Schroder: "Delegatable Functional Signatures". ePrint 2013.

- Chen, Zhang: "Publicly Evaluable Pseudorandom Functions and Their Applications". SCN 2014
- Georg Fuchsbauer, "Constrained Verifiable Random Functions". SCN 2014.
- Chandran, Raghuraman, Vinayagamurthy: "Constrained Pseudorandom Functions: Verifiable and Delegatable". ePrint 2014.
- Fuchsbauer, Konstantinov, Pietrzak, Rao: "Adaptive Security of Constrained PRFs". Asiacrypt 2014.
- Hofheinz, Kamath, Koppula, Waters: "Adaptively Secure Constrained Pseudorandom Functions". ePrint, 2014.
- Hohenberger, Koppula, Waters: Adaptively Secure Puncturable Pseudorandom Functions in the Standard Model". ePrint 2014.
- Abusalah, Fuchsbauer, Pietrzak: "Constrained PRFs for Unbounded Inputs". ePrint 2014.
- Cohen, Goldwasser, Vaikuntanathan: "Aggregate Pseudorandom Functions and Connections to Learning". TCC 2015.

Another Useful Trick: Extraction

- Show $iO(P) \approx iO(P^{\{t\}})$ even though $P(t) \neq \perp$.
  - Show distinguisher can extract "differing input" t.
  - Show t is hard to extract assuming OWFs.

- Show that $P^{\{t\}}$ lacks some property necessary for the adversary's attack.

iO and Differing Inputs Obfuscation (diO)

- Differing inputs obfuscation (diO):
  - Security definition: For every diO distinguisher, there is an extractor that gives a differing input.

- diO $\longrightarrow$ iO

- iO $\longrightarrow$ diO if # of differing inputs is *very small* [Boyle, Chung, Pass TCC 2014]
  - Apply iO scheme to programs P, P' that differ at one input t.
  - If iO(P) $\not\approx$ iO(P'), iO implies we can extract t.
    - If P' = $P^{\{t\}}$, we can extract t.
    - In general, Extractor's work scales with # of differing inputs.

iO → diO for One Differing Input

- Suppose P(x) = P'(x) for all x except t.
- Program P_k: If $x \geq k$ output P(x), else output P'(x).

- Assuming iO, if iO(P) $\not\approx$ iO(P'), we can find t by binary search.

Random Puncture is Undetectable

Let's Give $P^{\{t\}}$ Some Code:

Suppose t's domain supports an injective OWF f.

Let $y = f(t)$.

If $f(x) = y$, output $\perp$. Else, output $P(x)$.

- Assuming iO, if $iO(P) \not\approx iO(P^{\{t\}})$, we can break the OWF.
- Assuming iO and OWF, $iO(P) \approx iO(P^{\{t\}})$.

Circular Security from iO

Public Key: pk_1 and iO(P).

iO

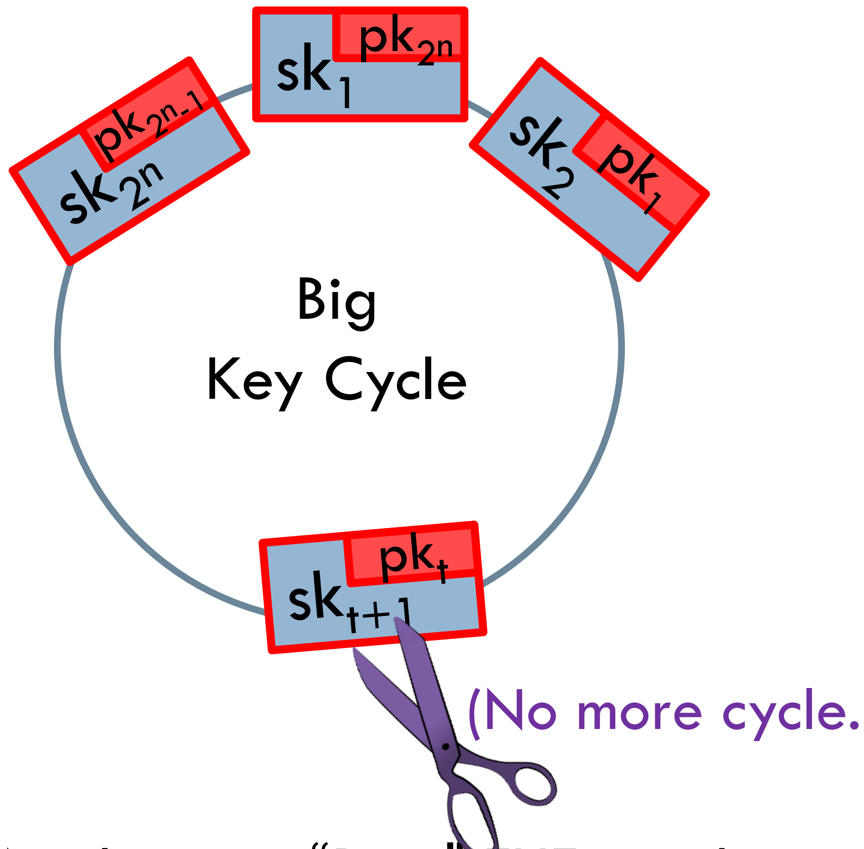
$i \longrightarrow$ P: Output pk_i, $Enc_{pk_i}(sk_{i+1})$.

$\approx$ If domain large enough to support OWFs

For random t:

iO

$i \longrightarrow$ $P^{\{t\}}$: If $i=t$, output $\perp$. Else pk_i, $Enc_{pk_i}(sk_{i+1})$.



Application: "Pure" FHE uses key cycle to handle unbounded depth.

Circular Security from iO: Rest of Proof

Additional Requirements: Encryption scheme is not only IND-CPA, but pairs (pk_i, $c_i = \mathrm{Enc}(pk_i, m)$) look pseudorandom.

$P^{\{t\}}$: If $i = t$, output $\perp$. Else output $P(i)$.	$Q^{\{t\}}$: If $i = t$, output $\perp$. Else output $Q(i)$.

P:

$(r_i, s_i) \leftarrow \mathrm{PPRF}(K, i)$
$(r_{i+1}, s_{i+1}) \leftarrow \mathrm{PPRF}(K, i+1)$
$(sk_i, pk_i) \leftarrow \mathrm{KGen}(r_i)$
$(sk_{i+1}, pk_{i+1}) \leftarrow \mathrm{KGen}(r_{i+1})$
$c_i \leftarrow \mathrm{Enc}(pk_i, sk_{i+1}; s_i)$
Output (pk_i, c_i)

Q:

Set $(pk_i', c_i') \leftarrow \mathrm{PRF}(K', i)$ except $pk_1' = pk_1$.
Output (pk_i', c_i')

Clearly IND-CPA is preserved when $iO(Q^{\{t\}})$ is added to public key.

Circular Security from iO: Rest of Proof

Claim:

iO: $P^{[t,u,0]}$: has PPRF key $K^{>u}$.
If $i > u$ output $P^{\{t\}}(i)$.
Else output $Q\{t\}(i)$.

$\approx$

iO: $P^{[t,u+1,0]}$: has PPRF key $K^{>u+1}$.
If $i > u+1$ output $P^{\{t\}}(i)$.
Else output $Q\{t\}(i)$.

iO security

iO: $P^{[t,u,1]}$: Like $P^{[t,u,0]}$, except output for u+1 hardwired.

iO security

iO: $P^{[t,u,2]}$: Like $P^{[t,u,1]}$, except PPRF key is now $K^{>u+1}$.

Pseudo-randomness

Related Work on Circular Security vs. iO

- iO + LFHE $\Rightarrow$ "Almost pure FHE"
  - Canetti, Lin, Tessaro, Vaikuntanathan: "Obfuscation of Probabilistic Circuits and Applications". TCC 2015.
  - Clear, McGoldrick: "Bootstrappable Identity-Based Fully Homomorphic Encryption". CANS, 2014.

- Negative results on circ. security (for poly-size cycles):
  - Marcedone, Orlandi:

    "iO $\Rightarrow$ (IND-CPA $\not\Rightarrow$ Circular Security)". SCN 2014.
  - Koppula, Ramchen, Waters: "Separations in Circular Security for Arbitrary Length Key Cycles". TCC 2015.

Trapdoor Permutation from iO [BPW15]

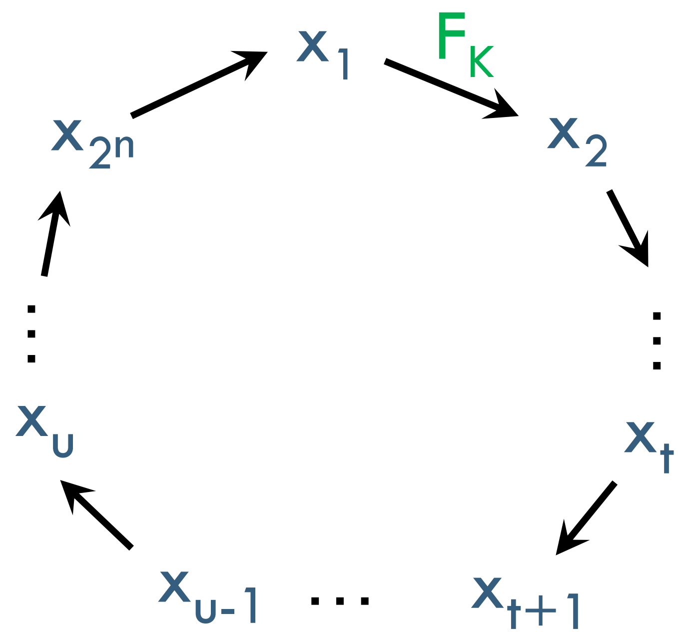
Trapdoor:

Key K for PRF

Public Key:

Obfuscated program for F_K.

$z \longrightarrow$ iO: F_K: If $z = x_i$ output x_{i+1}. Else output $\perp$.



Pseudorandom sampler:

$i \longrightarrow$ iO: Output $x_{PRG(i)}$.

$$x_i = (\ i, PRF(K, i)\ )$$

Remark: x_i's like secret keys in a key cycle.
Replace iO(F_k) with iO(P) that outputs Enc(pk_i, sk_{i+1}).
Get sk_{i+1} from sk_i via decryption.

Trapdoor Permutation from iO [BPW15]

Trapdoor:

Key K for PRF

Public Key:

Obfuscated program for F_K.

$z \longrightarrow$ iO

F_K: If $z = x_i$

output x_{i+1}.

Else output $\perp$.

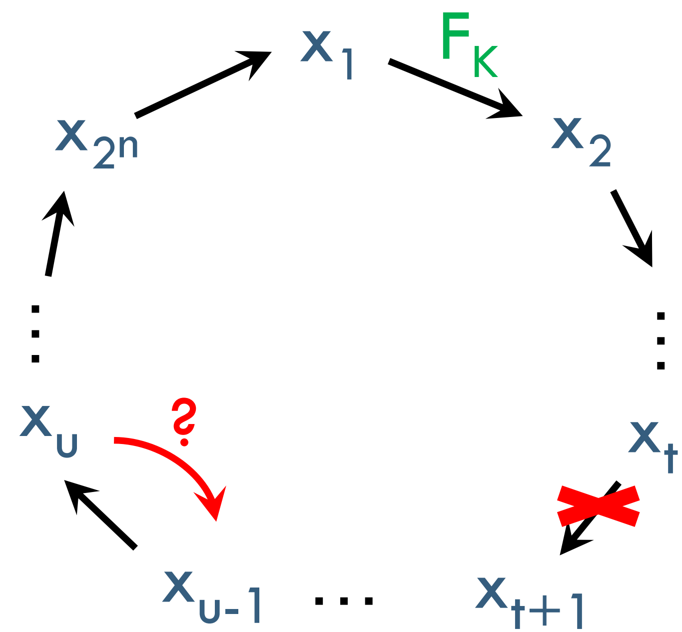
$\approx$

For random t:

iO

$F_K^{\{t\}}$: If $z = x_t$, output $\perp$.

Else output $F_K(z)$.



$x_i = (\ i,\ \mathrm{PRF}(K,\ i)\ )$

Other Work on iO vs. Delicate Graphs

- Bitansky, Paneth, Rosen: “On the Cryptographic Hardness of Finding a Nash Equilibrium”. ePrint 2015.
  - Prove that finding a Nash equilibrium of a game is hard, assuming the existence of iO and sub-exp OWFs.
  - Nash is PPAD-complete: [DGP09, CDT09].
  - END-OF-THE-LINE: Canonical PPAD-complete problem. Given succinct program P_G representing exponential-size directed graph G over $\{0,1\}^n$ with in/out degrees ≤ 1, and a source node s, find some other source/sink node.
  - $iO(P_G)$ for long-line-graph indistinguishable from $iO(P'_G)$ where end-of-the-line is inaccessible.

Scope of iO Applications

Roughly grouped by area:

1. FHE
2. Multilinear maps
3. Delegation
4. Secure multiparty computation
5. Garbled circuits
6. RAM computations
7. Differential privacy
8. Odds and Ends

FHE from iO and Re-randomizable PKE

Canetti, Lin, Tessaro, Vaikuntanathan: "Obfuscation of Probabilistic Circuits and Applications". TCC 2015.

Leveled FHE from iO+OWFs?

- Compact additive HE $\longrightarrow$ CRHFs.
  - Let $c_1 = \mathrm{Enc}(a)$, $c_2 = \mathrm{Enc}(b)$.
  - $H(x_1 \| x_2) = \mathrm{Enc}(ax_1 + bx_2)$ (computed homomorphically).
  - Collision gives linear equation on (a,b), violating semantic security.

- But don't know how to get CRHFs from iO+OWFs!
  - Asharov, Segev: "Limits on the Power of iO and FE" (ePrint 2015): "There is no fully black-box construction with a polynomial security loss of a collision-resistant function family from a general-purpose indistinguishability obfuscator and a… trapdoor permutation."

Leveled FHE from iO+Re-rand PKE

- Canetti, Lin, Tessaro, Vaikuntanathan: "Obfuscation of Probabilistic Circuits and Applications". TCC 2015.
  - "Assume the existence of a sub-exponentially indistinguishable IO for circuits, and a sub-exponentially secure OWF. Then any perfectly rerandomizable encryption scheme can be transformed into a leveled homomorphic encryption scheme."

L-Leveled FHE from iO + Re-rand PKE: Natural Approach

KeyGen as in PKE, but with obfuscated NANDs:

$i-1$ $\quad$ $c_1 = \mathrm{Enc}(pk_{i-1}, x)$ $\quad$ $c_2 = \mathrm{Enc}(pk_{i-1}, y)$

iO

Output

$\mathrm{Enc}(pk_i, \mathrm{NAND}(\mathrm{Dec}(sk_{i-1}, c_1), \mathrm{Dec}(sk_{i-1}, c_2)); \mathrm{PRF}(K, c_1, c_2))$

$c_3 = \mathrm{Enc}(pk_i, \mathrm{NAND}(x,y))$

General way to handle obfuscation of programs that use internal coins

L-Leveled FHE from iO + Re-rand PKE: Security Proof

- Uses 2 types of PKE scheme:
  - Normal
  - Trapdoor/Lossy: Enc(tpk, m; r) and Enc(tpk, 0; r) have statistically identical distributions.

- Hybrid H_k:
  - Use normal public keys pk_i for $i \leq k$.
  - Trapdoor keys tpk_i for $i > k$, and obfuscate program that just outputs encryptions of 0.
  - H_L: real game.
  - H_0: Challenge ct is always an encryption of 0.

L-Leveled FHE from iO + Re-rand PKE: Hybrids

$H_{i-1,0}$ iO: Output $\mathrm{Enc}(pk_i, \mathrm{NAND}(\mathrm{Dec}(sk_{i-1}, c_1), \mathrm{Dec}(sk_{i-1}, c_2)); \mathrm{PRF}(K,c_1,c_2))$

Trapdoor key indistinguishability

$H_{i-1,1}$ iO: Output $\mathrm{Enc}(tpk_i, \mathrm{NAND}(\mathrm{Dec}(sk_{i-1}, c_1), \mathrm{Dec}(sk_{i-1}, c_2)); \mathrm{PRF}(K,c_1,c_2))$

iO security

$H_{i-1,2,j,1}$ For $j = (c_1^*, c_2^*)$, hardcode response to (i-1,j) and puncture PRF there.

PPRF security

$H_{i-1,2,j,2}$ iO: Output $\mathrm{Enc}(tpk_i, \mathrm{NAND}(\mathrm{Dec}(sk_{i-1}, c_1^*), \mathrm{Dec}(sk_{i-1}, c_2^*)); r)$

Statistical re-randomizability

$H_{i-1,2,j,3}$ iO: Output $\mathrm{Enc}(tpk_i, 0; r)$

PPRF security and iO security

$H_{i-1,2,j,4}$ iO: Output $\mathrm{Enc}(tpk_i, 0; \mathrm{PRF}(K,j))$

L-Leveled FHE from iO + Re-rand PKE: Hybrids

$H_{i-1,0}$ iO: Output Enc(pk_i, NAND(Dec(sk_{i-1}, c_1), Dec(sk_{i-1}, c_2)); PRF(K,c_1,c_2))

Trapdoor key indistinguishability

$H_{i-1,1}$ iO: Output Enc(tpk$_i$, NAND(Dec(sk_{i-1}, c_1), Dec(sk_{i-1}, c_2)); PRF(K,c_1,c_2))

iO security

$H_{i-1,2,j,1}$ For j = (c_1*, c_2*), hardcode response to (i-1,j) and

PPRF security

$H_{i-1,2,j,2}$ iO: Output Enc(tpk$_i$, NAND(Dec(sk_{i-1}, c_1*), Dec

Statistical re-randomizability

$H_{i-1,2,j,3}$ iO: Output Enc(tpk$_i$, 0; r)

PPRF security and iO security

$H_{i-1,2,j,4}$ iO: Output Enc(tpk$_i$, 0; PRF(K,j))

Need > 2^{2len} security where *len* is then length of a ciphertext.

Hence need statistical re-randomizability.

Toward iO-based Multilinear Maps?

A Positive Result

- Yamakawa, Yamada, Hanaoka, Kunihiro: "Self-bilinear Map on Unknown Order Groups from Indistinguishability Obfuscation and Its Applications". Crypto 2014.
- Encoding of x consists of the obfuscation:

$g^y \longrightarrow$ iO [F_x: Output $e(g^x,g^y) \stackrel{\text{def}}{=} g^{2xy}$ by powering by 2x.] $\approx$ iO [F_x': Output g^{2xy} by powering by $2x+\mathrm{ord}(g)$.]

Odd number

- Degree of maps is unbounded-polynomial
- Applications to non-interactive key agreement and distributed broadcast encryption schemes.

The Problem with Truly Unbounded MMaps

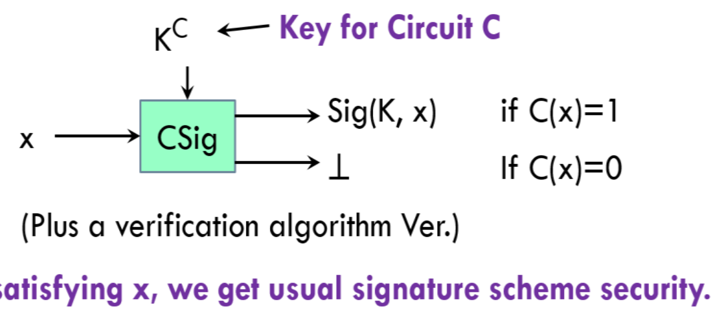
- van Dam, Hallgren, Ip, "Quantum Algorithms for Some Hidden Shift Problems". SODA 2003.
  - FHE scheme with equality test can be broken in quantum polynomial time.
  - Seems difficult to base security of truly unbounded mmap on quantum-resistant assumption like LWE.

Delegation using iO

Delegation using iO

- Constrained sigs via iO
  - But better options?
    - Yael's no-signaling proofs
      - Paneth/Rothblum use mmaps to get public verifiability here
    - Reusable garbled circuits
    - Fully homomorphic sigs
    - Quadratic arithmetic programs

Definition:



- Boneh, Gupta, Mironov, Sahai, "Hosting Services on an Untrusted Cloud", Eurocrypt 2015.
  - iO used to outsource provider's entire multi-client service, with privacy for both the provider and clients against the host.

Improving Secure MPC with iO

Some Applications of iO to MPC

- **Two-round adaptively secure MPC**
  - Canetti, Goldwasser, Poburinnaya: “Adaptively Secure 2PC From iO”. TCC 2015.
  - Dachman-Soled, Katz, Rao: “Adaptively Secure, Universally Composable MPC in Constant Rounds”. TCC 2015.
  - Garg, Polychroniadou: “Two-Round Adaptively Secure MPC from iO”. TCC 2015.
  - Garg, Gentry, Halevi, Raykova: “Two-round [static] secure MPC from iO”. TCC 2014.

- **SFE with Long Output**
  - Hubacek, Wichs, “On the Communication Complexity of SFE with Long Output”. ITCS 2015.
  - Jakobsen, Orlandi, “How to Bootstrap Anonymous Communication”. ePrint 2015.

Two-Round Adaptively-Secure MPC

Useful Ingredient: Deniable encryption, Explainability Compilers

- Definition [Explainability Compiler] (following [SW13, DKR15])
  A PPT algorithm Comp is an explainability compiler if for every efficient randomized circuit Alg, the following hold:
  - **Polynomial slowdown**: There is a polynomial $p(\cdot)$ such that for any (Alg*,Explain) output by Comp(1^λ, Alg) it holds that $|\mathrm{Alg}^*| \leq p(\lambda)|\mathrm{Alg}|$.
  - **Statistical functional equivalence**: Distributions of Alg*(x) and Alg(x) are statistically close for all x.
  - **Explainability**: Adversary A has negl advantage in following game:

> A(1^λ) outputs x* of its choice (selects target input).
> (Alg*, Explain) $\leftarrow$ Comp(1^λ, Alg).
> Choose uniform coins r_0 ∈ and compute $y^* = \mathrm{Alg}^*(x^*; r_0)$.
> Compute $r_1 \leftarrow$ Explain(x*, y*).
> Choose random bit b and give (Alg*, y*, r_b) to A.
> A tries to guess b.

Explain allows simulator to generate consistent randomness.

Explainability Compiler: Construction

Alg*:

u is a nonmalleable "encryption" of (x,y')

Hardwired keys: K_1, K_2, K_3

Input: x and randomness $u = u[1] \| u[2]$

1. Sparse Hidden Trigger: If $(x',y',r') \leftarrow \mathrm{PPRF}(K_3, u[1]) \oplus u[2]$ satisfies $x = x'$ and $u[1] = \mathrm{PPRF}(K_2, (x',y',r'))$, output y'.
2. Normal: Output $\mathrm{Alg}(x; \mathrm{PPRF}(K_1,(x,u)))$.

Explain:

Outputs an "encryption" of (x,y')

Hardwired keys K_2, K_3

Input: x, y' and randomness r

1. $u[1] \leftarrow \mathrm{PPRF}(K_2, (x,y',\mathrm{PRG}(r)))$ $u[2] \leftarrow \mathrm{PPRF}(K_3,u[1]) \oplus (x,y',\mathrm{PRG}(r))$.
2. Output $u \leftarrow u[1] \| u[2]$.

Communication Complexity of SFE with Long Output [Hubacek,Wichs'15]

- SFE with Long Output Scenario:
  - Function $f : \{0,1\}^A \times \{0,1\}^B \to \{0,1\}^t$.
  - Output is long: $t \gg A$. Maybe also $t \gg B$.
  - Want SFE protocol for Bob to get $f(x_A,x_B) \mid \in \{0,1\}^t$.
- Examples:
  - x_A = PRF key K, $f(x_A,\emptyset) = PRF(K,1), \ldots, PRF(K,L)$.
  - x_A = partial decryption key, x_B = encryption of long output of MPC protocol under threshold PKE, f = partial decryption procedure.
- Communication complexity:
  - Can we get communication sublinear in t?
  - Possible with insecure protocols.
  - FHE doesn't do this.
  - Constrained PRF? No, not simulatable from PRF outputs alone.

Communication Complexity of SFE with Long Output [Hubacek,Wichs'15]

- Negative Results:
  - Output-dependence inherent when Bob is malicious or even honest-but-deterministic.
  - The Problem: incompressibility
    - view_{Bob} allows Bob to compute PRF(K,1), …, PRF(K,L)
    - $|\text{view}_{Bob}| \ll t$ if Bob is deterministic, $|x_B| \ll t$, and $|\text{comm}| \ll t$.
    - view_{Bob} compresses PRF(K,1), …, PRF(K,L) (impossible).
- Positive results
  - Use iO to remove output-dependence in semi-honest case.
  - Requires Bob to use long randomness with succinct decommitments via an iO-friendly Merkle tree hash.

Communication Complexity of SFE with Long Output [Hubacek,Wichs'15]

Real World

Bob sends $z = H(r_1, \ldots, r_L)$.

Alice sends $iO(C_{K,z})$.

$iO(C_{K,z})$:

iO

Hardwired: key K, hash output z

Inputs: $(i \leq L, r_i, \pi_i)$

1. Verify decommitment π_i that r_i is i-th bit of hashed randomness.
2. If so, output $y_i = PRF(K, i)$.

Simulation

$z = H(r_1, \ldots, r_L)$ for $r_i = y_i \oplus PRF(K_{sim}, i)$.

Modified obfuscation.

$iO(C_{K_{sim}, \{y_i \,:\, i \leq L\}, z})$:

iO

Hardwired: key K_{sim}, H output z

Inputs: $(i \leq L, r_i, \pi_i)$

1. Verify decommitment π_i that r_i is i-th bit of hashed randomness.
2. If so, output $y_i = r_i \oplus PRF(K_{sim}, i)$.

Not functionally equivalent when $(i, r' \neq r_i, \pi_i')$ is a valid decommitment!

Communication Complexity of SFE with Long Output [Hubacek,Wichs'15]

iO-Friendly Merkle Tree (somewhere statistically binding hash):

- Functionality of a Merkle tree:
  - Short hash key: $z = H_{hk}(r_1, \ldots, r_L)$
  - Allows short decommitment of r_i.
- iO-friendly security:
  - Some index i is statistically binding: For that i, there is no opening to $r_i' \neq r_i$.
  - Hash key hk does not reveal which index i is binding.
- Construction: from FHE.

More iO-Friendly Techniques

- Koppula, Lewko, Waters: "Indistinguishability Obfuscation for Turing Machines with Unbounded Memory". STOC 2015.
  - Positional accumulators: Similar in concept to somewhere statistically binding hashes. Allows short commitment to large storage that is unconditionally sound for some hidden index.

More Powerful Protocols via iO

Functionalities:

- Goldwasser, Gordon, Goyal, Jain, Katz, Liu, Sahai, Shi, Zhou: "Multi-Input Functional Encryption". Eurocrypt '14.
  - Given ciphertexts Enc(x_1), …, Enc(x_n) and the secret key sk_f for n-ary function f, one can compute f(x_1, …, x_n), and nothing else about the {x_i's}.
- Waters, "A Punctured Programming Approach to Adaptively Secure FE". ePrint 2014.
- Boneh, Zhandry, "Multiparty Key Exchange, Efficient Traitor Tracing, and More from Indistinguishability Obfuscation". Crypto 2014.

More Powerful Protocols via iO

Functionalities from Mmaps (not iO):

- Boneh, Waters, Zhandry, “Low Overhead BE from Multilinear Maps”. Crypto 2014.
- Garg, Gentry, Halevi, Zhandry: “Fully Secure ABE using Multilinear Maps”. ePrint 2014.
- Garg, Gentry, Halevi, Zhandry: “Fully Secure FE without Obfuscation”. ePrint 2014.
- Boneh, Lewi, Raykova, Sahai, Zhandry, Zimmerman: “Semantically Secure Order-Revealing Encryption: Multi-Input FE without Obfuscation”. Eurocrypt ‘15.

More Powerful Protocols via iO

ZK, WI, Concurrency:

- Bitansky, Paneth: “ZAPs and NIWI from iO”. TCC 2015.
- Pandey, Prabhakaran, Sahai: “Obfuscation-based Non-black-box Simulation and Four Message Concurrent ZK for NP”. TCC 2015.
- Chung, Lin, Pass: “Constant-Round Concurrent ZK from iO”. ePrint 2014.

More Powerful Protocols via iO

Frameworks:

- Hofheinz, Jager, Khurana, Sahai, Waters, Zhandry: "How to Generate and use Universal Parameters". ePrint 2014.
- Agrawal, Agrawal, Prabhakaran: "Cryptographic Agents: Towards a Unified Theory of Computing on Encrypted Data". Eurocrypt 2015.

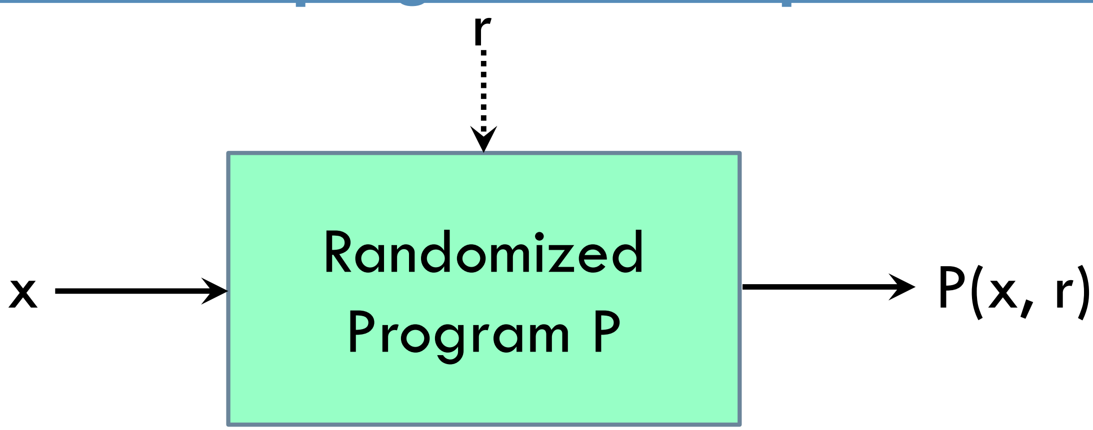
iO of Randomized Functionalities (Like Circuit Garbling)

Papers Considering Randomized Functionalities

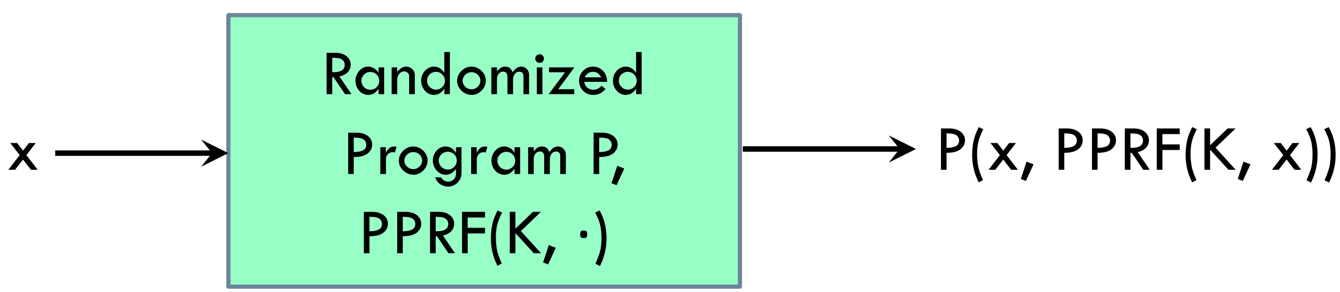
- Alwen, Barbosa, Farshim, Gennaro, Gordon, Tessaro, Wilson: “On the Relationship between Functional Encryption, Obfuscation, and Fully Homomorphic Encryption”. IMACC, 2013.
- Goyal, Jain, Koppula, Sahai: “Functional Encryption for Randomized Functionalities”. TCC 2015.
- Canetti, Lin, Tessaro, Vaikuntanathan: “Obfuscation of Probabilistic Circuits and Applications”. TCC 2015.

iO for Randomized Functionalities

Can we obfuscate a program that flips coins internally?



If we make it pseudorandom?



iO for Randomized Functionalities

The Question:

$$\{P_1(x, r) : r \in R\} \approx \{P_2(x, r) : r \in R\} \text{ for every } x$$

?

$$[P_1, P_2, iO(P_1(x, PPRF(K, x)))] \approx [P_1, P_2, iO(P_2(x, PPRF(K, x)))]$$

Certainly not true that $P_1(x,r) = P_2(x,r)$ for all (x,r)!

iO for Randomized Functionalities

iO

$P^{\leq t}(x)$:

If $x \leq t$, output $P_1(x, PPRF(K,x))$.

If $x > t$, output $P_2(x, PPRF(K,x))$.

iO security

iO

Hardcode $y = P_1(t, PPRF(K,t))$.

If $x < t$, output $P_1(x, PPRF(K^{\{t\}},x))$.

If $x > t$, output $P_2(x, PPRF(K^{\{t\}},x))$.

If $x = t$, output y.

PPRF security

iO

Hardcode $y = P_1(t, r_1)$.

If $x < t$, output $P_1(x, PPRF(K^{\{t\}},x))$.

If $x > t$, output $P_2(x, PPRF(K^{\{t\}},x))$.

If $x = t$, output y.

iO

$P^{\leq t+1}(x)$:

If $x < t$, output $P_1(x, PPRF(K,x))$.

If $x \geq t$, output $P_2(x, PPRF(K,x))$.

iO security

iO

Hardcode $y = P_2(t, PPRF(K,t))$.

If $x < t$, output $P_1(x, PPRF(K^{\{t\}},x))$.

If $x > t$, output $P_2(x, PPRF(K^{\{t\}},x))$.

If $x = t$, output y.

PPRF security

iO

Hardcode $y = P_2(t, r_2)$.

If $x < t$, output $P_1(x, PPRF(K^{\{t\}},x))$.

If $x > t$, output $P_2(x, PPRF(K^{\{t\}},x))$.

If $x = t$, output y.

Indistinguishable distributions

Randomized Encodings

Randomized Encoding RE for a circuit family $\mathcal{C}$

- Given $C \in \mathcal{C}$ input x, RE(C,x) outputs (C’,x’) such that:
  - From (C’, x’) one can efficiently recover C(x)
  - Given C(x), one can efficiently simulate the pair (C’,x’), implying RE(C,x) reveals nothing beyond C(x).
  - Also, RE is typically not only efficient, but has low parallel complexity (e.g., it’s in NC^1).

iO and Randomized Encodings

Using Randomized Encodings to Bootstrap iO

- If we have VBB for class WEAK that includes RE and a PRF, we get VBB for get for general circuits.
  - O(C)(x) outputs RE(C,x), using PRF(K,x) as randomness.

- [CLTV15] iO suffices for this purpose.
  - If C_1, C_2 are functionally equiv., $\{RE(C_1, x; r)\} \approx \{RE(C_2, x; r)\}$.
  - Therefore $iO(C_1(x,PRF(K,x))) \approx iO(C_1(x,PRF(K,x)))$
  - Don't need FHE to bootstrap iO.

Obfuscating RAM Computations

Obfuscating RAM Computations

- Gentry, Halevi, Raykova, Wichs: “Outsourcing Private RAM Computation”. FOCS 2014.
- Bitansky, Garg, Lin, Pass, Telang: “Succinct Randomized Encodings and their Applications”. STOC 2015.
- Canetti, Holmgren, Jain, Vaikuntanathan: “Succinct Garbling and Indistinguishability Obfuscation for RAM Programs”. STOC 2015.
- Canetti, Holmgren: “Fully Succinct Garbled RAM”. ePrint 2015.
- Chen, Chow, Chung, Lai, Lin, Zhou: “Computation-Trace Indistinguishability Obfuscation”. ePrint 2015.

Garbling RAM Computations

- Lu, Ostrovsky: “How to Garble RAM Programs?” Eurocrypt 2013.
  - Seminal work on garbling programs without going through circuits.
- Gentry, Halevi, Lu, Ostrovsky, Raykova, Wichs: “Garbled RAM revisited.” Eurocrypt 2014.
- Garg, Lu, Ostrovsky, Scafuro: “Garbled RAM from OWFs”. STOC’15.
  - Based on OWFs
  - Runtime proportional to runtime of plaintext RAM program
  - But garbled program size also proportional to runtime.

Fully Succinct Garbled RAM [CH15]

- Fully succinct garbling scheme for RAM programs assuming iO for circuits and OWFs.
  - Fully succinct: the size, space requirements, and runtime of the garbled program are the same as those of the input program, up to poly-logarithmic factors and a polynomial in the security parameter.
- Constructs iO for RAMs assuming iO for circuits and sub-exp OWFs.
- Combines iO, garbling, ORAM, and other techniques.

Differential Privacy

iO + Differential Privacy: A Positive App

- Scenario:
  - Hospitals generate patient records
  - Medical researchers want access to patient records to test hypotheses
- Differential privacy:
  - Publish differentially-private “noisy” “sanitized” DB
  - Hospitals don’t need to interact in each research analysis
  - Researchers don’t need to share hypothesis or algorithm
  - Issue: Allowing diverse research analytics → high accuracy loss [DNR⁺09].

iO + Differential Privacy: A Positive App

- Randomized FE/iO can help. Randomized FE approach:
  - Government is authority of system.
  - Government issues key to researcher if function is differentially private.
  - Randomized FE decryption adds noise to exact response, to “sanitize” it. Decryption process adds low-level of noise to exact response.
  - Pro: Better accuracy, since noise can be added fresh for each function.
  - Con: Researcher reveals hypothesis to government.
    - Maybe automate government’s role using iO.

iO vs. Differential Privacy

- iO gives very efficient traitor tracing schemes [BZ14].
- Traitor tracing is opposite of differential privacy [Dwork, Naor, Reingold, Rothblum, Vadhan, STOC ’09]
  - Differential privacy: Summarize data in meaningful way while hiding individual information.
  - Traitor tracing: Prevent keys from being summarized in a way that hides individual information.

More on iO and Complexity Theory

- Bitansky, Paneth, Rosen: “On the Cryptographic Hardness of Finding a Nash Equilibrium”. ePrint 2015.
  - Finding Nash Eq hard assuming iO and sub-exp OWFs.
- Bun, Zhandry: “Order-Revealing Encryption and the Hardness of Private Learning”. ePrint 2015.
  - Separates efficient PAC learning from efficient differentially-private PAC learning assuming iO and simple primitives.
- Cohen, Goldwasser, Vaikuntanathan: “Aggregate PRFs and Connections to Learning”. TCC 2015.
  - [Val84]: PRF in a complexity class C implies the existence of concept classes in C unlearnable by membership queries.
  - Explores implications of constrained PRFs etc.

Odds and Ends

Software Watermarking

- Nishimaki, Wichs: “Watermarking Cryptographic Programs Against Arbitrary Removal Strategies”. ePrint 2015.
- Cohen, Holmgren, Vaikuntanathan: ”Publicly Verifiable Software Watermarking”. ePrint 2015.
- Result: For certain types of programs, like PRFs, they create a marked program $C^{\#}$ such that:
  - Evaluates C correctly on overwhelming fraction of inputs
  - Adversary cannot come up with any program with mark removed that evaluates correctly on even a small fraction of inputs.

Hashing Using iO

- Bellare, Stepanovs, Tessaro: "Poly-Many Hardcore Bits for Any One-Way Function and a Framework for Differing-Inputs Obfuscation". Asiacrypt 2014.
- Brzuska, Mittelbach: "Using Indistinguishability Obfuscation via UCEs". Asiacrypt 2014.
- Canetti, Chen, Reyzin: "On the Correlation Intractability of Obfuscated Pseudorandom Functions". ePrint 2015.
- Hohenberger, Sahai, Waters: "Replacing a Random Oracle: Full Domain Hash From Indistinguishability Obfuscation". Eurocrypt 2014.
- Bernstein, Lange, van Vredendaal et al. ePrint 2015: "Bad Directions in Cryptographic Hash Functions"?

Impossibilities Implied by iO

- Bitansky, Canetti, Cohn, Goldwasser, Kalai, Paneth, Rosen: ”The Impossibility of Obfuscation with Auxiliary Input or a Universal Simulator”. Crypto 2014.
- Brzuska, Farshim, Mittelbach: “Random-Oracle Uninstantiability from iO”. TCC 2015.
- Brzuska, Farshim, Mittelbach: “iO and UCEs: The Case of Computationally Unpredictable Sources”. Crypto 2014.
- Brzuska, Mittelbach: “iO versus Multi-Bit Point Obfuscation with Auxiliary Input”. Asiacrypt 2014.
- Green, Katz, Malozemoff, Zhou, “A Unified Approach to Idealized Model Separations via iO”. ePrint 2014.

Thank You! Questions?