APPLICATIONS OF (INDISTINGUISHABILITY) OBFUSCATION

Craig Gentry, IBM Research
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 P₁ and P₂ of equal complexity and functionality:

\[ |\Pr[A(iO(P₁))=1] - \Pr[A(iO(P₂))=1]| \text{ is negligible.} \]

- In English: If two programs have the same input-output behavior, the adversary cannot distinguish which was obfuscated.

- **Circuits**: Usually our model of computation.
Plant of the Talk

VBBeanstalk
[BGI+01]
Does not exist.

Amazing Crypto Applications

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

Warning: Not to scale!
Plant of the Talk

VBBeanstalk  
[BGI+01] 
Does not exist.

Amazing Crypto Applications

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

Warning: Not to scale!
Simple Applications of iO
Best-Possible Obfuscation

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

Indist. Obfuscation

Best Obfuscation

Some program $P$

$P(x)$

Indist. Obfuscation

Padding

Some program $P$

$P(x)$

$\approx$ Computationally Indistinguishable
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

Indist. Obfuscation

Demo version of program P with features laboriously removed

\[ P(x) \]

\[ \approx \]

Computationally Indistinguishable

Indist. Obfuscation

Restricted Some program P Interface

\[ P(x) \]
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: $\{m, \bot\} \leftarrow WDec(w;c)$ works if w is a witness for $x \in L$.

- **Secret key?**: No “secret key” per se.

- **Security**: $WEnc(x;m_0) \approx WEnc(x;m_1)$ when $x \notin L$. 
Witness Encryption from iO [GGHRSW13]

Program $P_{x,m}(w)$: Output $m$ if $w$ is a witness that $x \in L$. Otherwise output $\bot$.

Indist. Obfuscation

$\approx$

Computationally Indistinguishable if $x \notin L$.

Program that always outputs $\bot$.

$\bot$
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]

- **KeyGen**: PRG : \{0,1\}^n \rightarrow \{0,1\}^{2n}.
  - Secret key: s^* \in \{0,1\}^n. Public key: t^* = PRG(s^*).

- **Encryption**:
  - Let \( x_{t^*} \) be the statement “\( \exists s \text{ such that } t^* = PRG(s) \)”.
  - \( c \leftarrow \text{WEnc}(x_{t^*};m) \).

- **Decryption**: \( m \leftarrow \text{WDec}(s^*;c) \).

Hyb\(_0\) (real world): \( c = \text{WEnc}(x_{t^*}; m_0) \).

Hyb\(_1\) (t \leftarrow \{0,1\}^{2n}): \( c = \text{WEnc}(x_t; m_0) \).

Hyb\(_2\) (m\(_0\) \rightarrow m\(_1\)): \( c = \text{WEnc}(x_t; m_1) \).

Hyb\(_3\) (real world): \( c = \text{WEnc}(x_{t^*}; m_1) \).
Other Apps of WE (+ Simple Primitives)

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

Two main techniques:
1. Shell games with secrets
2. Punctured programs
1st Technique: Shell Games with Secrets
Shell Games with Secrets

**Thought Experiment**

Indist. Obfuscation

Program $P_1$ uses secret $sk_1$ and does not contain $sk_2$.

Functionally Equivalent

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

Indist. Obfuscation

Program $P_1$ uses secret $sk_1$ and does not contain $sk_2$.

Functionally Equivalent

$\approx$

Computationally Indistinguishable

Indist. Obfuscation

Program $P_2$ uses secret $sk_2$ and does not contain $sk_1$.

- $P_1$ 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 \([RAD78, Gen09, \ldots]\)

Data \(x\)

FHE Encryption

Data \(x\)

Eval( \(pk, C, Enc(x)\) )

Circuit \(C\)

FHE Encryption

Eval( \(pk, U, Enc(C), Enc(x)\) )

Eval( \(pk, U, Enc(C), x\) )

C(x)

\(U =\) 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\(^1\))

Obfuscation of General Circuit C

Program CondDec:
Hardwired: sk\(_1\) and \(\{e_i = \text{Enc}(pk_i, c)\}\)
Verify proof \(\pi\) that
\(c_1 = \text{Eval}(pk_1, U, e_1, x),\)
\(c_2 = \text{Eval}(pk_2, U, e_2, x).\)
If true, output \(\text{Dec}(sk_1, c_1)\); else output \(\bot\).

C(x) or \(\bot\)
Shell Game Application: iO for Circuits from (iO for NC^1) + (FHE with decryption in NC^1)

- **CondDec_{sk_1}:**
  - Verify proof \( \pi \) that
  \[ c_1 = \text{Eval}(pk_1, U, e_1, x), \]
  \[ c_2 = \text{Eval}(pk_2, U, e_2, x). \]
  - If true, output \( \text{Dec}(sk_1, c_1) \).
  - Else output \( \bot \).

- **CondDec_{sk_2}:**
  - Verify proof \( \pi \) that
  \[ c_1 = \text{Eval}(pk_1, U, e_1, x), \]
  \[ c_2 = \text{Eval}(pk_2, U, e_2, x). \]
  - If true, output \( \text{Dec}(sk_2, c_2) \).
  - Else output \( \bot \).

- \( \cong \)

- **CondDec_{sk_1} and CondDec_{sk_2}** 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 $H_{C_{b_1}, C_{b_2}, sk_{b_3}}$ the obfuscation consists of:
  
  $e_1 = \text{Enc}(pk_1, C_{b_1})$, $e_2 = \text{Enc}(pk_2, C_{b_2})$, iO($\text{CondDec}_{sk_{b_3}}$)

Real iO of $C_1$

$H_{C_1, C_1, sk_1}$

Real iO of $C_2$

$H_{C_2, C_2, sk_1}$

FHE security under $pk_2$

$H_{C_1, C_2, sk_1}$

$H_{C_1, C_2, sk_2}$

$H_{C_2, C_2, sk_2}$

iO security

FHE security under $pk_1$
Functional Encryption [S84, SW05, BSW11, …]

Syntax

- $(MPK, MSK) \leftarrow FE.\text{Setup}(1^\lambda)$
- $sk_f \leftarrow FE.\text{KeyGen}(MSK, f)$
- $ct_x \leftarrow FE.\text{Enc}(MPK, x)$
- $f(x) \leftarrow FE.\text{Dec}(sk_f, ct_x)$
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 \[ \text{GGHRSW13} \]

- **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:
  - \(c_1 \leftarrow \text{Enc}(pk_1, x; r_1)\),
  - \(c_2 \leftarrow \text{Enc}(pk_2, x; r_2)\),
  - NIZK proof \(\pi\) that \(c_1, c_2\) encrypt same value: that \(\exists\ r_1, r_2\ s.t.
    \(c_1 = \text{Enc}(pk_1, x; r_1), c_2 = \text{Enc}(pk_2, x; r_2)\).

**Decryption key for \(f\):**
Verify proof \(\pi\) that \(c_1, c_2\) encrypt same value under \(pk_1, pk_2\).
If so, output \(f(\text{Dec}(sk_1, c_1))\).
Else output \(\bot\).

\(f(x)\) or \(\bot\)
Shell Game Application:
Functional Encryption for Circuits

Security Proof for FE Scheme:

- Hybrids: In $H_{x_{b_1}, x_{b_2}, sk_{b_3}}$ 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 $H_{x_{b1},x_{b2},sk_{b3}}$ the challenge ciphertext is $c_1^*$, $c_2^*$, $\pi^*$, where $c_1^*$ encrypts $x_{b1}$, $c_2^*$ encrypts $x_{b2}$, and user keys decrypt under $sk_{b3}$.

---

**Diagram:**

- **Real enc of $x_1$**
  - $H_{x_1,x_1,sk_1}$
  - $H'_{x_1,x_1,sk_1}$
  - PKE security under $pk_2$

- **Stat. sim. sound proof allows “escape hatch” only for challenge ciphertexts $c_1^*$, $c_2^*$**

- **Real enc of $x_2$**
  - $H_{x_2,x_2,sk_1}$
  - $H'_{x_2,x_2,sk_1}$
  - iO security

- **PKE security under $pk_1$**
  - $H'_{x_1,x_2,sk_1}$
  - $H'_{x_1,x_2,sk_2}$
  - iO security

- **iO security**
2nd Technique: Punctured Programs
**Punctured Programming** [SW13]

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

\[ x \rightarrow P\{t\} \rightarrow P(x) \]

\[ \perp \text{ if } x \neq t \]
\[ \perp \text{ if } x = t \]

**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:**

\[
Punctured \text{ Key} \quad K^{\{t\}} \quad \text{PPRF}\(K, x)\]

if \(x \neq t\)

\[
\downarrow
\]

if \(x = t\)

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

**From GGM:**

\[\text{PRG} \ G : \{0,1\}^s \rightarrow \{0,1\}^{2s}\]

\[G(r) = G_0(r) \ || \ G_1(r)\]

\[\text{PRF}(K, x) = G_{x_n}(\cdots G_{x_1}(K) \cdots)\]
PKE Using Punctured PRFs [SW13]

**Secret Key:**
Key $K$ for symmetric encryption

**Public Key:**

![Diagram](image)

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]

**Secret Key:**
Key $K$ for PRF

**Public Key:**

$$m \oplus \text{PRF}(K, r)$$

**Problem:** Stream cipher encryption is its own inverse!
PKE Using Punctured PRFs [SW13]

**Secret Key:**
Key $K$ for PRF

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

- $m \oplus \text{PPRF}(K, G(r))$
- $G(r)$

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

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

Actual Scheme
Super-fast Decryption!
PKE Using Punctured PRFs [SW13]

**Secret Key:**
Key K for PRF

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

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

Use $\text{PPRF}(K^{\{t\}}, \cdot)$ instead of $\text{PPRF}(K, \cdot)$ inside Enc. (iO security, since $t$ is almost certainly not in range of $G$.)
PKE Using Punctured PRFs [SW13]

**Secret Key:**
Key K for PRF

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

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

Replace PPRF($K$, $t$) with random value $u$. (Punctured PRF security)
PKE Using Punctured PRFs [SW13]

**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).
  - \( \varepsilon_{selective}(\lambda) \leq \varepsilon_{iO}(\lambda) + \varepsilon_{PPRF}(\lambda) \).

- **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
    \( \varepsilon_{adaptive}(\lambda)/2^k \leq \varepsilon_{selective}(\lambda) \leq \varepsilon_{iO}(\lambda) + \varepsilon_{PPRF}(\lambda) \).
  - So \( \varepsilon_{adaptive}(\lambda) \leq 2^k (\varepsilon_{iO}(\lambda) + \varepsilon_{PPRF}(\lambda)) \).
  - Choose \( \lambda = \text{poly}(k) \) so that \( \varepsilon_{adaptive}(\lambda) \) is negligible.

Security at one input boosted to security at many inputs.
**Constrained PRF**

**Definition:**

\[ \text{PRF}(K, x) \]

- If \( C(x) = 1 \)
- If \( C(x) = 0 \)

*Key for Circuit C*

Security: \( \text{PRF}(K, x) \) 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

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

- iO

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

- \( \approx \)

  If \( C(x) = 0 \), output \( \perp \).
  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:**

\[
K^C \xrightarrow{\text{Key for Circuit C}} \\
\xrightarrow{\text{C Sig}} \xrightarrow{\text{Sig}(K, x)} \xrightarrow{\bot} \text{if } C(x)=1 \\
\xrightarrow{\bot} \text{If } C(x)=0
\]

(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 \(\{0,1\}^n \rightarrow \{0,1\}^m\). Let \(f\) be a public OWF. \(|x| = k\).

Constrained signing:
- If \(C(x) = 0\), output \(\bot\).
- Else output \(PRF(K, x)\).

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

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 → iO**

- **iO → 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.
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 \( \text{iO} \), if \( \text{iO}(P) \not\approx \text{iO}(P') \), we can find \( t \) by binary search.
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 $\bot$.
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 \( \text{iO}(P) \).

- **P:** Output \( pk_i, \text{Enc}_{pk_i}(sk_{i+1}) \).
- **For random \( t \):**
  - \( P^\{t\} \): If \( i=t \), output \( \perp \).
  - Else \( pk_i, \text{Enc}_{pk_i}(sk_{i+1}) \).

**Big Key Cycle**

If domain large enough to support OWFs

(No more cycle.)

Application: “Pure” FHE uses key cycle to handle unbounded depth.
Additional Requirements: Encryption scheme is not only IND-CPA, but pairs \((pk_i, c_i = \text{Enc}(pk_i, m))\) look pseudorandom.

\[ P^t: \text{If } i = t, \text{ output } \bot. \text{ Else output } P(i). \]

\[ Q^t: \text{If } i = t, \text{ output } \bot. \text{ Else output } Q(i). \]

P:

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

Q:

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

\text{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:

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

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

\[ P[t,u,1] : \text{Like } P[t,u,0], \text{except output for } u+1 \text{ hardwired.} \]

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

iO security

≈

Pseudo-randomness
Related Work on Circular Security vs. iO

- iO + LFHE ⇒ “Almost pure FHE”

- Negative results on circ. security (for poly-size cycles):
**Trapdoor Permutation from iO [BPW15]**

**Trapdoor:**
Key $K$ for PRF

**Public Key:**
Obfuscated program for $F_K$.

**Pseudorandom sampler:**
Output $x_{PRG(i)}$.

**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:**
Key $K$ for PRF

**Public Key:**
Obfuscated program for $F_K$.

For random $t$:

- $F_K^t$: If $z = x_t$, output $\bot$.
  Else output $F_K(z)$.

\[ x_i = (i, \text{PRF}(K, i)) \]
Other Work on iO vs. Delicate Graphs

  - 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 $\leq 1$, and a source node $s$, find some other source/sink node.
  - $\text{iO}(P_G)$ for long-line-graph indistinguishable from $\text{iO}(P'_G)$ where end-of-the-line is inaccessible.
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

Leveled FHE from iO+OWFs?

- Compact additive HE → CRHFs.
  - Let $c_1 = \text{Enc}(a)$, $c_2 = \text{Enc}(b)$.
  - $H(x_1 \| x_2) = \text{Enc}(ax_1 + bx_2)$ (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


  “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:

\[ \text{Output} = \text{Enc}(pk_i, \text{NAND}(\text{Dec}(sk_{i-1}, c_1), \text{Dec}(sk_{i-1}, c_2)); \text{PRF}(K, c_1, c_2)) \]

\[ c_1 = \text{Enc}(pk_{i-1}, x) \]
\[ c_2 = \text{Enc}(pk_{i-1}, y) \]
\[ c_3 = \text{Enc}(pk_i, \text{NAND}(x, y)) \]

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: \( \text{Enc}(\text{tpk}, m; r) \) and \( \text{Enc}(\text{tpk}, 0; r) \) have statistically identical distributions.

- Hybrid \( H_k \):
  - Use normal public keys \( \text{pk}_i \) for \( i \leq k \).
  - Trapdoor keys \( \text{tpk}_i \) 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

<table>
<thead>
<tr>
<th>Hybrid</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_{i-1,0}$</td>
<td>Output $\text{Enc}(pk_i, \text{NAND} (\text{Dec}(sk_{i-1}, c_1), \text{Dec}(sk_{i-1}, c_2)); \text{PRF}(K,c_1,c_2))$</td>
</tr>
<tr>
<td>$H_{i-1,1}$</td>
<td>Output $\text{Enc}(tpk_i, \text{NAND} (\text{Dec}(sk_{i-1}, c_1), \text{Dec}(sk_{i-1}, c_2)); \text{PRF}(K,c_1,c_2))$</td>
</tr>
<tr>
<td>$H_{i-1,2,j,1}$</td>
<td>For $j = (c_1^<em>, c_2^</em>)$, hardcode response to (i-1,j) and puncture PRF there.</td>
</tr>
<tr>
<td>$H_{i-1,2,j,2}$</td>
<td>Output $\text{Enc}(tpk_i, \text{NAND} (\text{Dec}(sk_{i-1}, c_1^<em>), \text{Dec}(sk_{i-1}, c_2^</em>)); r)$</td>
</tr>
<tr>
<td>$H_{i-1,2,j,3}$</td>
<td>Output $\text{Enc}(tpk_i, 0; r)$</td>
</tr>
<tr>
<td>$H_{i-1,2,j,4}$</td>
<td>Output $\text{Enc}(tpk_i, 0; \text{PRF}(K,j))$</td>
</tr>
</tbody>
</table>

- Trapdoor key indistinguishability
- iO security
- PPRF security
- Statistical re-randomizability
- PPRF security and iO security
L-Leveled FHE from iO + Re-rand PKE: Hybrids

<table>
<thead>
<tr>
<th>Hybrid</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_{i-1,0}$</td>
<td>Output $\text{Enc}(\text{pk}<em>i, \text{NAND}(\text{Dec}(\text{sk}</em>{i-1}, c_1), \text{Dec}(\text{sk}_{i-1}, c_2)); \text{PRF}(K,c_1,c_2))$</td>
</tr>
<tr>
<td>$H_{i-1,1}$</td>
<td>Trapdoor key indistinguishability</td>
</tr>
<tr>
<td>$H_{i-1,2,j,1}$</td>
<td>For $j = (c_1^<em>, c_2^</em>)$, hardcode response to $(i-1,j)$ and make PPRF secure</td>
</tr>
<tr>
<td>$H_{i-1,2,j,2}$</td>
<td>Output $\text{Enc}(\text{tpk}<em>i, \text{NAND}(\text{Dec}(\text{sk}</em>{i-1}, c_1^<em>), \text{Dec}(\text{sk}_{i-1}, c_2^</em>)))$</td>
</tr>
<tr>
<td>$H_{i-1,2,j,3}$</td>
<td>Statistical re-randomizability</td>
</tr>
<tr>
<td>$H_{i-1,2,j,4}$</td>
<td>Output $\text{Enc}(\text{tpk}_i, 0; \text{PRF}(K,j))$</td>
</tr>
</tbody>
</table>

**Notes:**
- Need $> 2^{2\text{len}}$ security where len is the length of a ciphertext.
- Hence need statistical re-randomizability.
Toward iO-based Multilinear Maps?
A Positive Result


- Encoding of $x$ consists of the obfuscation:

  \[ F_x : \text{Output } e(g^x, g^y) \overset{\text{def}}{=} g^{2xy} \text{ by powering by } 2x. \]

  \[ F'_x : \text{Output } g^{2xy} \text{ by powering by } 2x + \text{ord}(g). \]

- Degree of maps is unbounded-polynomial.

- Applications to non-interactive key agreement and distributed broadcast encryption schemes.

- 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

  - 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**

**SFE with Long Output**
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 $(\text{Alg}^*, \text{Explain})$ output by $\text{Comp}(1^\lambda, \text{Alg})$ it holds that $|\text{Alg}^*| \leq p(\lambda) |\text{Alg}|$.
  - **Statistical functional equivalence**: Distributions of $\text{Alg}^*(x)$ and $\text{Alg}(x)$ are statistically close for all $x$.
  - **Explainability**: Adversary $A$ has negl advantage in following game:

    
    $\begin{align*}
    &A(1^\lambda) \text{ outputs } x^* \text{ of its choice (selects target input).} \\
    &\text{(Alg}^*, \text{Explain}) \leftarrow \text{Comp}(1^\lambda, \text{Alg}). \\
    &\text{Choose uniform coins } r_0 \in \text{ and compute } y^* = \text{Alg}^*(x^*; r_0). \\
    &\text{Compute } r_1 \leftarrow \text{Explain}(x^*, y^*). \\
    &\text{Choose random bit } b \text{ and give (Alg}^*, y^*, r_b) \text{ to } A. \\
    &A \text{ tries to guess } b.
    \end{align*}$

    Explain allows simulator to generate consistent randomness.
**Explainability Compiler: Construction**

**Alg*:**

Hardwired keys: $K_1$, $K_2$, $K_3$

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

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

**Explain:**

Hardwired keys $K_2$, $K_3$

Input: $x$, $y'$ and randomness $r$

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

---

*Outputs an “encryption” of $(x,y')$*
SFE with Long Output Scenario:
- Function $f: \{0,1\}^A \times \{0,1\}^B \rightarrow \{0,1\}^t$.
- Output is long: $t \gg A$. Maybe also $t \gg B$.
- Want SFE protocol for Bob to get $f(x_A,x_B) | \in \{0,1\}^t$.

Examples:
- $x_A = $ PRF key $K$, $f(x_A,\emptyset) = PRF(K,1)$, ..., $PRF(K,L)$.
- $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**
    - \( \text{view}_{\text{Bob}} \) allows Bob to compute \( \text{PRF}(K,1), \ldots, \text{PRF}(K,L) \)
    - \( |\text{view}_{\text{Bob}}| \ll t \) if Bob is deterministic, \( |x_B| \ll t \), and \( |\text{comm}| \ll t \).
    - \( \text{view}_{\text{Bob}} \) compresses \( \text{PRF}(K,1), \ldots, \text{PRF}(K,L) \) (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 $iO(C_{K,z})$.

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

- **Hardwired:** key $K$, hash output $z$
- **Inputs:** $(i \leq L, r_i, \pi_i)$
  1. Verify decommitment $\pi_i$ 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.

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

- **Hardwired:** key $K_{sim}$, $H$ output $z$
- **Inputs:** $(i \leq L, r_i, \pi_i)$
  1. Verify decommitment $\pi_i$ 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

  - **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:**

  - Given ciphertexts $\text{Enc}(x_1), \ldots, \text{Enc}(x_n)$ and the secret key $\text{sk}_f$ for $n$-ary function $f$, one can compute $f(x_1, \ldots, x_n)$, and nothing else about the $\{x_i\}$’s.


More Powerful Protocols via iO

**Functionalities from Mmaps (not iO):**

More Powerful Protocols via iO

**ZK, WI, Concurrency:**

- Bitansky, Paneth: “ZAPs and NIWI from iO”. TCC 2015.
More Powerful Protocols via iO

Frameworks:

iO of Randomized Functionalities
(Like Circuit Garbling)
Papers Considering Randomized Functionalities

Can we obfuscate a program that flips coins internally?

If we make it pseudorandom?
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

\[ P^{\leq t}(x): \]
If \( x \leq t \), output \( P_1(x, \text{PPRF}(K, x)) \).
If \( x > t \), output \( P_2(x, \text{PPRF}(K, x)) \).

\[ \text{iO security} \]
Hardcode \( y = P_1(t, \text{PPRF}(K, t)) \).
If \( x < t \), output \( P_1(x, \text{PPRF}(K^{\{t\}}, x)) \).
If \( x > t \), output \( P_2(x, \text{PPRF}(K^{\{t\}}, x)) \).
If \( x = t \), output \( y \).

\[ \text{PPRF security} \]
Hardcode \( y = P_1(t, r_1) \).
If \( x < t \), output \( P_1(x, \text{PPRF}(K^{\{t\}}, x)) \).
If \( x > t \), output \( P_2(x, \text{PPRF}(K^{\{t\}}, x)) \).
If \( x = t \), output \( y \).

\[ P^{\leq t+1}(x): \]
If \( x < t \), output \( P_1(x, \text{PPRF}(K, x)) \).
If \( x \geq t \), output \( P_2(x, \text{PPRF}(K, x)) \).

\[ \text{iO security} \]
Hardcode \( y = P_2(t, \text{PPRF}(K, t)) \).
If \( x < t \), output \( P_1(x, \text{PPRF}(K^{\{t\}}, x)) \).
If \( x > t \), output \( P_2(x, \text{PPRF}(K^{\{t\}}, x)) \).
If \( x = t \), output \( y \).

\[ \text{PPRF security} \]
Hardcode \( y = P_2(t, r_2) \).
If \( x < t \), output \( P_1(x, \text{PPRF}(K^{\{t\}}, x)) \).
If \( x > t \), output \( P_2(x, \text{PPRF}(K^{\{t\}}, x)) \).
If \( x = t \), output \( y \).

Indistinguishable distributions
Randomized Encodings

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

- Given $C \in \mathcal{C}$ input $x$, $\text{RE}(C, x)$ 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 $\text{RE}(C, x)$ reveals nothing beyond $C(x)$.
  - Also, $\text{RE}$ is typically not only efficient, but has low parallel complexity (e.g., it’s in $\text{NC}^1$).
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

Garbling RAM Computations

  - Seminal work on garbling programs without going through circuits.


  - 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
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].
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

  - Finding Nash Eq hard assuming iO and sub-exp OWFs.

  - Separates efficient PAC learning from efficient differentially-private PAC learning assuming iO and simple primitives.

  - [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


- 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

- 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.
Thank You! Questions?