Indistinguishability
Obfuscation for all Circuits

Sanjam Garg, Craig Gentry*, Shai Halevi*, Mariana Raykova, Amit Sahai, Brent Waters

Faces in Modern Cryptography, Oct-2013
A Celebration in Honor of Goldwasser and Micali’s Turing Award

* Supported by IARPA contract number D11PC20202
Code Obfuscation

• Make programs “unintelligible” while maintaining their functionality
  ◦ Example from Wikipedia:

```perl
@P=split//,".URRUU\c8R";@d=split//,"\nrekcah xinU /
1reP rehtona tsuJ";sub p{
    @p{"r$p","u$p"}=(P,P);pipe"r$p","u$p";++$p;($q*=2)+
    =$f=!fork;map{+$P=+$P[$f^ord ($p{$_})&6];$p{$_}=/
        ^$P/ix?$P:close$_}keys%p;p;p;p;p;map{$p{$_}=~/^[P.
    .]/&& close$_}p;wait
    until?$;map{/^r/&&<$_>}%p;$=_=$d[$q];sleep
    rand(2)if\S/;print
```

• Why do it?
• How to define “unintelligible”?  
• Can we achieve it?
Why Obfuscation?

- Hiding secrets in software
  - AES encryption

Plaintext\rightarrow\text{Ciphertext}\rightarrow\text{strutpatent.com}

10/4/2013 Indistinguishability Obfuscation
Why Obfuscation?

- Hiding secrets in software

- AES encryption ➔ Public-key encryption
Why Obfuscation?

- Hiding secrets in software

1,2d0
< The Way that can be told of is not the eternal Way;
< The name that can be named is not the eternal name
4c2,3
< The Named is the mother of all things.
---
> The named is the mother of all things.
11a11,13
> They both may be called deep and profound.
> Deeper and more profound,
> The door of all subtleties!

- Distributing software patches
Why Obfuscation?

- Hiding secrets in software

- Distributing software patches while hiding vulnerability
Why Obfuscation?

- Hiding secrets in software
  - Uploading my expertise to the web
Why Obfuscation?

- Hiding secrets in software
  - Uploading my expertise to the web without revealing my strategies

Game of Go

```perl
@P=spli\t/,,\\.URRUU\c8R''@d=split/,,\\nrekcah xinU / IreP rehtona tsuJ''\sub P{
  @p"r$p","u$p"=(P,P);pipe"r$p","u$p";++p;($q*=2)+=$f=!fork;map{$P=$P[$f^ord ($p{$_})&6];$p{$_}=/
  ^$P/ix?$P:close$_}keys$p;p;p;p;map{$p{$_}=~/^\P./&\& close$_}%p;wait until$?;map{/^r/&#<$_>}%p;$_=$d[$q];sleep rand(2)if\S;/print
```
Contemporary Obfuscation

- Used fairly widely in practice
- Mostly as an art form
  - Some rules-of-thumb, sporadic tool support
  - Relies on human ingenuity, security-via-obscurity
  - “At best, obfuscation merely makes it time-consuming, but not impossible, to reverse engineer a program” (from Wikipedia)
- Can it be done the Goldwasser-Micali way?
  - Definitions, constructions, concrete assumptions
  - Question addressed 1st by Barak et al. in 2001 [B+01]
Defining Obfuscation

- An efficient public procedure $O(\ast)$
  - Everything is known about it
  - Except the random coins that it uses
- Takes as input a program $C$
  - E.g., encoded as a circuit
- Produce as output another program $C'$
  - $C'$ computes the same function as $C$
  - $C'$ at most polynomially larger than $C$
  - $C'$ is “unintelligible”
    - Okay, defining this is tricky
What’s “Unintelligible”?

- What we want: cannot do much more with $C'$ than running it on various inputs
  - At least: If $C$ depends on some secrets that are not readily apparent in its I/O, then $C'$ does not reveal these secrets

- [B+01] show that even this is impossible:
  - **Thm:** If PRFs exist, then there exists PRF families $F = \{f_s\}$, for which it is possible to recover $s$ from any circuit that computes $f_s$.
    - These PRFs are *unobfuscatable*
What’s “Unintelligible”? 

- Okay, some functions are bad, but can we get $O()$ that does “as well as possible” on every function? 
- [B+01] suggested the weaker notion of “indistinguishability obfuscation” ($iO$) 
  - Gives the “best-possible” guarantee [GR07] 
  - It turns out to suffice for many applications (examples in [GGH+13, SW13,…])
Indistinguishability Obfuscation

**Def:** If $C_1, C_2$ compute the same function (and $|C_1| = |C_2|$) then $O(C_1) \approx O(C_2)$

- Indistinguishable even if you know $C_1, C_2$

**Note:** Inefficient iO is always possible

- $O(C) = \text{lexicographically } 1^{st} \text{ circuit computing the same function as } C$

- Canonicalization is inefficient (unless P=NP)
Best-Possible Obfuscation

Some circuit $C$

Indist. Obfuscation

Best Obfuscation

$C(x)$

$\approx$

Computationally Indistinguishable

Indist. Obfuscation

Padding

Some circuit $C$

$C(x)$
Many Applications of iO

- AES ➔ public key encryption [GGH+13, SW13]

- Witness encryption: Encrypt $x$ so only someone with proof of Riemann Hypothesis can decrypt [GGSW13]

- Functional encryption: Noninteractive access control [GGH+13], $\text{Dec}(\text{Key}_y, \text{Enc}(x)) \Rightarrow F(x, y)$

- Many more (all of them this year)…

- One notable thing iO doesn’t give us (yet): Homomorphic Encryption (HE)
Beyond $iO$

- For very few functions, we know how to achieve stronger notions than $iO$
  - “Virtual Black Box” (VBB)
- Point-functions / cryptographic locks
  - $f_{a,b}(x) = \begin{cases} b & \text{if } x = a \\ \perp & \text{otherwise} \end{cases}$
  - [C97, CMR98, LPS04, W05]
  - Many extensions, generalizations [DS05, AW07, CD08, BC10, HMLS10, HRSV11, BR13]
Aside: Obfuscation vs. HE

Obfuscation:
- $F \rightarrow$ Obfuscation
- $F \rightarrow F(x)$
- $x \rightarrow F(x)$
- Result in the clear

Encryption:
- $F \rightarrow$ Encryption
- $F \rightarrow F(x)$
- $x \rightarrow F(x)$
- $x \rightarrow F(x)$
- Result encrypted
OUR CONSTRUCTION
Obfuscating Arbitrary Circuits

- A two-step construction
  1. Obfuscating “shallow circuits” (NC$^1$)
     - This is where the meat is
     - Using multilinear maps
     - Security under a new (ad-hoc) assumption
  2. Bootstrapping to get all circuits
     - Using homomorphic encryption with NC$^1$ decryption
     - Very simple, provable, transformation
NC¹ Obfuscation $\Rightarrow$ P Obfuscation

If $\pi$ describes homomorphic evaluation that takes $x, F$ to $c$, then use $sk$ to decrypt $c$

Encrypted-result $c$

+ eval transcript $\pi$

$F(x)$

CondDec
NC¹ Obfuscation $\rightarrow$ P Obfuscation

**Homomorphic Encryption**

- **F**
- **X**
- **F(X)**

**Encrypted result $C$**

- +
- **eval transcript $\pi$**

**NC¹ Circuit Obfuscation**

- Only this part

**Output of P obfuscator**

CondDec
Conditional Decryption with $iO$

- We have $iO$, not “perfect” obfuscation
- But we can adapt the CondDec approach
  - We use *two* HE secret keys
iO for CondDec $\rightarrow$ iO for All Circuits

$\pi, x$, and two ciphertexts
$c_0 = \text{Enc}_{\text{PK0}}(F(x))$ and
$c_1 = \text{Enc}_{\text{PK1}}(F(x))$

$\pi$, $x_i$’s, and two ciphertexts
$c_0 = \text{Enc}_{\text{PK0}}(F(x))$ and
$c_1 = \text{Enc}_{\text{PK1}}(F(x))$

$\pi$, $x$’s, and two ciphertexts
$c_0 = \text{Enc}_{\text{PK0}}(F(x))$ and
$c_1 = \text{Enc}_{\text{PK1}}(F(x))$

$\text{CondDec}_{F,\text{SK0}}(\cdot, \ldots, \cdot)$

$\text{CondDec}_{F,\text{SK1}}(\cdot, \ldots, \cdot)$

$F(x)$ if $\pi$ verifies

$F(x)$ if $\pi$ verifies
Analysis of Two-Key Technique

- 1st program has secret $SK_0$ inside (but *not* $SK_1$).
- 2nd program has secret $SK_1$ inside (but *not* $SK_0$).
- But programs are indistinguishable.
- So, neither program “leaks” either secret.

- Two-key trick is very handy in iO context.
- Similar to Naor-Yung ’90 technique to get encryption with chosen ciphertext security
NC$^1$ OBFUSCATION
Outline of Our Construction

- Describe Circuits as Branching Programs (BPs) using Barrington’s theorem [B86]
- Randomized BPs (RBPs) a-la-Kilian [K88]
- Encode RBPs “in the exponent” using multilinear maps [GGH13,CLT13]
- Modifications to defeat attacks
  - Multiplicative bundling against ”partial evaluation” and “mixed input” attacks
  - Defenses against “DDH attacks”, “rank attacks”
Oblivious Branching Programs

- A specific way of describing a function
- Length-\(m\) BP with \(n\)-bit input is a sequence
  \((j_1, A_{1,0}, A_{1,1}), (j_2, A_{2,0}, A_{2,1}), \ldots, (j_m, A_{m,0}, A_{m,1})\)
  \(j_i \in \{1, \ldots, n\}\) are indexes, \(A_{i,b}\)'s are matrices
- Input \(x = (x_1, \ldots, x_n)\) chooses matrices \(A_{i,x_{j_i}}\)
  - Compute the product \(P_x = \prod_{i=1}^{m} A_{i,x_{j_i}}\)
  - \(F(x) = 1\) if \(P_x = I\), else \(F(x) = 0\)
This length-9 BP has 4-bit inputs
(Oblivious) Branching Programs

- This length-9 BP has 4-bit inputs
(Oblivious) Branching Programs

- This length-9 BP has 4-bit inputs

Multiply the chosen 9 matrices together
- If product is $I$ output 1. Otherwise output 0.
Barrington’s Theorem [B86]

- $F$ computable by depth-$d$ circuit $\Rightarrow F$ computable by a BP of length $4^d$
  - With constant-dimension matrices
- Corollary: every function in NC$^1$ has a polynomial-length BP
  - Recall: NC$^1$ = O(log n)-depth circuits
Oblivious BP Evaluation [K88]

- Alice has $x$. Bob has $y$. They want Bob to get $F(x, y)$
  - They start with a $BP = \{(j_i, A_{i,0}, A_{i,1})\}_{i=1}^m$ for $F$

- Randomized BP Generation
  - Alice chooses random matrices $R_1, \ldots, R_m$, set $R_0 = R_m$
  - $RBP = \{(j_i, B_{i,0} = R_{i-1}A_{i,0}R^{-1}_i, B_{i,1} = R_{i-1}A_{i,1}R^{-1}_i)\}_{i=1}^m$

- Matrix Collection
  - Alice sends matrices for her input $\{B_{i,x_{j_i}} : i \leq |x|\}$
  - Bob gets matrices for his input via OT

- Evaluation of Randomized BP
  - $R_i$’s and their inverses cancel, $R_0, R_m^{-1}$ cancel if $P = I$

- Randomized BP gives Alice perfect privacy
Kilian’s Protocol ➔ BP-Obfuscation?

- RBP for $F_x(y) = F(x, y)$ with the $x$ part fixed
  - Bob gets $B_{i,x_{j_i}}$ as in Kilian, but both $B_{i,b}$’s for $y$
  - Evaluates randomized BP in usual way, choosing appropriate $B_{i,0}$ or $B_{i,1}$ for the $y$-parts.

- Biggest problems:
  - Perfect privacy is lost once we give both $B_{i,b}$’s
  - Partial evaluation attacks: Adversary computes partial product of matrices from positions $i_1$ to $i_2$, makes comparisons.
  - Mixed Input attacks: Adversary computes matrix product that does not respect the BP structure.
Multilinear Maps to Hide Matrices

- Recall cryptographic $d$-multilinear map:
  - Groups $G_1, ..., G_d$ of order $p$, generators $g_1, ..., g_d$
  - Computable maps $e_{ij}: G_i \times G_j \rightarrow G_{i+j}$ for $i + j \leq d$
  - Multi-linearity: $e_{ij}(g_i^a, g_j^b) = g_{i+j}^{ab}$ for all $a, b$

- Cryptographic hardness:
  - DL analog: hard to recover $a$ from $g_i^a$
  - Multilinear-DDH: Given $g_1^{a_i} \in G_1$ for $d + 1$ random $a_i$’s, hard to distinguish $g_1^{a_1} \cdot \ldots \cdot g_d^{a_{d+1}}$ from random in $G_d$
  - Etc.

- [GGH13, CLT13] don’t exactly give this
  - But it’s close enough for our purposes
Multilinear Maps to Hide Matrices

- Encode the $B_{i,b}$’s in the exponent, $g_1^{B_{i,b}}$
  - Matrix is encoded element-wise
- Can use the maps $e_{ij}$’s to multiply them
  - Given $g_i^M, g_j^N$, compute $\tilde{e}_{ij}(g_i^M, g_j^N) = g_{i+j}^{MN}$
  - From $\{g_1^{B_{i,b_i}}\}_{i=1..m}$, can compute $g_m^P = \prod_i B_{i,b_i}$
- Then we can check if $P = I$
- Are the $B_{i,b}$’s really hidden?
“Partial Evaluation” Attacks

- Evaluate the program on two inputs $y, y'$, but only use matrices between steps $i_1, i_2$, $P = \prod_{i=i_1}^{i_2} B_{i,y_{j_i}}$, $P' = \prod_{i=i_1}^{i_2} B_{i,y'_{j_i}}$
  - Check if $P = P'$

- Roughly, you learn if in the computations of the circuits for $F(y), F(y')$, you have the same value on some internal wire
“Mixed Input” Attack

- Inconsistent matrix selection:
  - Product includes $B_{i_1,0}$ and $B_{i_2,1}$, but these two steps depend on the same input bit (i.e., $j_{i_1} = j_{i_2}$)

- Roughly, you learn what happens when fixing some internal wire in the circuit of $F(y)$
  - Fixing the wire value to 0, or to 1, or copying value from another wire, …
“Multiplicative Bundling”

- Obfuscator uses two randomized BPs
  - “Main BP” computing $F_x(y) = F(x, y)$
  - “Dummy BP’” computing $c(y) = 1$
    - Same length and $j_i$-assignments as the BP for $F_x$
    - All the $A'_{i,b}$’s are the identity
    - Independent randomizer matrices $R_i'$

- For every step $i$ choose random scalars $\alpha_{i,0}, \alpha_{i,1}, \alpha'_{i,0}, \alpha'_{i,1} \leftarrow Z_p$ under the constraint:
  - For every input bit position $j$ and value $b \in \{0,1\}$
    $\prod_{i: j_i = j} \alpha_{i,b} = \prod_{i: j_i = j} \alpha'_{i,b}$
“Multiplicative Bundling”

- Obfuscator outputs
  \[
  \begin{align*}
  B_{i,b} &= \alpha_{i,b} \cdot R_{i-1} A_{i,b} R_i^{-1} \\
  B'_{i,b} &= \alpha'_{i,b} \cdot R'_{i-1} I R_i'^{-1}
  \end{align*}
  \]

- To evaluate \( F(y) \), compute the products (in the exponent)
  \[
  P = \prod_{i=1}^{m} B_{i,y_j_i} \quad \text{and} \quad P' = \prod_{i=1}^{m} B'_{i,y_j_i}
  \]

- If \( F(y) = 1 \) then \( P = P' = \alpha \cdot I \)
  - For some constant \( \alpha \) (the same for \( P, P' \))

- “Partial evaluation” & “mixed input” attacks yield matrices that differ by a multiplicative constant
  - Rather than identical matrices
DDH Attacks

- Identifying matrices (in the exponent) that differ by a multiplicative constant is DDH

- But we can solve DDH using MMAPs:
  - Given \( \begin{pmatrix} g_i^a & g_i^b \\ g_i^c & g_i^d \end{pmatrix}, \begin{pmatrix} g_i^{a'} & g_i^{b'} \\ g_i^{c'} & g_i^{d'} \end{pmatrix} \) (with \( 2i \leq d \)),
  - check \( e_{i,i} (g_i^a, g_i^{b'}) = e_{i,i} (g_i^{a'}, g_i^b) \) etc.

- Not out of the woods yet…
More Attacks: Determinant & Rank

- Use MMAPs to compute determinant
  - E.g., given \( g^A = \begin{pmatrix} g^a_1 & g^b_1 \\ g^c_1 & g^d_1 \end{pmatrix} \) compute 
    \[ e_{1,1} (g^a_1, g^d_1)/e_{1,1} (g^b_1, g^c_1) = g^{\text{det}(A)}_2 \]

- For matrices of dimension \( \leq d \), can check if they are singular
  - Use projections to compute rank

- Not sure how to use for actual attack, but it is something to look for
Fixing DDH, Rank Attacks

- One option (also used in [BR13b]) is to switch to “asymmetric maps”
  - Just like XSDH for bilinear maps, DDH can potentially be hard in the different groups, even though you have pairing
  - A little awkward to define in the multilinear setting, so will not do it here
Fixing DDH, Rank Attacks

- Or embed in higher-dimension matrices
  - Set $D_{i,b} = \begin{pmatrix} \$ & \cdots & \$ \\ & \ddots & \$ \\ & & \alpha_{i,b} A_{i,b} \end{pmatrix}$
  - Then $B_{i,b} = R_{i-1} D_{i,b} R_i^{-1}$

- Matrix rank $> d$, too high to compute
- $'$s are independent between all the matrices $D_{i,0}, D_{i,1}, D'_{i,0}, D'_{i,1}$
  - Matrices in attacks no longer differ just by a multiplicative constant factor
How To Evaluate?

- We have \( P = \prod_{i=1}^{m} B_{i,y_{i}} = R_0 DR_m^{-1} \), and similarly \( P' = R_0' D'R_m'^{-1} \)
  - \( D' \) diagonal, and if \( F_x(y) = 1 \) then so is \( D \)
  - But top entries on the diagonal are random, different between \( D, D' \)
- Add pairs of “bookend” vectors
  - \( u = sR_0^{-1}, v = R_m t, u' = s'R_0'^{-1}, v' = R_m' t' \)
  - \( s, t, s', t' \) have 0’s to eliminate the $’\)s in \( D, D' \)
  - Compute \( r = uPv = sDt, r' = u'P'v' = s'D't' \), check that \( r = r' \)
Summary of BP-Obfuscation

- “Main BP” for $F_x(y)$, “dummy” for $c(y) = 1$
- Multiplicative bundling with $\alpha_{i,b}, \alpha'_{i,b}$
- Embed $\alpha_{i,b}A_{i,b}$’s in higher-degree $D_{i,b}$’s
- Multiply by randomizers $B_{i,b} = R_{i-1}D_{i,b}R_i^{-1}$
- Add “bookend” vectors $u = sR_0^{-1}, v = R_m t$
- Encode everything with $(m + 2)$-MMAPs

- To evaluate: compare products of “main”, “dummy”, output 1 if they match.
Is This Indistinguishable?

- It’s plausible...
- Don’t know to distinguish $O(F_{x1})$, $O(F_{x2})$, except by finding $y$ s.t. $F_{x1}(y) \neq F_{x2}(y)$
- We can prove that some “generic attacks” do not work
- But no simple hardness assumption that we can reduce to
  - This is important future work
Open Problems

• Better underlying hardness assumptions
• Faster constructions
  ◦ Complexity of our construction is horrendous
• Better notions
  ◦ $iO$ is okay for some things, not others
  ◦ Certainly does not capture our intuition of what an obfuscator is
    • Doesn’t even capture the intuition of what the current construction achieves
• Applications
  ◦ The sky is the limit…
Thank You

Questions?