# 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

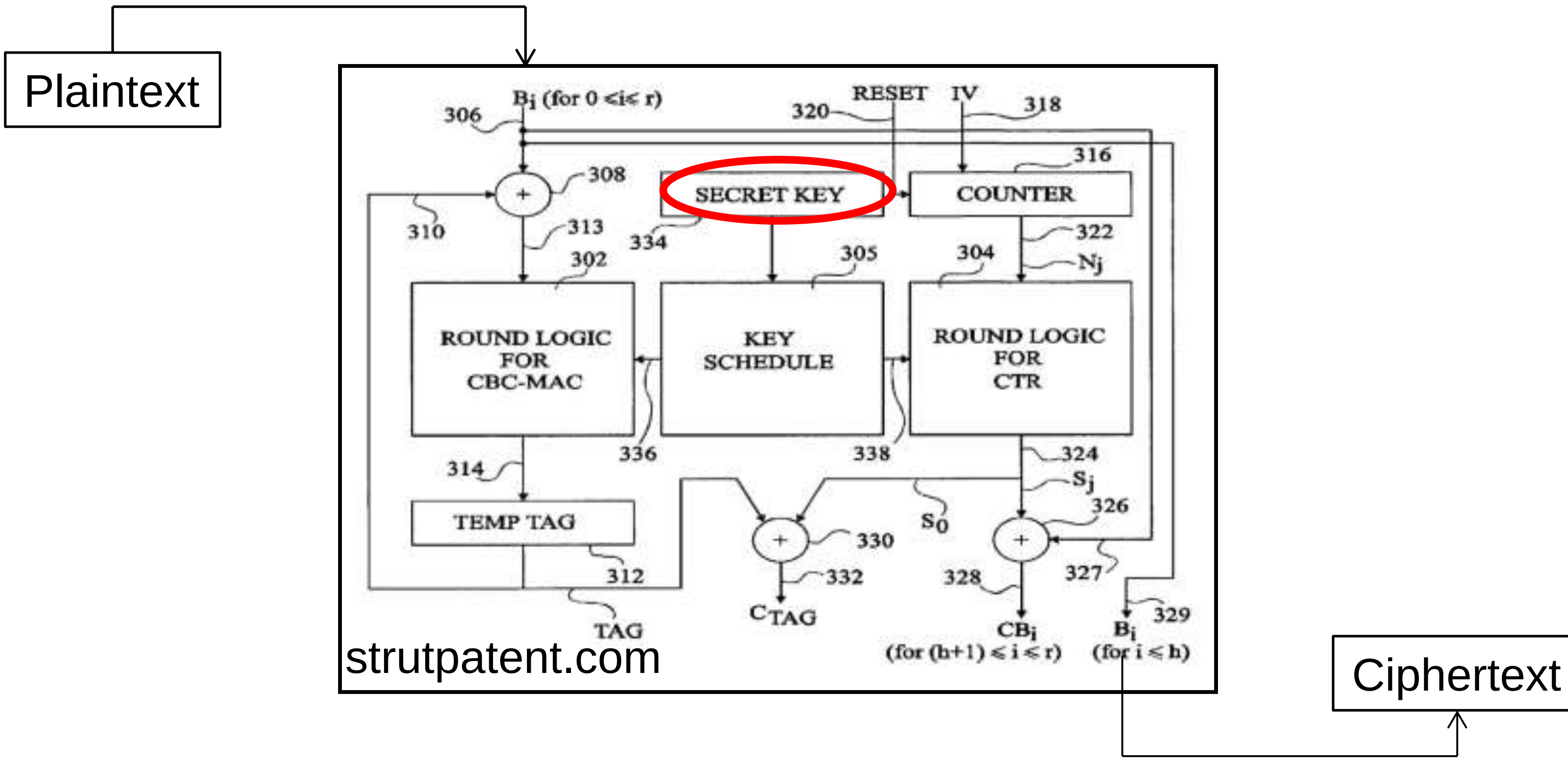
- Make programs “unintelligible” while maintaining their functionality
  - Example from Wikipedia:

```
@P=split//,".URRUU\c8R";@d=split//,"\nrekcah xinU /
lreP 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$_}keys%p}p;p;p;p;p;map{$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?

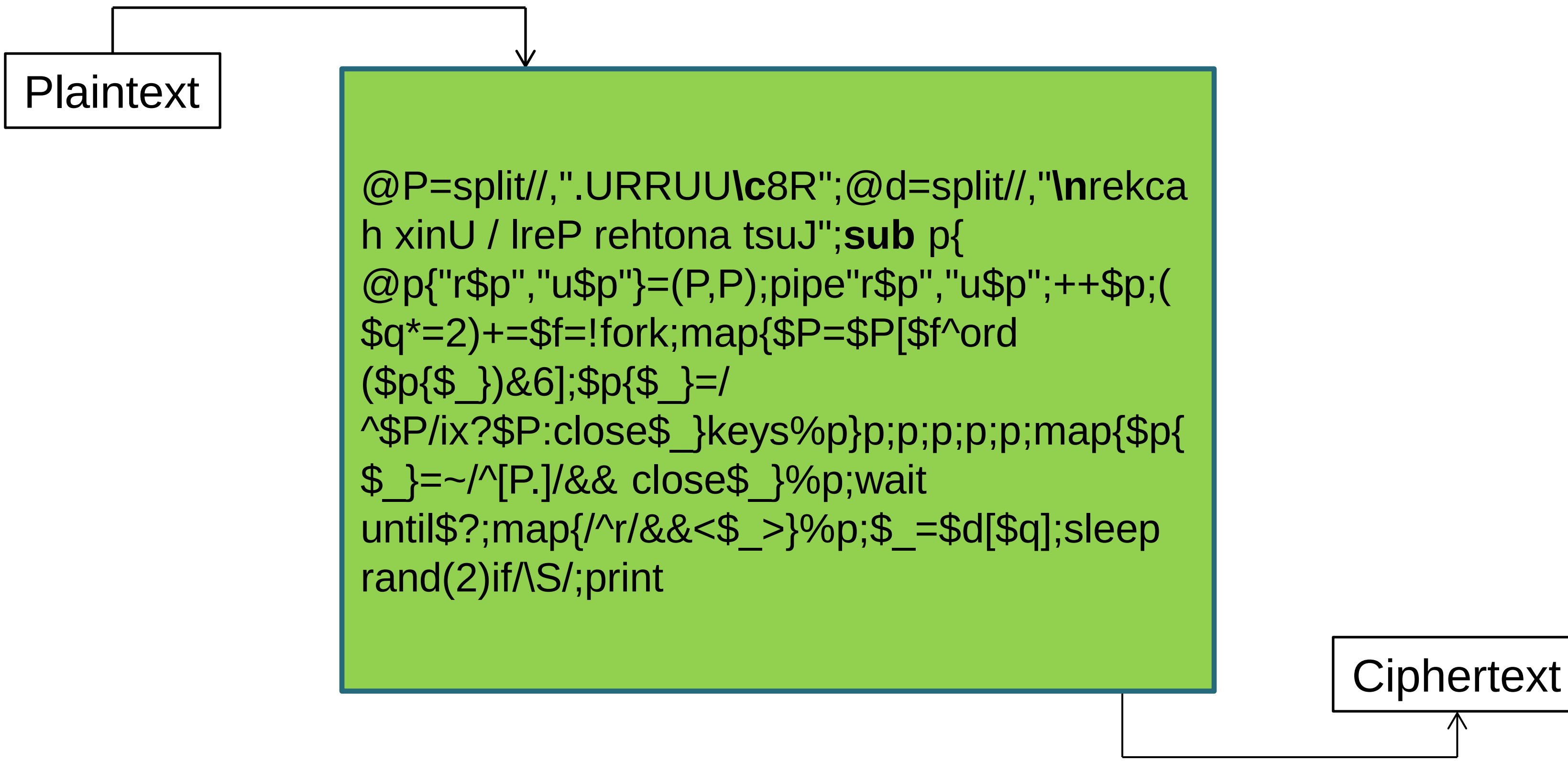
- Hiding secrets in software




  - AES encryption

# Why Obfuscation?

- Hiding secrets in software




  - AES encryption ➔ Public-key encryption

# Why Obfuscation?

- Hiding secrets in software

Vulnerable program

```
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!
```

Patched program

  - Distributing software patches

# Why Obfuscation?

- Hiding secrets in software

Vulnerable program

```
@P=split//,".URRUU\c8R";@d=split//,"\nrekcah xinU /
lreP 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$_}keys%p}p;p;p;p;p;map{$p{$_}=~/^[P
.]/&& close$_}%p;wait
until$?;map{/^r/&&<$_>}%p;$_=$d[$q];sleep
rand(2)if/\S/;print
```

Patched program

  - Distributing software patches while hiding vulnerability

# Why Obfuscation?

- Hiding secrets in software

Game of Go

http://www.arco-iris.com/George/images/game_of_go.jpg


Next move

  - Uploading my expertise to the web

# Why Obfuscation?

- Hiding secrets in software

Game of Go

```
@P=split//,".URRUU\c8R";@d=split//,"\nrekcah xinU
/ lreP 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$_}keys%p}p;p;p;p;p;map{$p{$_}=~/^
[P.]/&& close$_}%p;wait
until$?;map{/^r/&&<$_>}%p;$_=$d[$q];sleep
rand(2)if/\S/;print
```

Next move

  - Uploading my expertise to the web without revealing my strategies

# 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($*$)
  - 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*

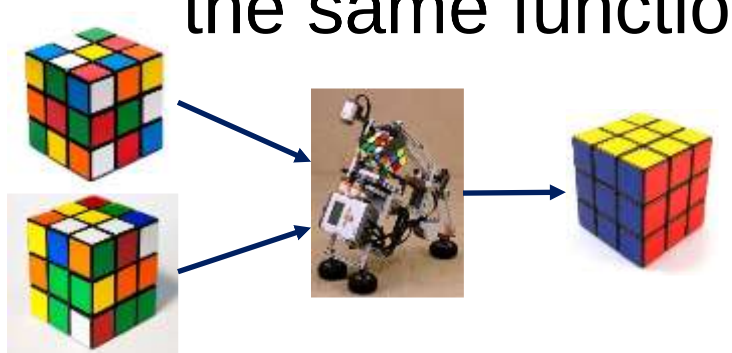
# What's "Unintelligible"?

- Okay, some function 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

- **<u>Def:</u>** 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 $C_1, C_2$

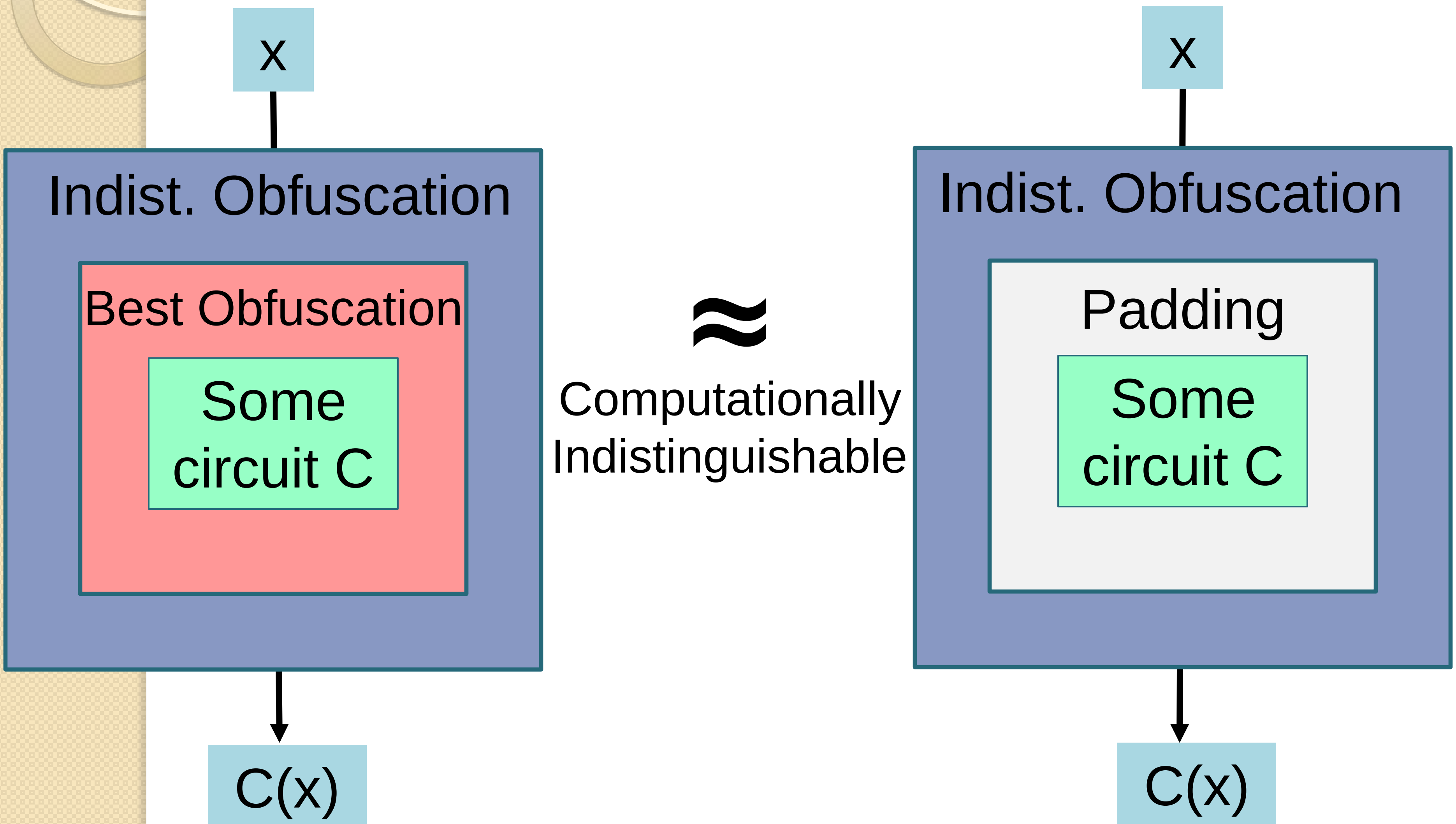
- Note: Inefficient *iO* is always possible
  - $O(C)$ = lexicographically 1st circuit computing the same function as $C$



(canonical form)

  - Canonicalization is inefficient (unless P=NP)

# Best-Possible Obfuscation



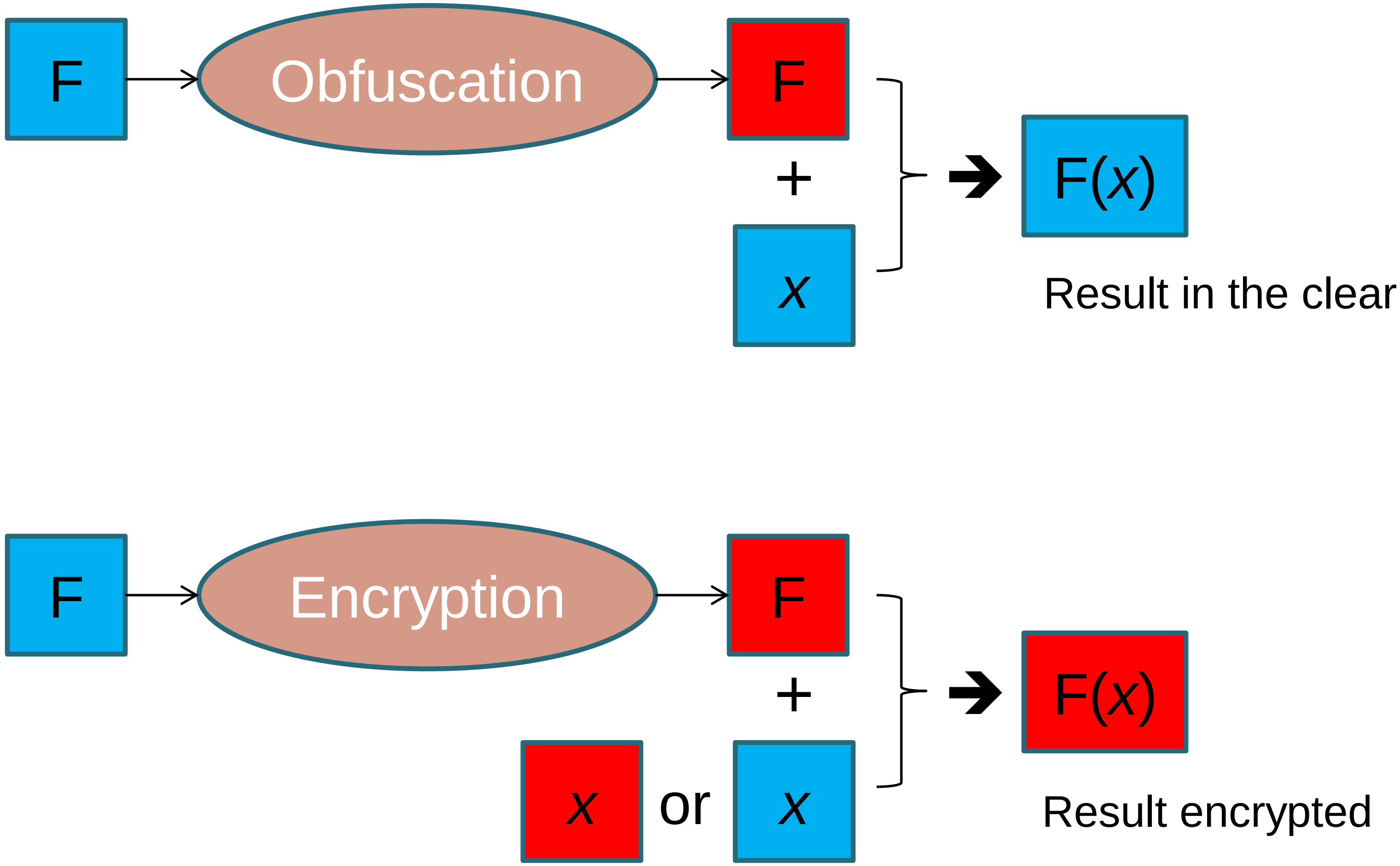
# Many Applications of iO

- 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], Dec(Key$_y$, Enc($x$))➔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
  - $f_{a,b}(x) = \begin{cases} b & if\ x = a \\ \bot & otherwise \end{cases}$
  - [C97, CMR98, LPS04, W05]
  - Many extensions, generalizations [DS05, AW07, CD08, BC10, HMLS10, HRSV11, BR13]

# Aside: Obfuscation vs. HE



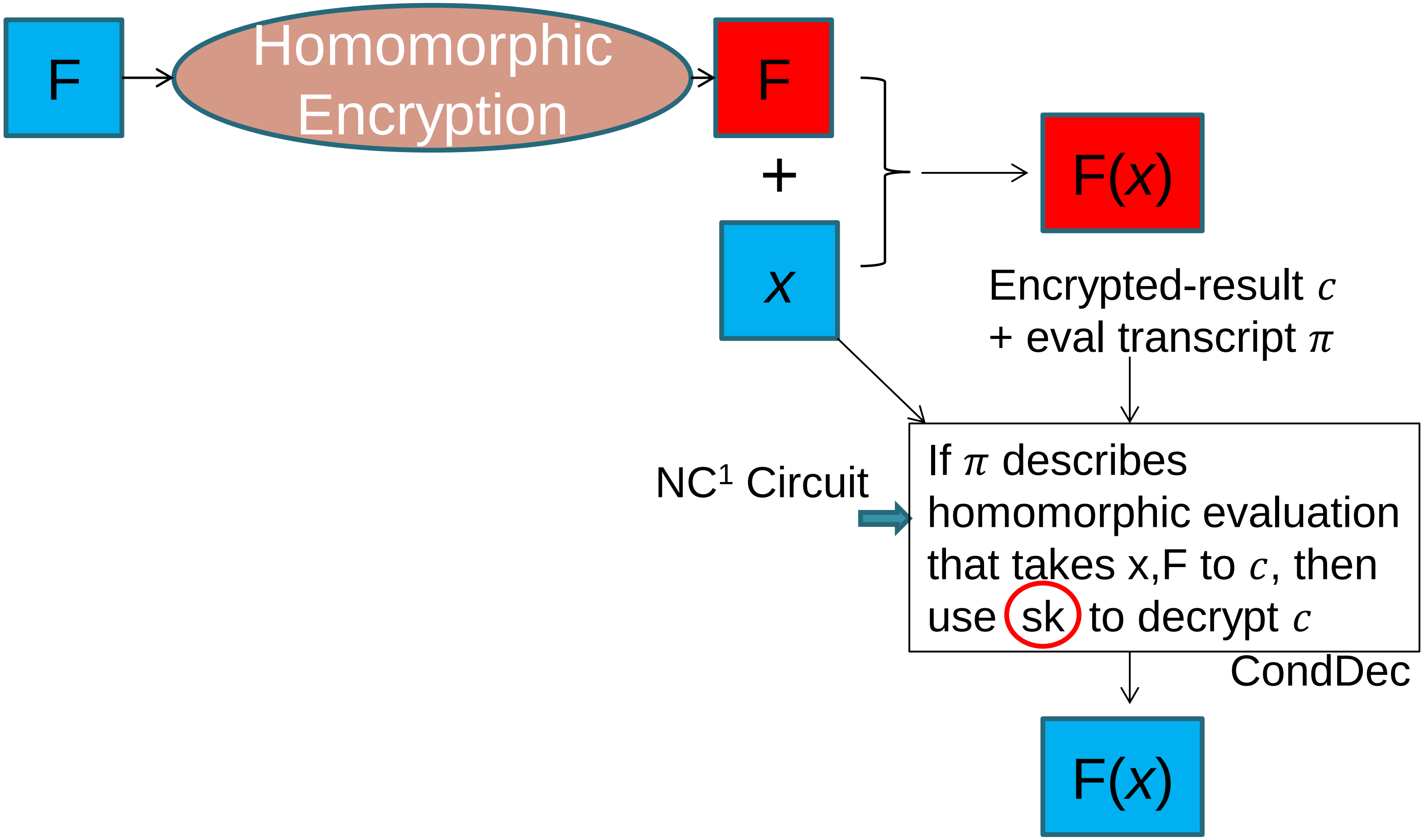
# OUR CONSTRUCTION

# Obfuscating Arbitrary Circuits

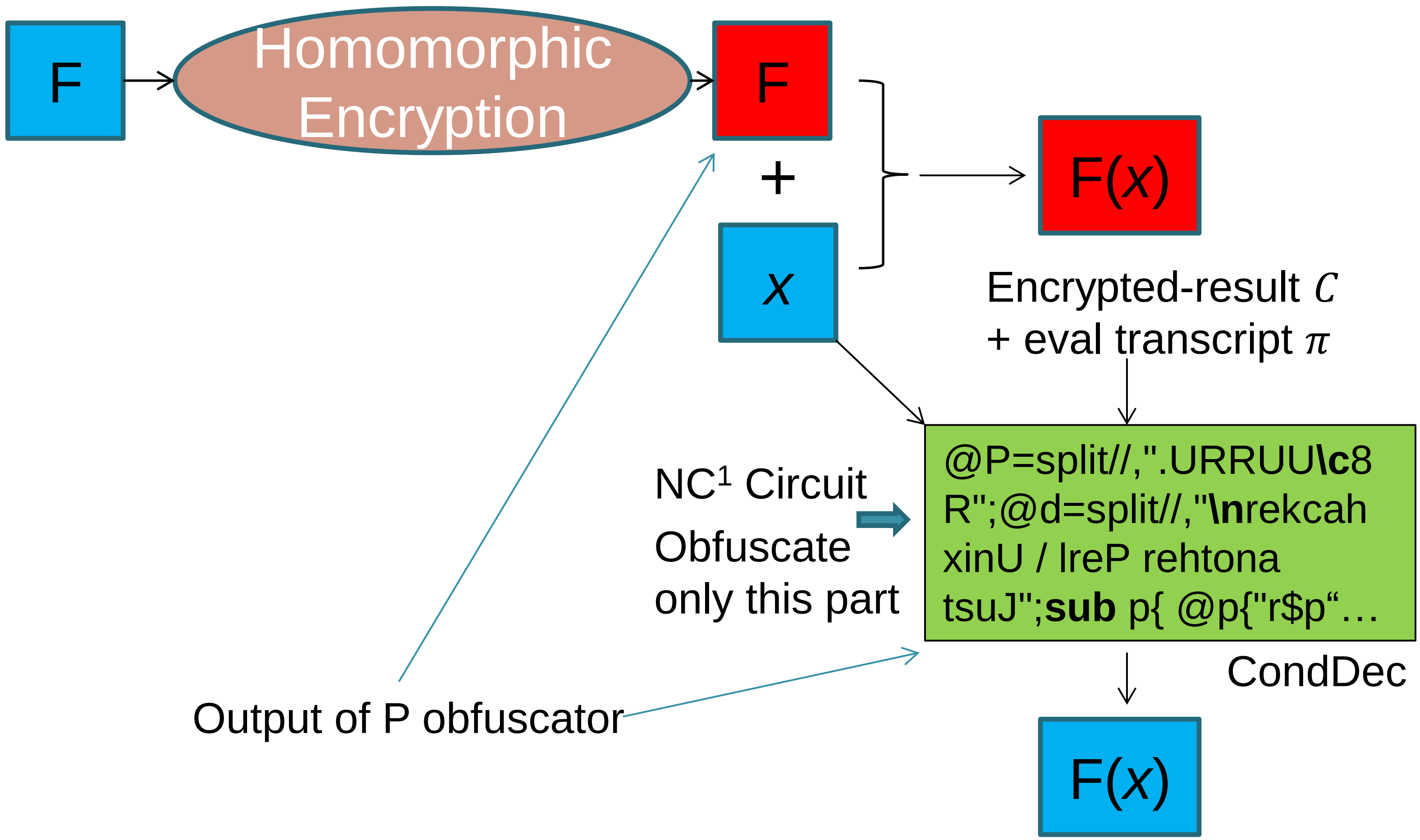
- A two-step construction

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

# $NC^1$ Obfuscation→P Obfuscation



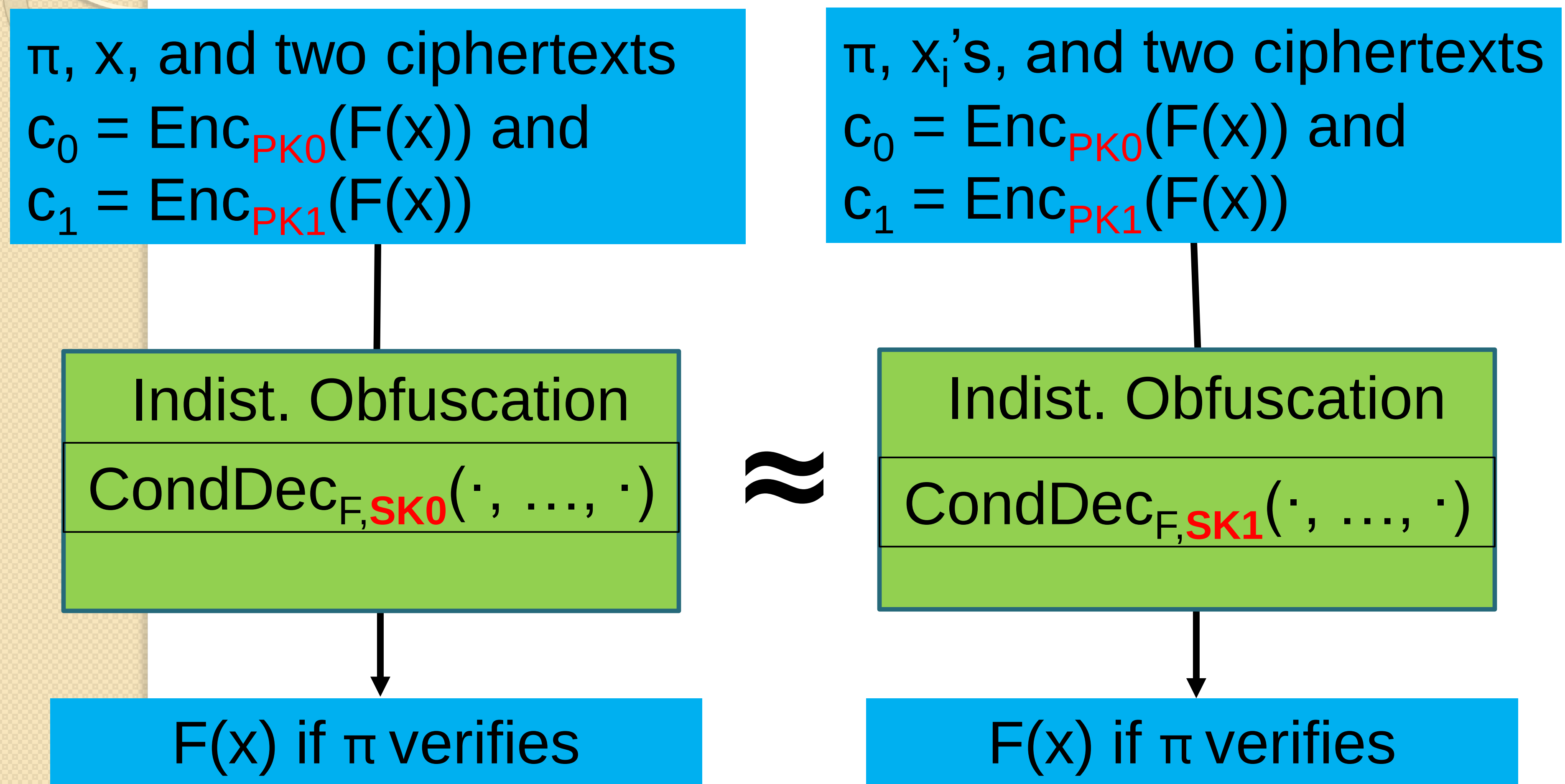
# NC¹ Obfuscation→P Obfuscation



# 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 → iO for All Circuits



# 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

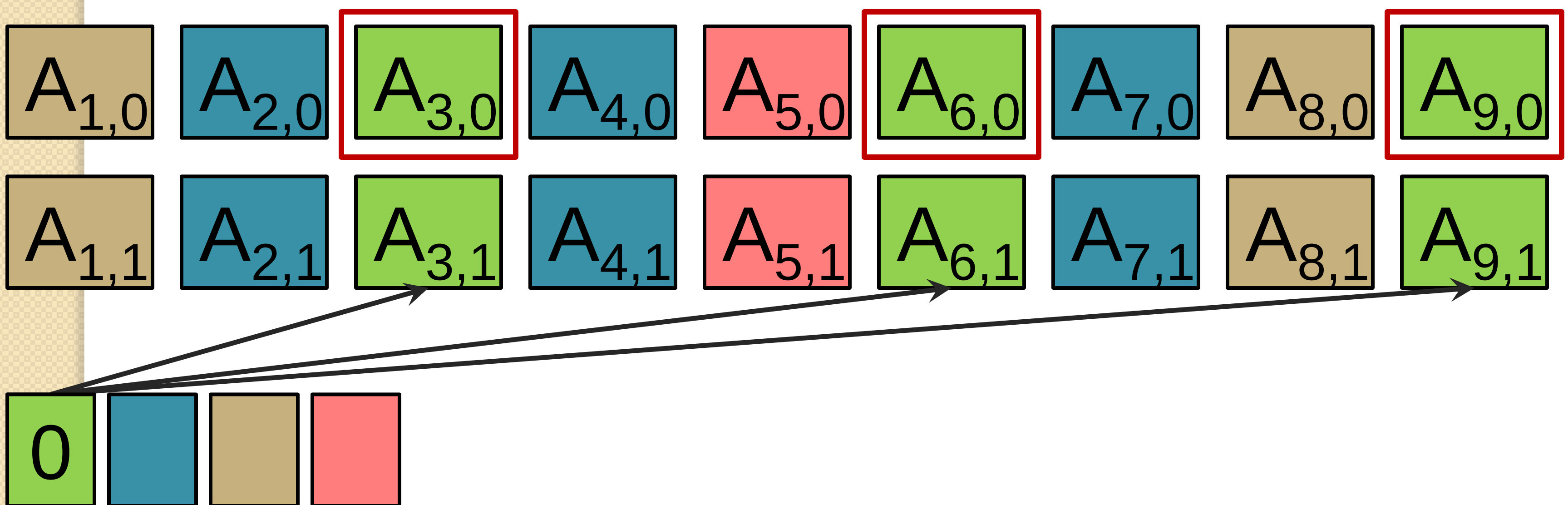
- Describe Circuits as Branching Programs (BPs) using Barrington's theorem [B86]
- Randomized BPs (RBPs) a-la-Kilian [K88]
- Encode RBPs "in the exponent" using multilinear maps [GGH13,CLT13]
- Modifications to defeat attacks
  - Multiplicative bundling against "partial evaluation" and "mixed input" 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 $F(x) = 0$

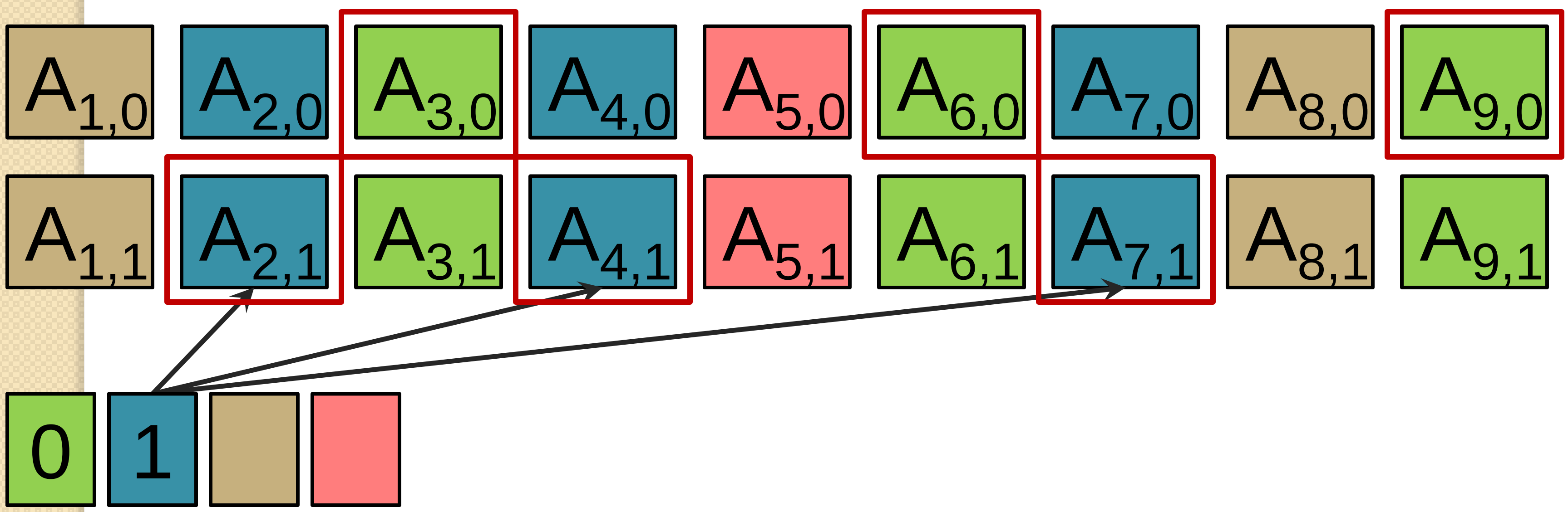
# (Oblivious) Branching Programs

- This length-9 BP has 4-bit inputs



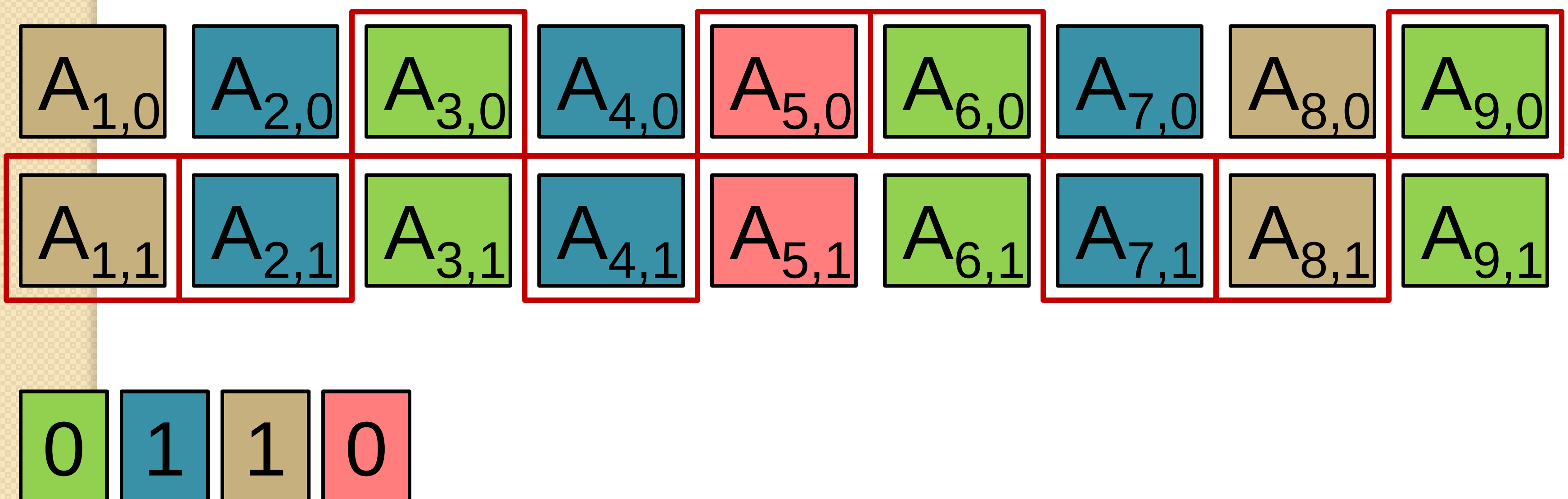
# (Oblivious) Branching Programs

- 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 ➔ $F$ 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 $R_1, \dots, R_m$, set $R_0 = R_m$
  - RBP=$\{(j_i,\ \ B_{i,0} = R_{i-1} A_{i,0} R_i^{-1},\ B_{i,1} = R_{i-1} A_{i,1} R_i^{-1})\}_{i=1}^{m}$
- 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 $G_1, \dots, G_d$ of order $p$, generators $g_1, \dots, g_d$
  - Computable maps $e_{ij}: G_i \times G_j \to G_{i+j}$ for $i + j \leq d$
  - Multi-linearity: $e_{ij}\left(g_i^a, g_j^b\right) = g_{i+j}^{ab}$ 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 $g_d^{a_1 \cdot \dots \cdot a_{d+1}}$ 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 $\tilde{e}_{ij}\left(g_i^M, g_j^N\right) = g_{i+j}^{MN}$
  - From $\left\{g_1^{B_{i,b_i}}\right\}_{i=1..m}$, can compute $g_m^P = g_m^{\prod_i B_{i,b_i}}$
- 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

$$\left\{B_{i,b} = \alpha_{i,b} \cdot R_{i-1} A_{i,b} R_i^{-1}\right\}_{i,b}$$

$$\left\{B'_{i,b} = \alpha'_{i,b} \cdot R'_{i-1}\ I\ {R'_i}^{-1}\right\}_{i,b}$$

- To evaluate $F(y)$, compute the products (in the exponent) $P = \prod_{i=1}^{m} B_{i,y_{j_i}}$ and $P' = \prod_{i=1}^{m} {B'}_{i,y_{j_i}}$
- 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 $e_{i,i}\left(g_i^a, g_i^{b'}\right) = e_{i,i}\left(g_i^{a'}, g_i^b\right)$ etc.
- Not out of the woods yet…

# More Attacks: Determinant & Rank

- Use MMAPs to compute determinant
  - E.g., given $g^A = \begin{pmatrix} g_1^a & g_1^b \\ g_1^c & g_1^d \end{pmatrix}$ compute

$$e_{1,1}(g_1^a, g_1^d)/e_{1,1}(g_1^b, g_1^c) = g_2^{\det(A)}$$

- 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 $D_{i,b} = \begin{pmatrix} \$ & & & \\ & \ddots & & \\ & & \$ & \\ & & & \alpha_{i,b} A_{i,b} \end{pmatrix}$
  - 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 $P = \prod_{i=1}^{m} B_{i,y_{j_i}} = R_0 D R_m^{-1}$, and similarly $P' = R'_0 D' R'^{-1}_m$
  - $D'$ 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
  - $\boldsymbol{u} = \boldsymbol{s} R_0^{-1}, \boldsymbol{v} = R_m \boldsymbol{t}$, $\boldsymbol{u}' = \boldsymbol{s}' \mathrm{R}'^{-1}_0$, $\boldsymbol{v}' = R'_m \boldsymbol{t}'$
  - $\boldsymbol{s}, \mathbf{t}, \boldsymbol{s}', \mathbf{t}'$ have 0’s to eliminate the $’s in $D, D'$
  - Compute $r = \boldsymbol{u} P \boldsymbol{v} = \boldsymbol{s} D \boldsymbol{t}$, $r' = \boldsymbol{u}' P' \boldsymbol{v}' = \boldsymbol{s}' D' \boldsymbol{t}'$, 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 $\boldsymbol{u} = \boldsymbol{s} R_0^{-1}, \boldsymbol{v} = R_m \boldsymbol{t}$
- 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?