Obfuscation

Lecture 24

Obfuscation

- The art & science of making programs "unintelligible"

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

from International Obfuscated C Code Contest 1988 (via Wikipedia)

  - The program should be fully functional
  - It may contain secrets that shouldn't be revealed to the users (e.g., signature keys) — any more than executing it

Obfuscation

- For protecting proprietary algorithms, for crippling functionality (until license bought), for hiding potential bugs, for hardwiring cryptographic keys into apps, for reducing the need for interaction with a trusted server (say for auditing purposes), ...
- Several heuristic approaches to obfuscation exist
  - All break down against serious program analysis

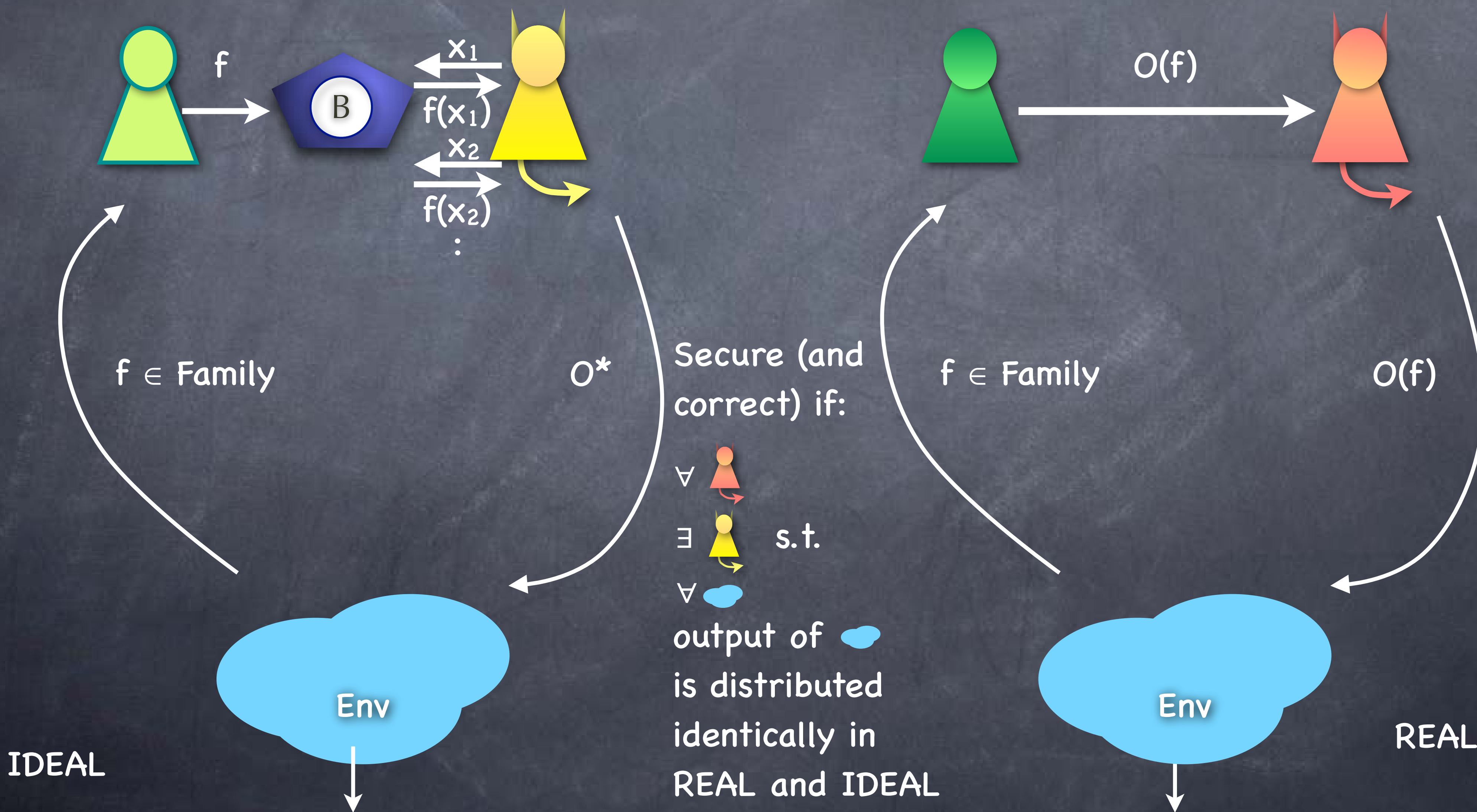
Cryptographic Obfuscation

- Obfuscation using cryptography?
  - Need to define a security notion
  - Constructions which meet the definition under computational hardness assumptions
- Cryptography using obfuscation
  - If realized, obfuscation can be used to instantiate various other powerful cryptographic primitives
  - Example: PKE from SKE. Obfuscate the SKE encryption program with the key hardwired (plus a PRF for generating randomness from the plaintext), and release as public-key
    - Or FE: Encrypt message x with a CCA-secure PKE. Function key SK_f is a program that decrypts, computes f(x) and outputs it.

Defining Obfuscation: First Try

Note: Considers only corrupt receiver

Too strong! Requires family to be <u>learnable</u> from black-box access



Secure (and correct) if:

∀

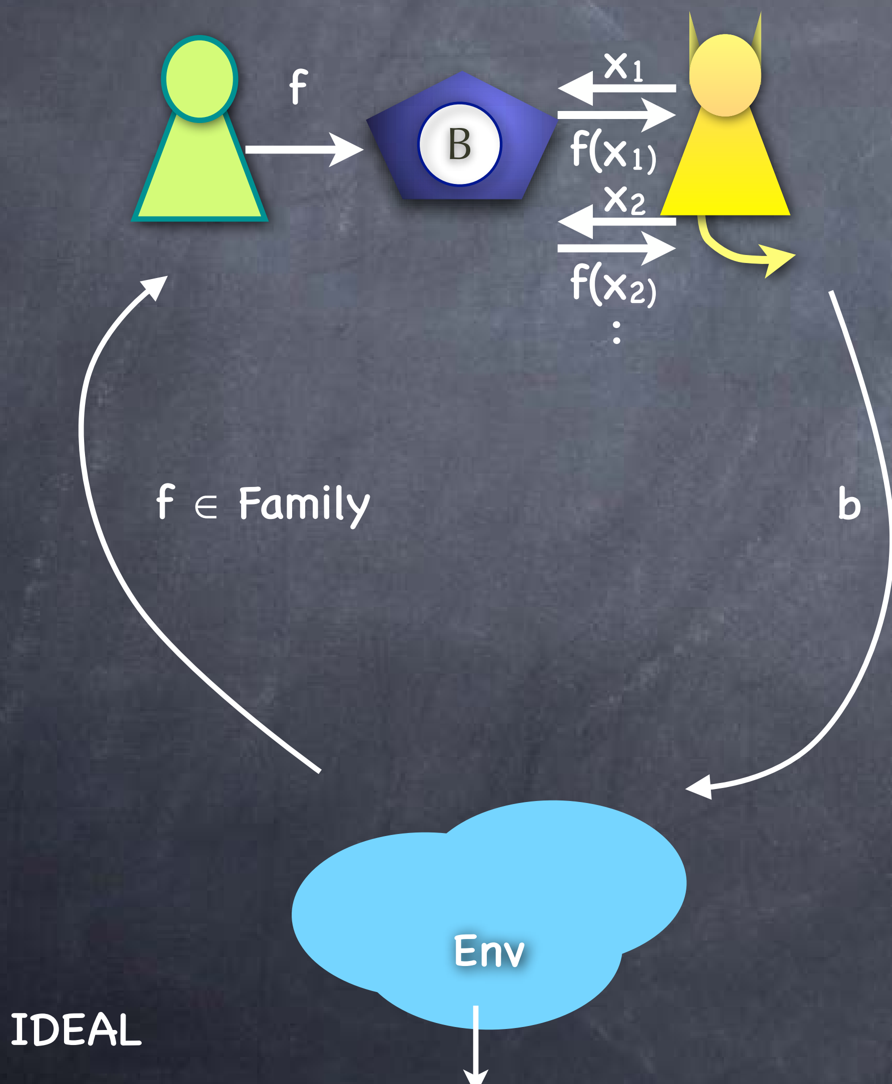
∃ s.t.

∀

output of is distributed identically in REAL and IDEAL

Defining Obfuscation: First Try

Note: Considers only corrupt receiver



Virtual Black-Box (VBB) Obfuscation

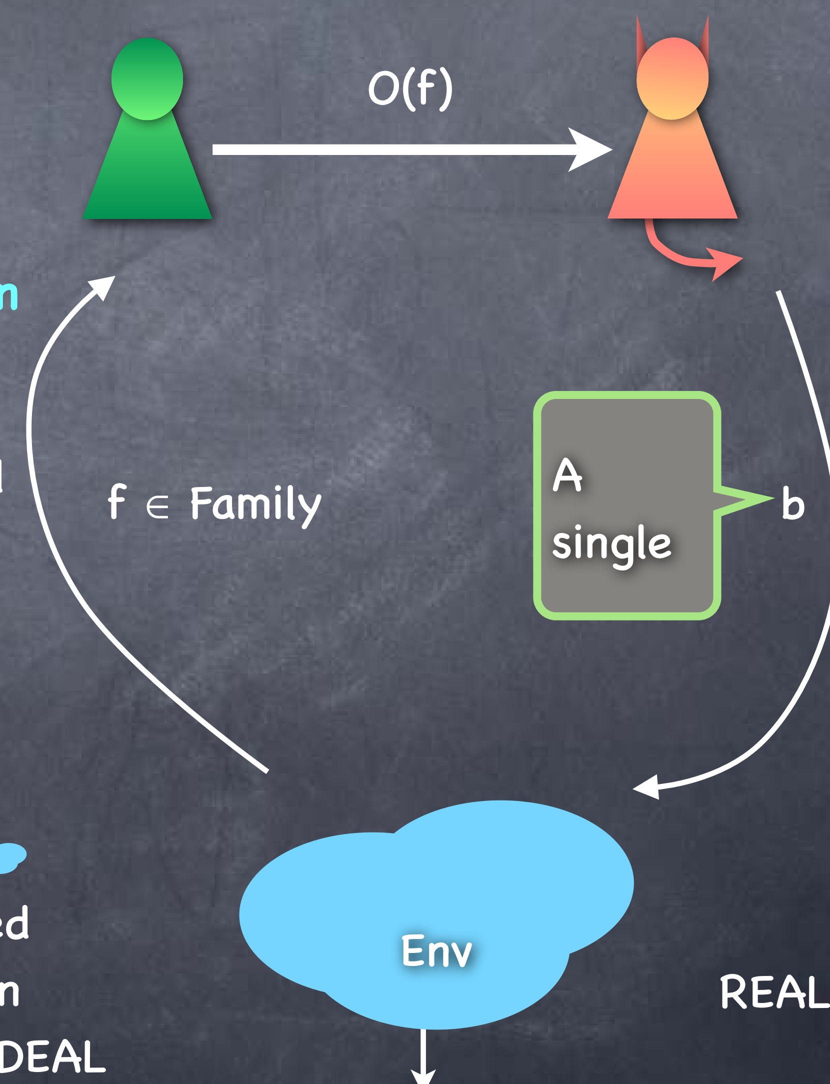
Secure (and correct) if:

∀

∃ s.t.

∀

output of is distributed identically in REAL and IDEAL



Impossibility of Obfuscation

- VBB obfuscation is impossible in general
- Explicit example of an unobfuscatable function family
  - Idea: program which when fed its own code (even obfuscated) as input, outputs secrets
  - Programs $P_{\alpha,\beta}$ with secret strings α and β:
    - If input is of the form $(0,\alpha)$ output β
    - If input is of the form $(1,P)$ for a program P, run P with input $(0,\alpha)$ and if it outputs β, output (α,β)
  - When $P_{\alpha,\beta}$ is run on its own (obfuscated) code, it outputs (α,β). Can learn, e.g., first bit of α. In the ideal world, need to guess!

Possibility of Obfuscation

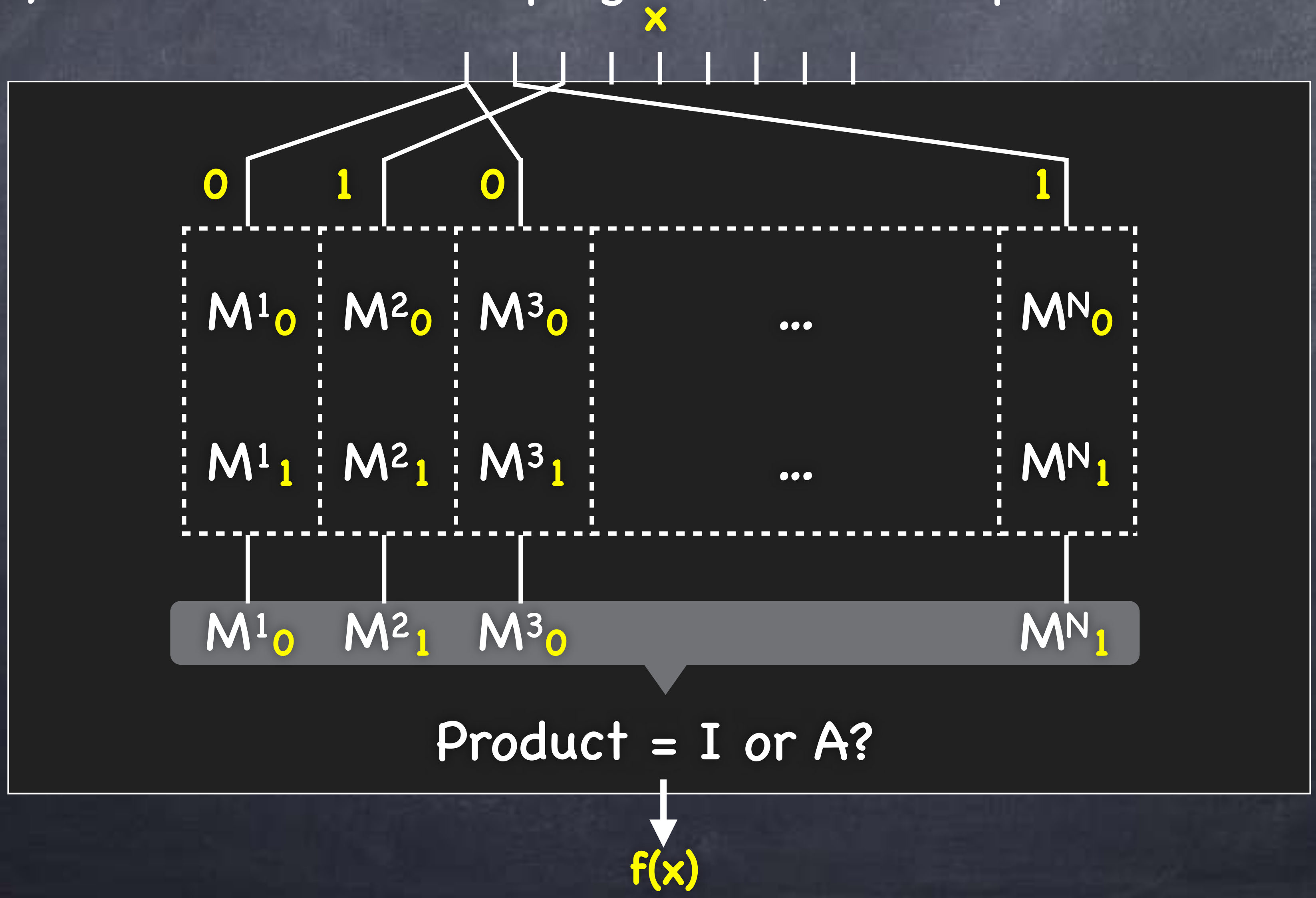
- Hardware assisted
- For simple function families
  - e.g., Point functions (from perfectly one-way permutations)
  - But general "low complexity classes" are still unobfuscatable (under cryptographic assumptions)
- In idealized models like generic group model (coming up)
- For weaker definitions like iO (coming up)

Obfuscation from Multi-Linear Map

- Recall bilinear pairing: e: $G_1 \times G_2 \rightarrow G_T$ such that $e(g_1^a, g_2^b) = g_T^{ab}$
- Extension to more than 2 groups
  - Let T = {1,...,k}. For each non-empty subset $S \subseteq T$, a group G_S.
  - $e(g_{S_1}{}^a, g_{S_2}{}^b) = g_{S_3}{}^{ab}$, where $S_1 \cap S_2 = \varnothing$ and $S_3 = S_1 \cup S_2$
- An element a encoded in G_S (S not hidden): $[a]_S$ (think g_S^a)
  - Need a private key for encoding (think of keeping g_S secret)
- Following public operations:
  - $[a]_S + [b]_S \rightarrow [a+b]_S$ (note that S is the same for all)
  - $[a]_{S_1} * [b]_{S_2} \rightarrow [ab]_{S_1 \cup S_2}$ where $S_1 \cap S_2 = \varnothing$ and $S_3 = S_1 \cup S_2$
  - Zero-Test($[a]_T$) checks if a=0 or not (note: only for set T)
- Generic Group Model heuristic: No other operation possible!
- Obfuscation uses a "matrix program" representation of the function

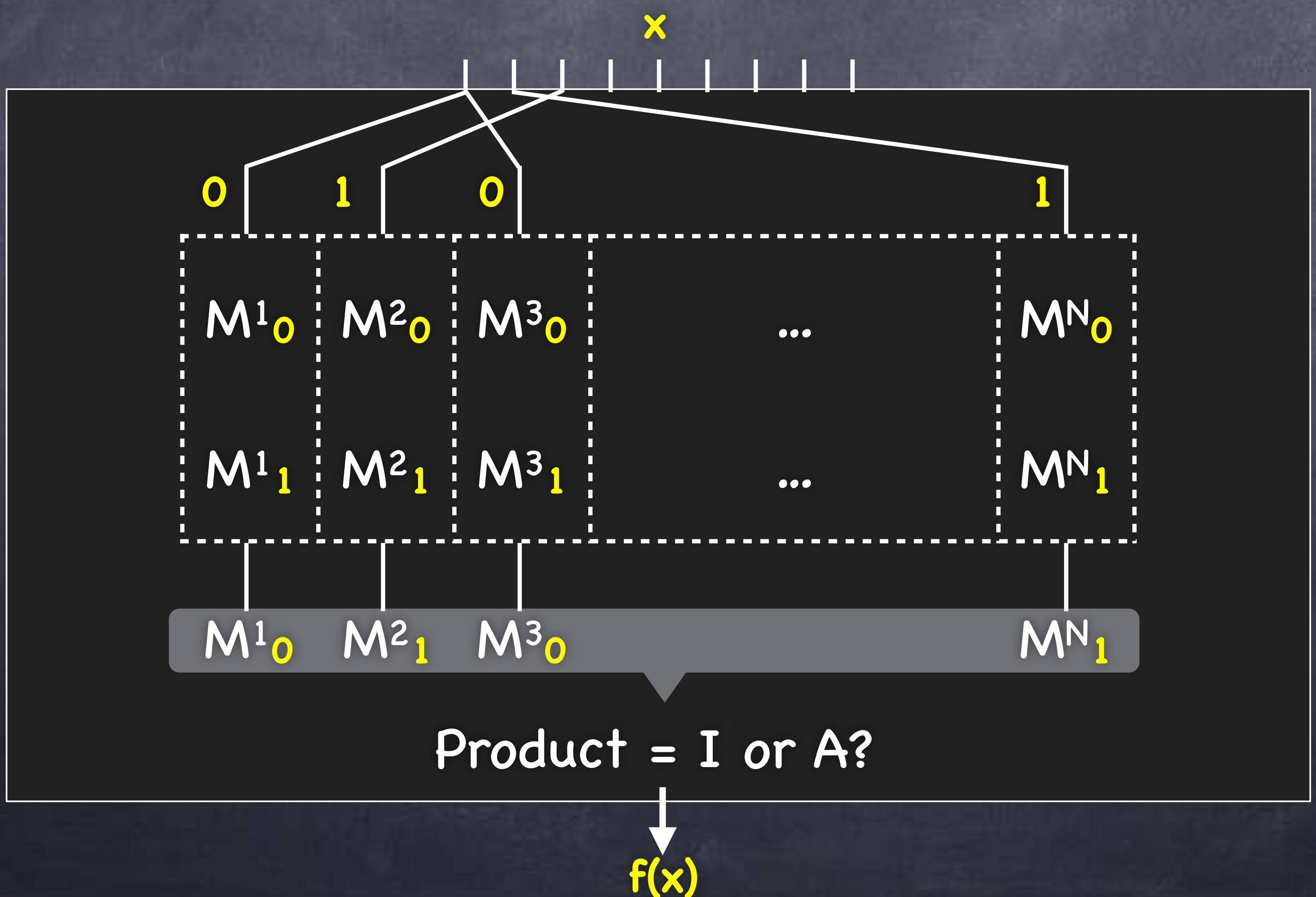
Matrix Programs

- $f : \{0,1\}^n \to \{0,1\}$ using a set of 2N w×w matrices (N = poly(n))
- Barrington's Theorem: "Shallow" circuits (NC^1 functions) have polynomial-sized matrix programs (with 5x5 permutation matrices)



Matrix Programs

- Idea: Encode matrices s.t. only valid matrix multiplications and final check (I or A?) can be carried out (for any x)



Obfuscation from Multi-Linear Map

- Such encodings are known based on multi-linear maps
  - Using generic model multi-linear map, this yields Virtual Black-Box obfuscation for polynomial-sized matrix programs
    - And hence for NC^1 circuits from Barrington's theorem
    - Can "bootstrap" to all polynomial-sized circuits/ polynomial-time computable functions, assuming FHE with decryption in NC^1
  - Instantiating obfuscation constructions using concrete hardness assumptions on these candidates yields weaker flavours of obfuscation
- Several candidate multi-linear maps proposed [GGH'13, CLT'13,...]
  - Initial candidates broken...

Flavours of Obfuscation

- VBB Obf.
- Adaptive DIO
- Differing Inputs Obf.
- PC Differing Inputs Obf.
- Indistinguishability Obf.
- XIO

VGB Obf.

Not an exhaustive list!

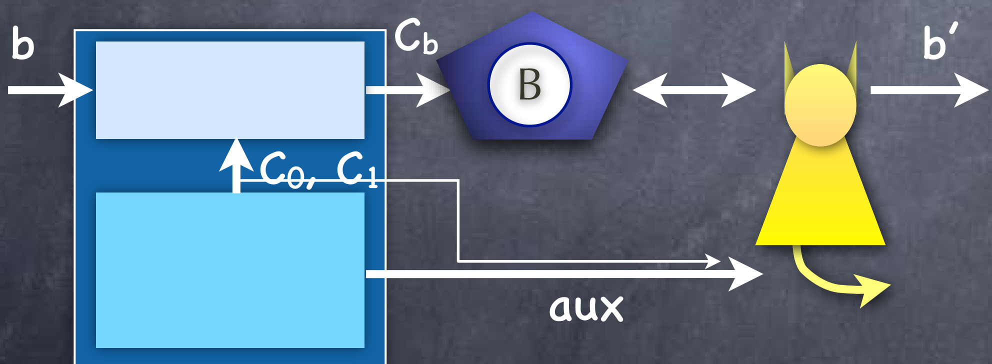
IND-PRE Security

Different variants of the definition in this framework

Typically C_0, C_1 given to the adversary (part of aux)

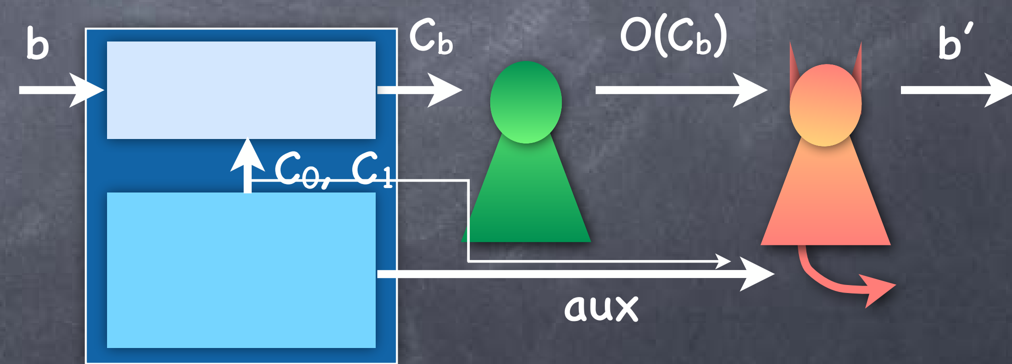
is IDEAL-Hiding if

$\forall$ PPT $\Pr[b'=b] = \frac{1}{2} \pm$ negl.



is REAL-Hiding if

$\forall$ PPT $\Pr[b'=b] = \frac{1}{2} \pm$ negl.



IND-PRE secure if $\forall$ PPT in <u>Test-Family</u>

IDEAL-hiding $\Rightarrow$ REAL-hiding

IDEAL

REAL

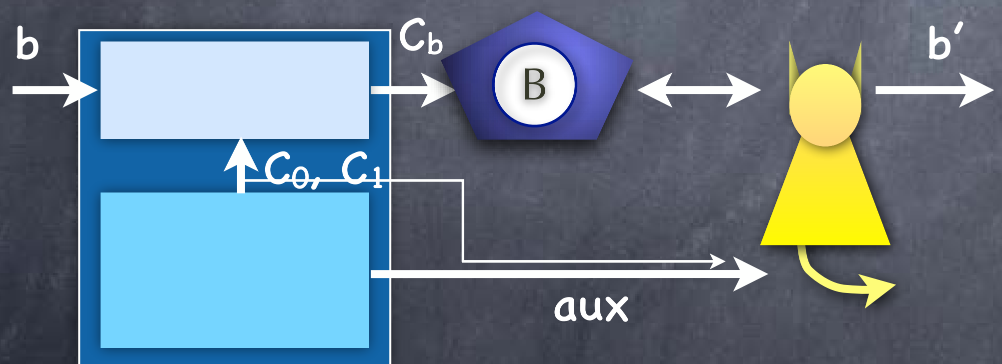
Indistinguishability Obf. (iO)

Test picks functionally equivalent C_0, C_1 (hardwired into it)

Guaranteed to be IDEAL-hiding

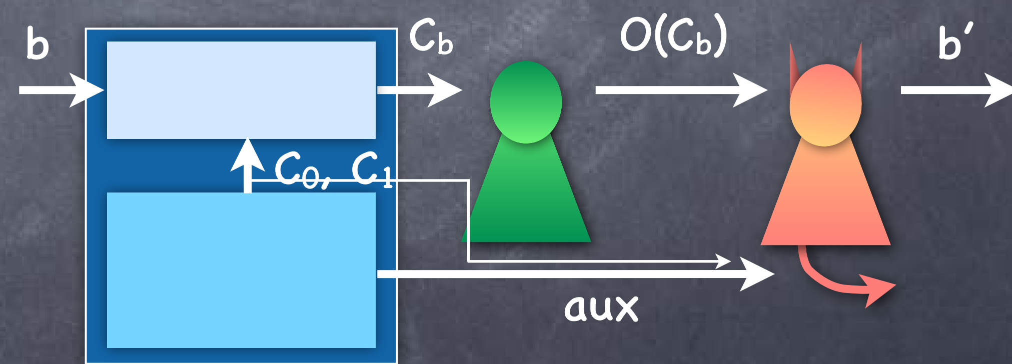
is IDEAL-Hiding if

$\forall$ PPT Pr[b′=b] = ½ ± negl.



is REAL-Hiding if

$\forall$ PPT Pr[b′=b] = ½ ± negl.



iO if $\forall$ PPT in iO Test-Family

IDEAL-hiding $\Rightarrow$ REAL-hiding

IDEAL

REAL

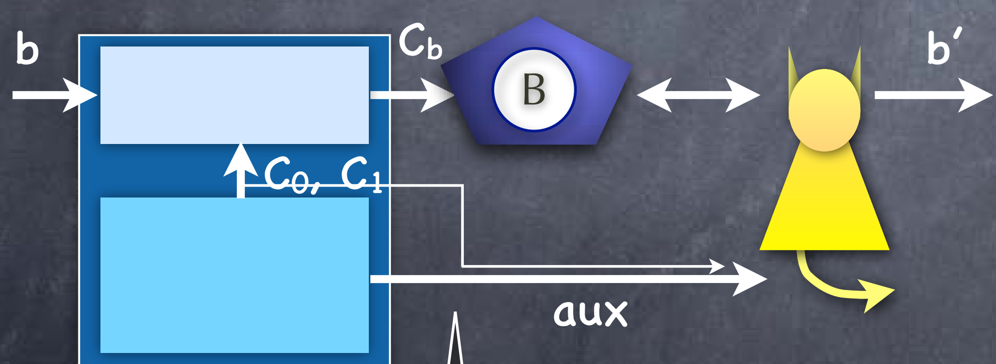
Differing Input Obf.

C_0, C_1 need not be functionally equivalent

To be not IDEAL-hiding, need a PPT which can find a "differing input"

is IDEAL-Hiding if

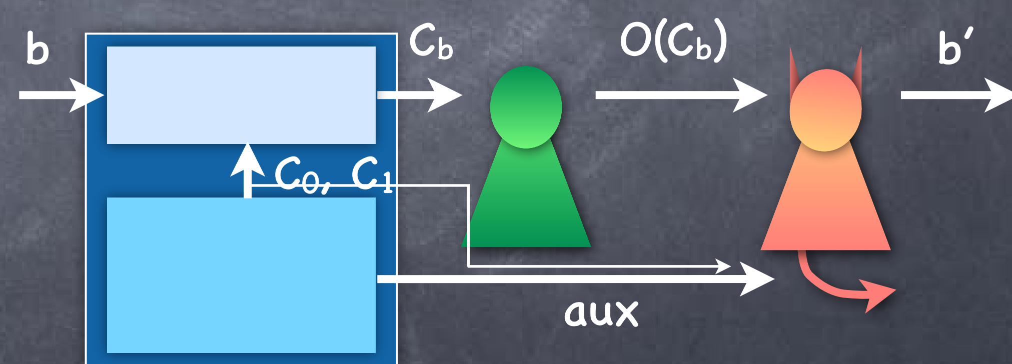
∀ PPT $\Pr[b'=b] = 1/2 \pm \text{negl.}$



Adaptive DIO allows 2-way interaction

is REAL-Hiding if

∀ PPT $\Pr[b'=b] = 1/2 \pm \text{negl.}$



DIO if ∀ PPT in DIO Test-Family

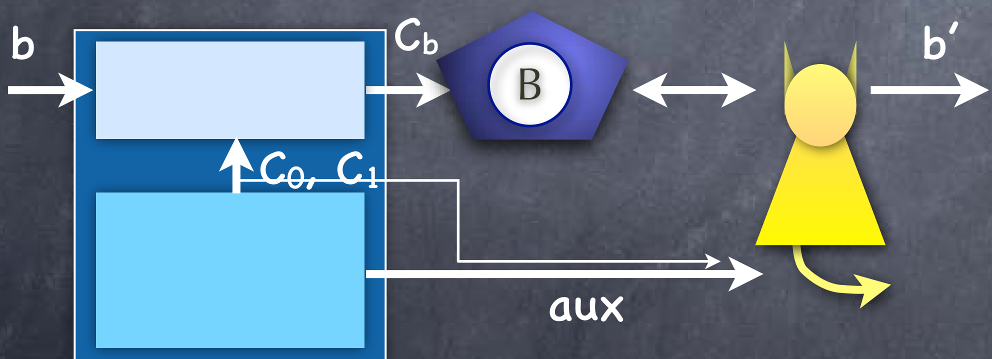
IDEAL-hiding ⇒ REAL-hiding

Public-Coin DIO

Test as in DIO, but aux includes all the randomness used by Test

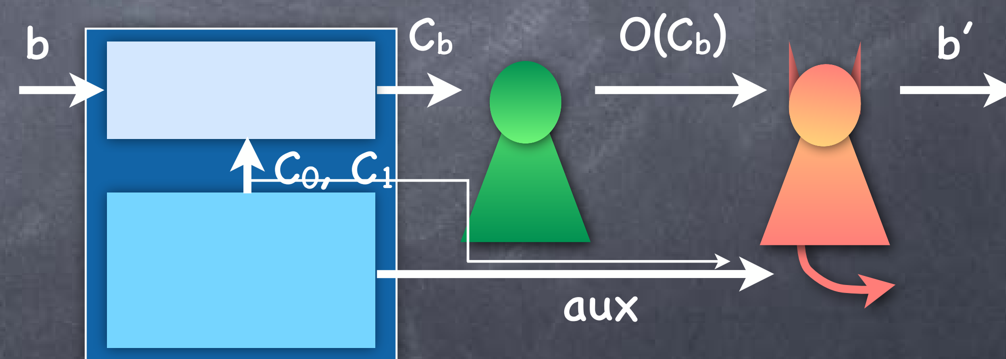
is IDEAL-Hiding if

$\forall$ PPT Pr[b′=b] = ½ ± negl.



is REAL-Hiding if

$\forall$ PPT Pr[b′=b] = ½ ± negl.



PC-DIO if $\forall$ PPT in PC-DIO Test-Family

IDEAL-hiding $\Rightarrow$ REAL-hiding

IDEAL

REAL

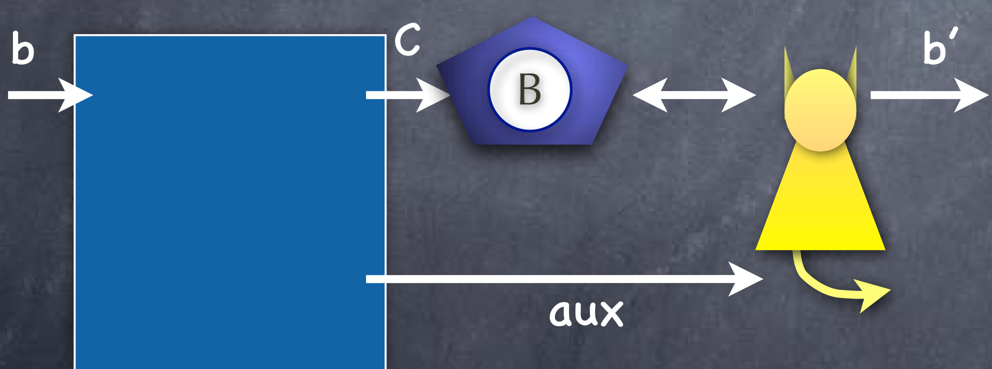
Virtual Grey Box Obf.

Arbitrary PPT Test, with arbitrary aux (C_0, C_1 not necessarily included).
Allow computationally unbounded adversaries in the ideal world.

Original definition is simulation-based a la VBB Obfuscation

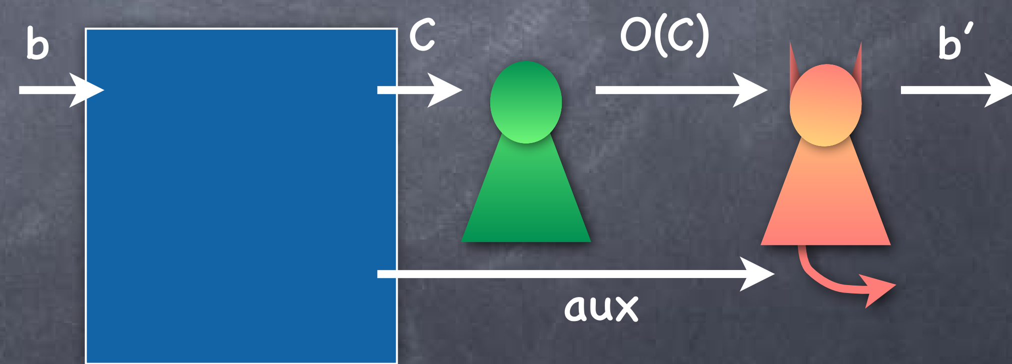
is IDEAL-Hiding if

∀ Pr[b′=b] = ½ ± negl.



is REAL-Hiding if

∀ PPT Pr[b′=b] = ½ ± negl.



VGB Obf. if ∀ PPT in <u>VGB Test-Family</u>

IDEAL-hiding statistically ⇒ REAL-hiding

IDEAL

REAL

Inefficient iO

XIO: Allows inefficient evaluation, slightly better than truth table

- Write down the truth table of the function! But not efficient.
- Better solution: Find a canonical circuit for the given circuit (e.g., smallest, lexicographically first)
- Meets every requirement except that of the obfuscator being efficient
- Fact: Can find the canonical circuit in polynomial time if P=NP
  - i.e., P=NP $\Rightarrow$ iO (with efficient obfuscator) exists
  - Cannot rule out the possibility that iO exists but there is no OWF (say), unless we prove P≠NP

iO from Compact FE

- High-level idea:
  - Obfuscation is an FE encryption of the program, Enc(P)
  - Function keys to get Enc(P||x), and then to evaluate P(x) from it

    Challenge: How?

    As U(P||x) = P(x), where U is a universal circuit
  - Incrementally: to compute Enc(α||0) and Enc(α||1) from Enc(α)
    - Just give a function key to compute $f_b(\alpha)$ = Enc(α||b) !
    - An issue: $f_0(P) \approx f_0(P')$, but not equal. Still, issuing key for f_0 should keep Enc(P) ≈ Enc(P')

      Enhance FE to work for this
    - Another issue: Enc should be a function supported by FE. (By default, Enc is more complex than supported functions.)

Use a hierarchy of (single-key) FE schemes, with level i function space containing level i-1 Enc

Need to avoid exponential blowup: Enc shouldn't be much more complex than supported functions.

Compact FE: Recent constructions from strong but plausible assumptions.

Best-Possible Obfuscation

- iO as good at hiding information as any (perfectly correct) obfuscation O
  - Anything that can be efficiently learned from (aux,iO(P)) can be efficiently learned from (aux,O(P))
- (aux,iO(O(P))) ≈ (aux,iO(P)), where O is <u>any</u> compiler that perfectly preserves functionality
  - i.e., Any information that can be efficiently learned from (aux,iO(P)) can be efficiently learned from (aux,iO(O(P)))
    - In turn, efficiently learned from (aux,O(P))
  - Note: Only holds when iO is efficient (so not applicable to the canonical encoding construction)

Is iO Any Good?

- iO does not promise to hide anything about the function (only its representation)
- Can we use iO in cryptographic constructions?
  - Yes (combined with other cryptographic primitives)
  - e.g. PKE from SKE using iO
  - In fact, can get FE (from PKE and NIZK) using iO
    - Recent results: iO "essentially" equivalent to FE for general functions (note: FE doesn't hide function)

With different levels of security

Implausibility of DIO?

- Is DIO (im)possible?
- Open
- Constructions from multi-linear maps under strong (or idealized) assumptions
- Implausibility results
  - If highly secure ("sub-exponentially secure") one-way functions exist, then highly secure DIO for Turing machines cannot exist!
- Problem is the auxiliary information
  - Let aux be an obfuscated program which can extract secrets from the obfuscated program. But in the ideal world, aux would be useless (as it is obfuscated).

Today

- Obfuscation
- Strong definitions are provably impossible to achieve
  - Several weaker definitions
- Recent breakthroughs for iO