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 reveals

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

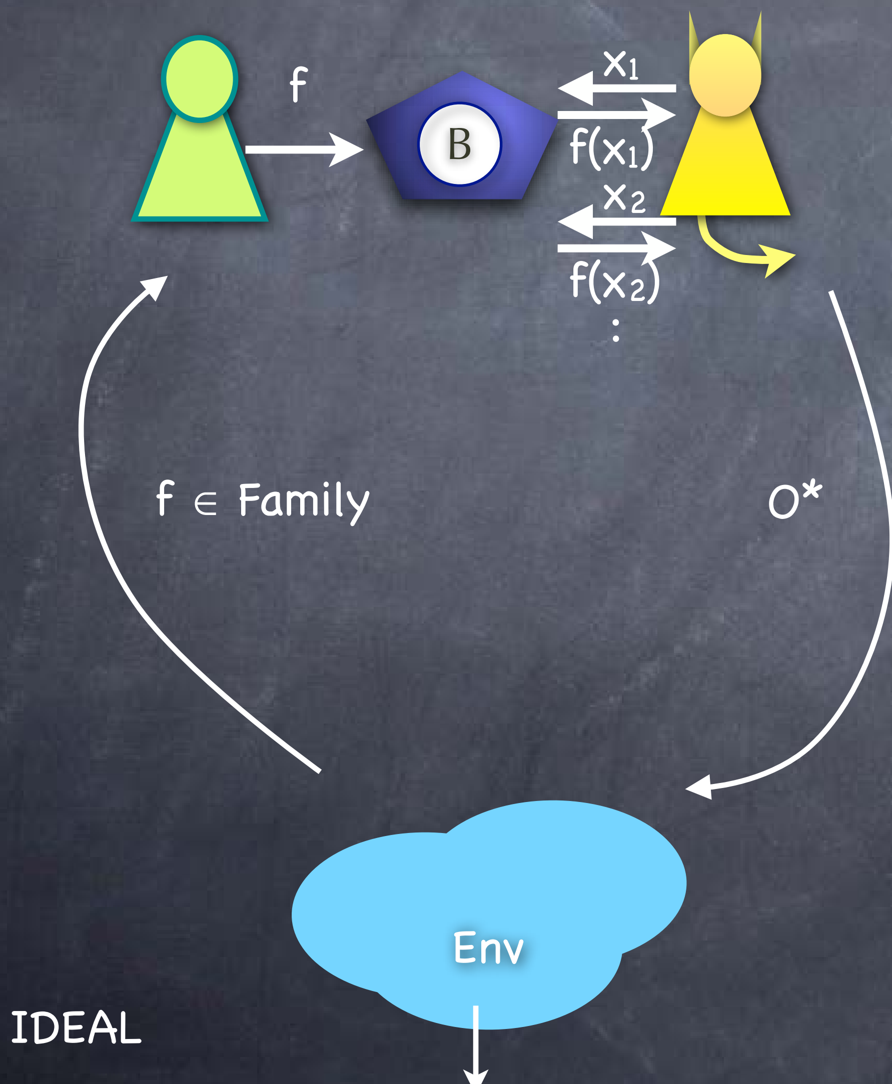
Cryptographic Obfuscation

- 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:

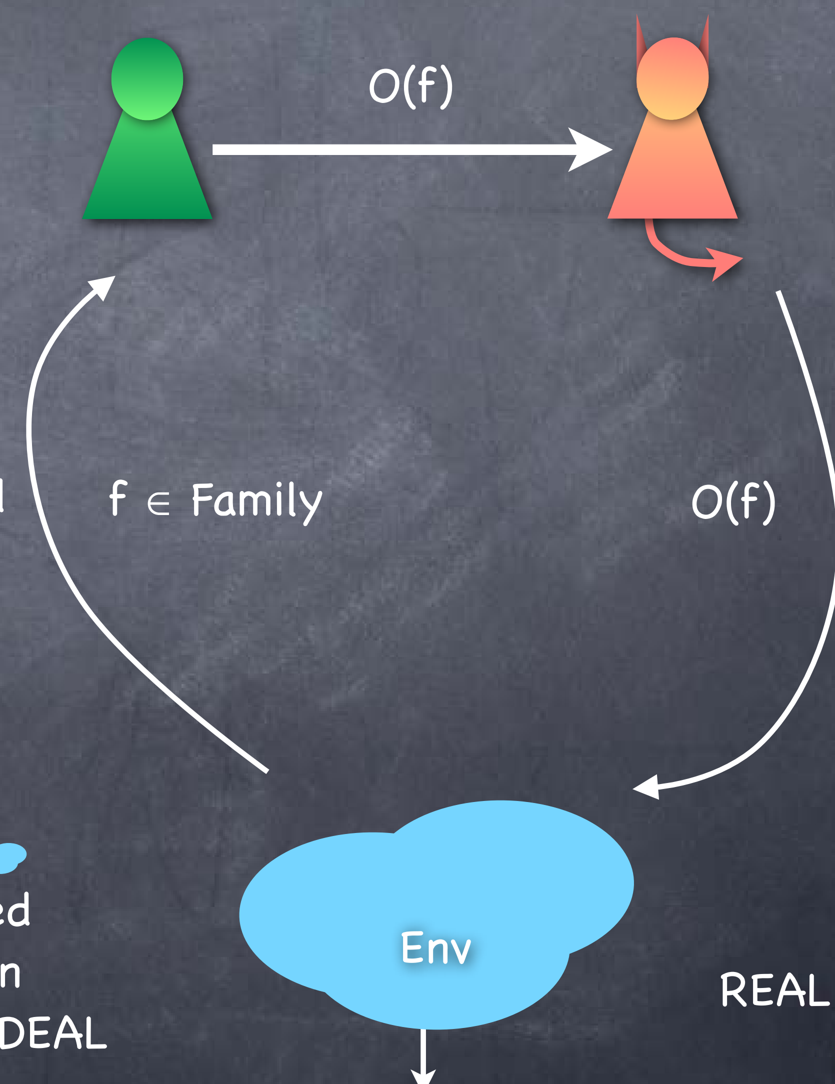
$\forall$

$\exists$ s.t.

$\forall$

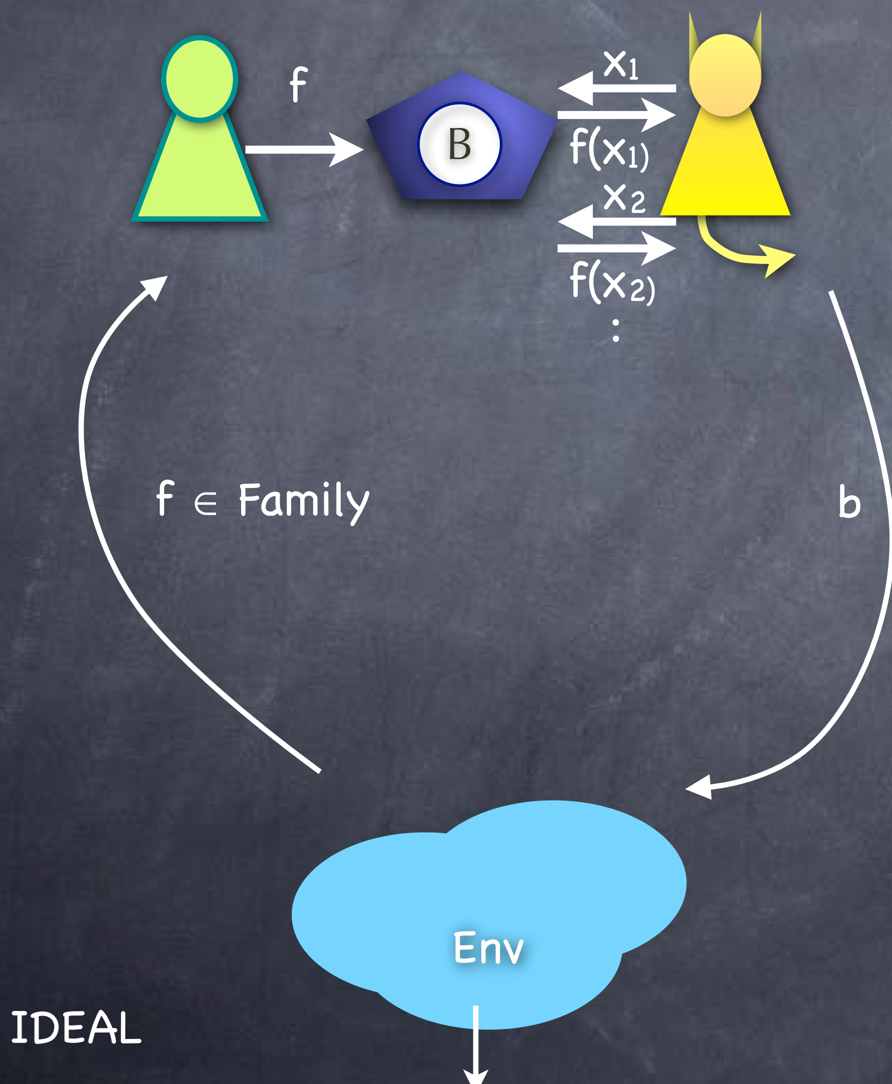
output of

is distributed identically in REAL and IDEAL



Defining Obfuscation: First Try

Note: Considers only corrupt receiver



Virtual Black-Box (VBB) Obfuscation

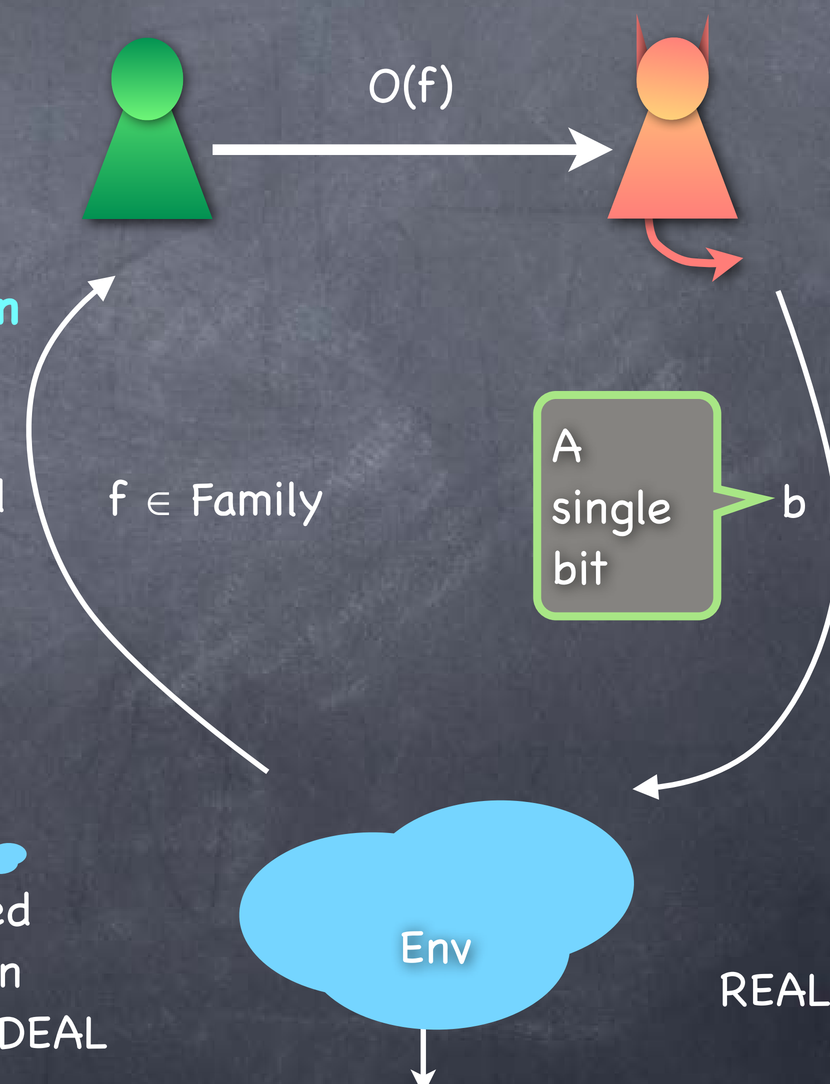
Secure (and correct) if:

$\forall$

$\exists$ s.t.

$\forall$

output of is distributed identically in REAL and IDEAL



Impossibility of Obfuscation

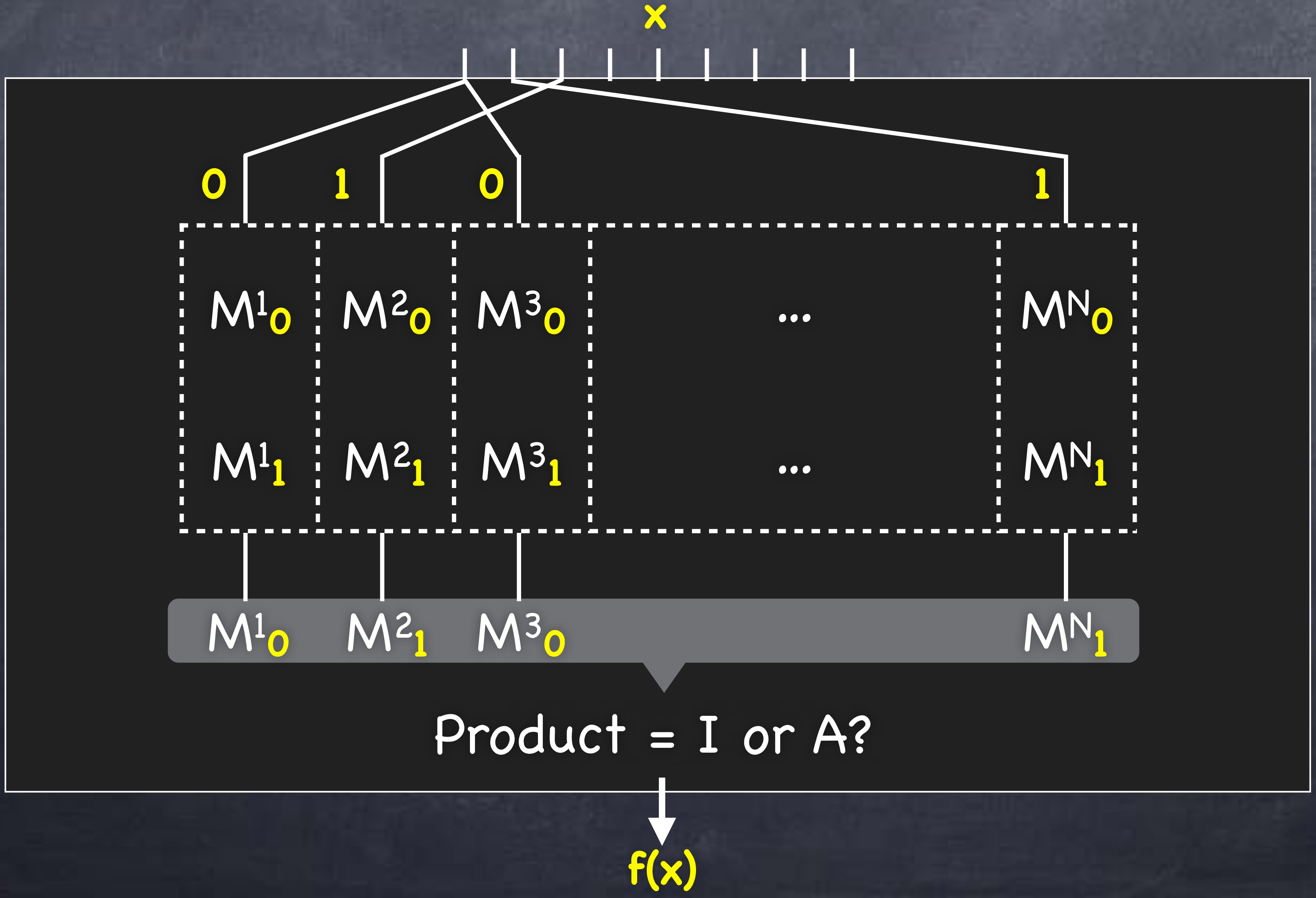
- VBB obfuscation is impossible in general
- Explicit example of an unobfuscatable function family
  - Idea: program which when fed its own code (even obfuscated) as input, outputs secrets
  - Programs $P_{\alpha,\beta}$ with secret strings α and β:
    - If input is of the form $(0,\alpha)$ output β
    - 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 (random oracle model, generic group model, etc.)
- For weaker definitions
- Obfuscation constructions need a suitable representation of the function

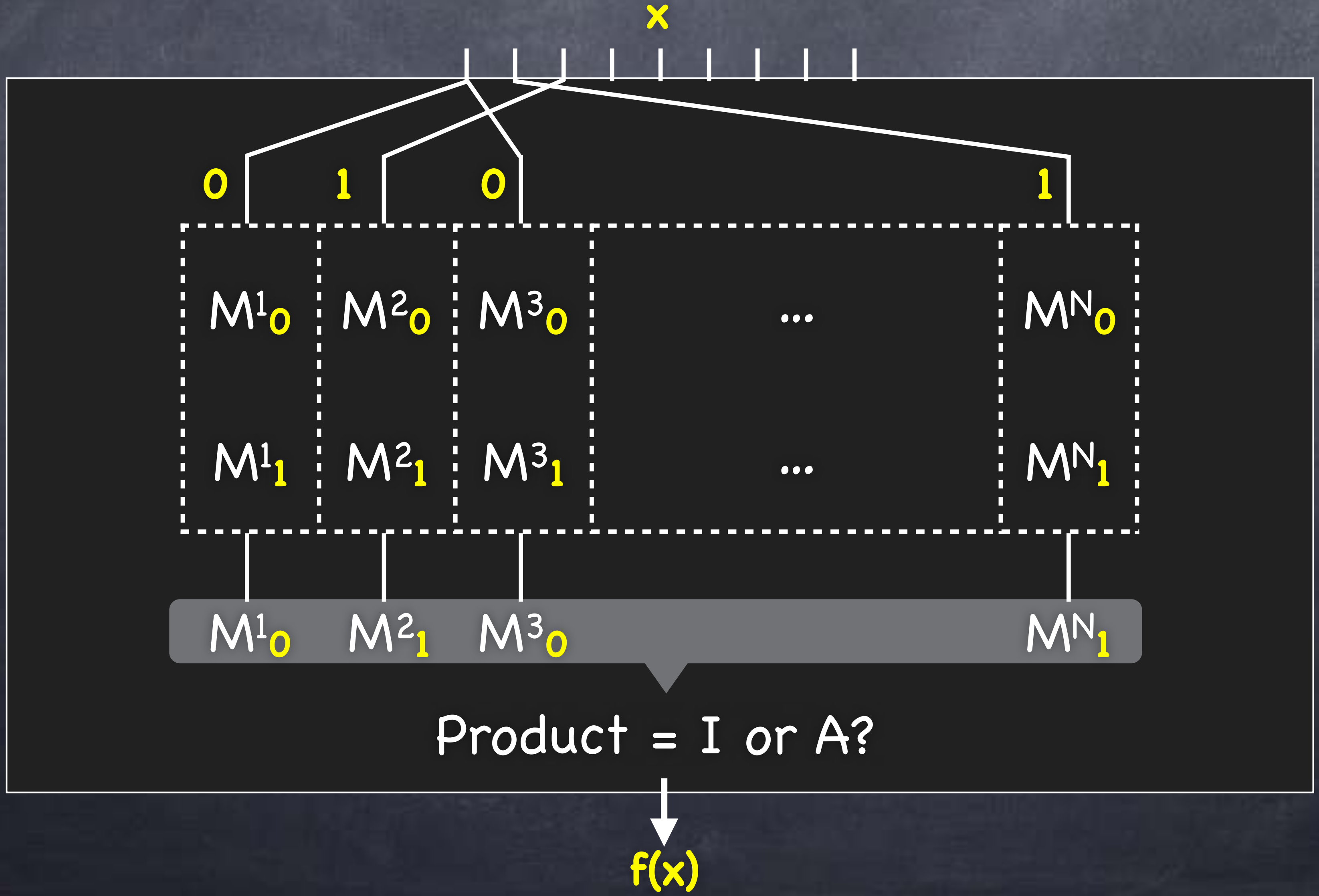
Matrix Programs

- $f : \{0,1\}^n \rightarrow \{0,1\}$ 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 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)
- No other information about the 2N matrices should be deducible



Obfuscation from Multi-Linear Map

- Such encodings are known using "multi-linear maps"
  - Using <u>**generic model**</u> 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 Fully Homomorphic Encryption with decryption in NC^1
  - Instantiating obfuscation constructions using concrete hardness assumptions on these candidates yields weaker flavours of obfuscation (coming up)
- 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.

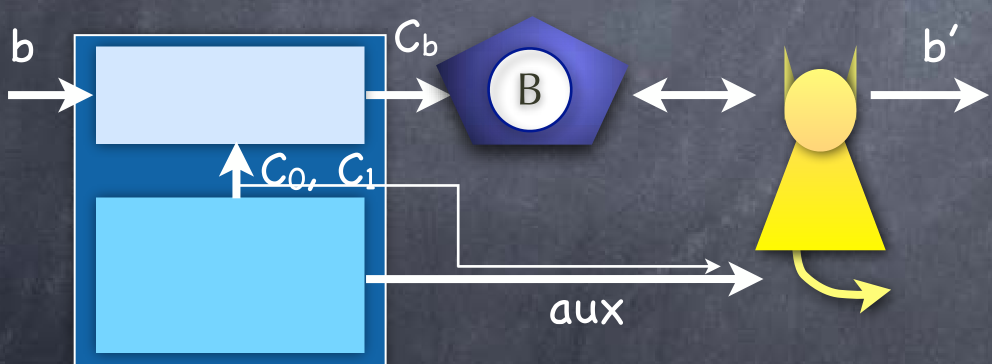
IND-PRE Security

Different variants of the definition in this framework

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

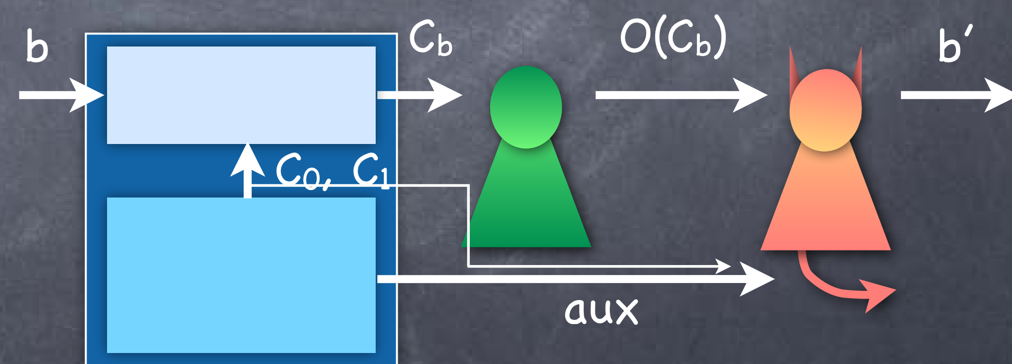
is IDEAL-Hiding if

$\forall$ PPT Pr[b′=b] = ½ ± negl.



is REAL-Hiding if

$\forall$ PPT Pr[b′=b] = ½ ± negl.



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

IDEAL-hiding $\Rightarrow$ REAL-hiding

IDEAL

REAL

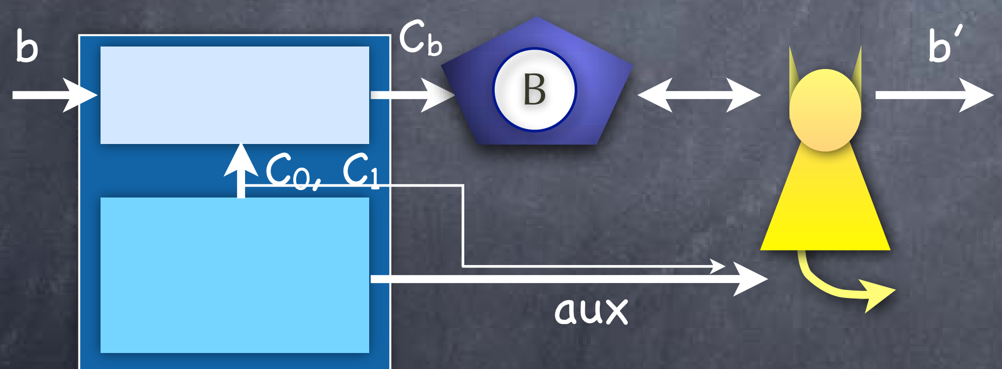
Indistinguishability Obf. (iO)

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

Guaranteed to be IDEAL-hiding

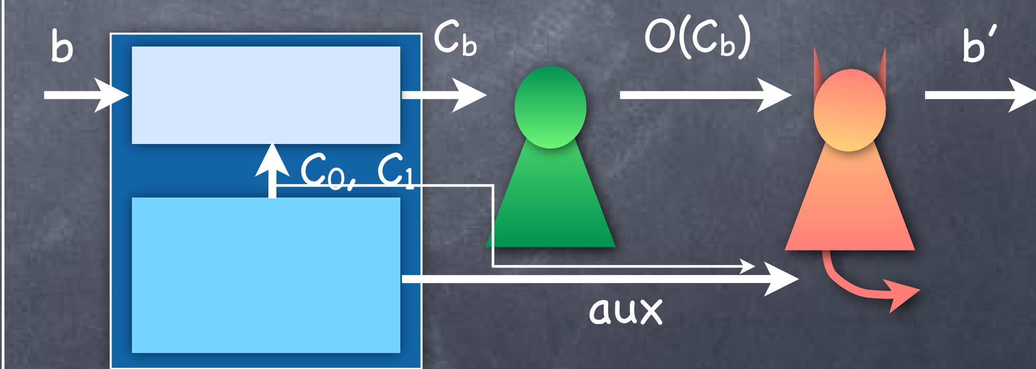
is IDEAL-Hiding if

∀ PPT Pr[b′=b] = ½ ± negl.



is REAL-Hiding if

∀ PPT Pr[b′=b] = ½ ± negl.



iO if ∀ PPT in iO Test-Family

IDEAL-hiding ⇒ REAL-hiding

IDEAL

REAL

Differing Input Obf.

C_0, C_1 need not be functionally equivalent

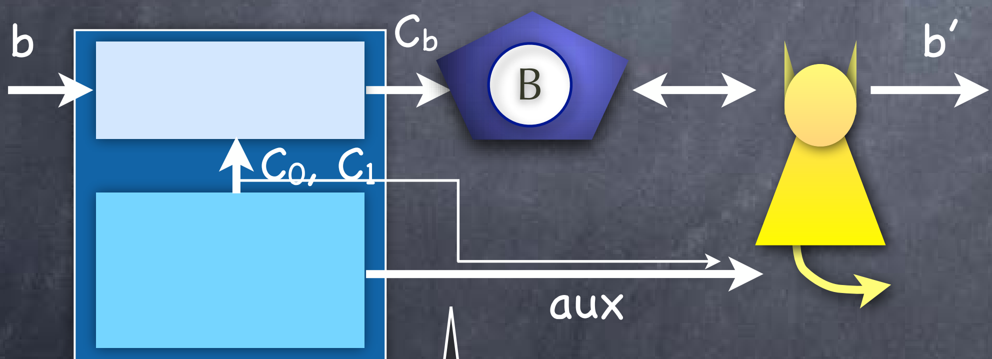
To be not IDEAL-hiding, need a PPT which can find a "differing input"

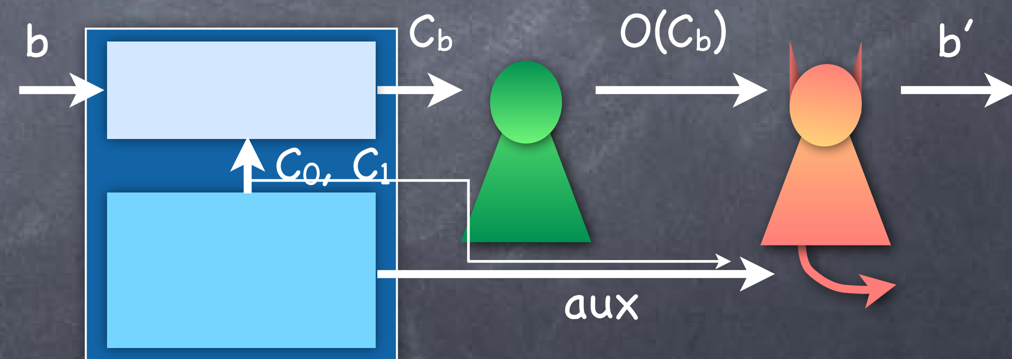
is IDEAL-Hiding if

$\forall$ PPT Pr[b′=b] = ½ ± negl.

is REAL-Hiding if

$\forall$ PPT Pr[b′=b] = ½ ± negl.





Adaptive DIO allows 2-way interaction

IDEAL

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

IDEAL-hiding $\Rightarrow$ REAL-hiding

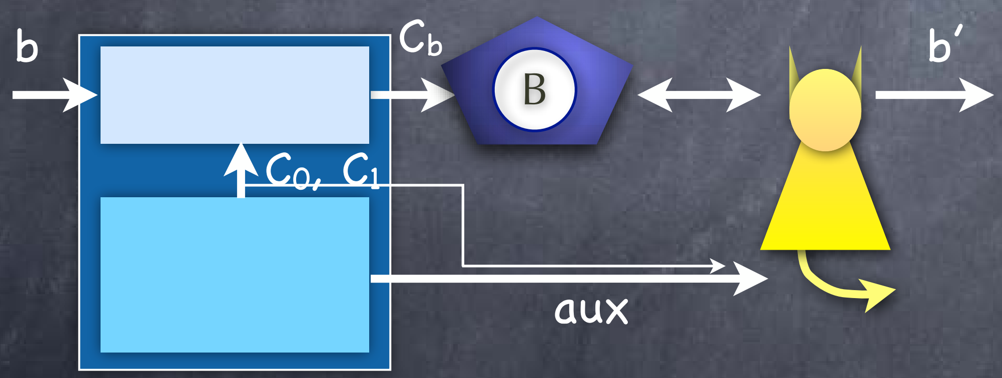
REAL

Public-Coin DIO

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

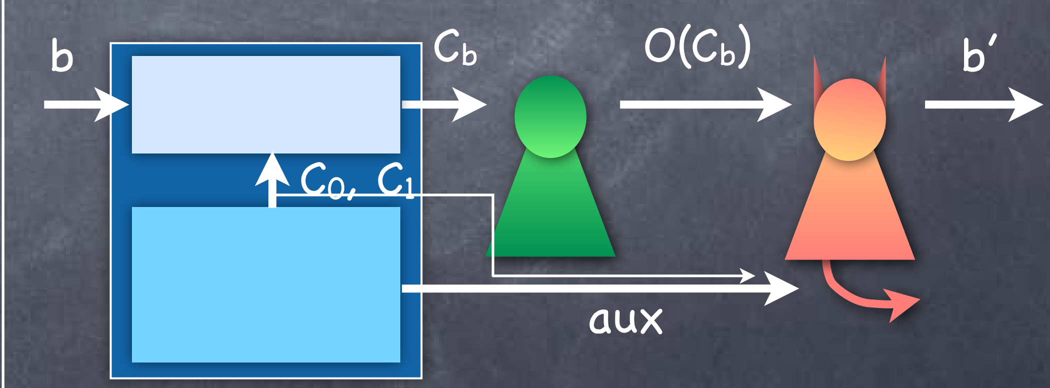
is IDEAL-Hiding if

∀ PPT Pr[b′=b] = ½ ± negl.



is REAL-Hiding if

∀ PPT Pr[b′=b] = ½ ± negl.



PC-DIO if ∀ PPT in <u>PC-DIO Test-Family</u>

IDEAL-hiding ⇒ REAL-hiding

IDEAL

REAL

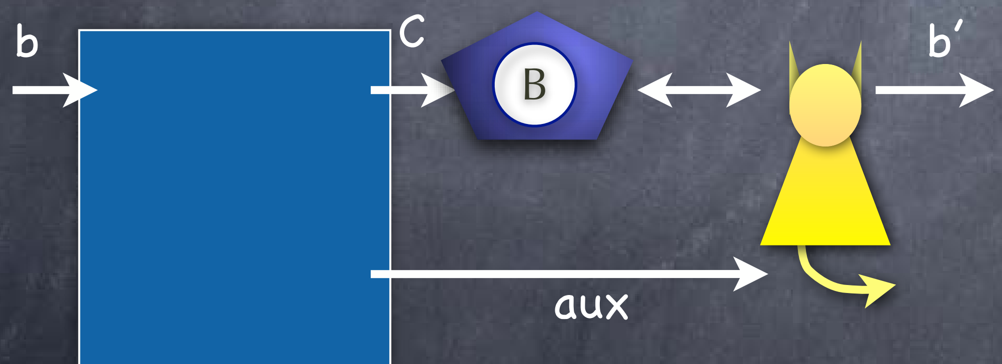
Virtual Grey Box Obf.

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

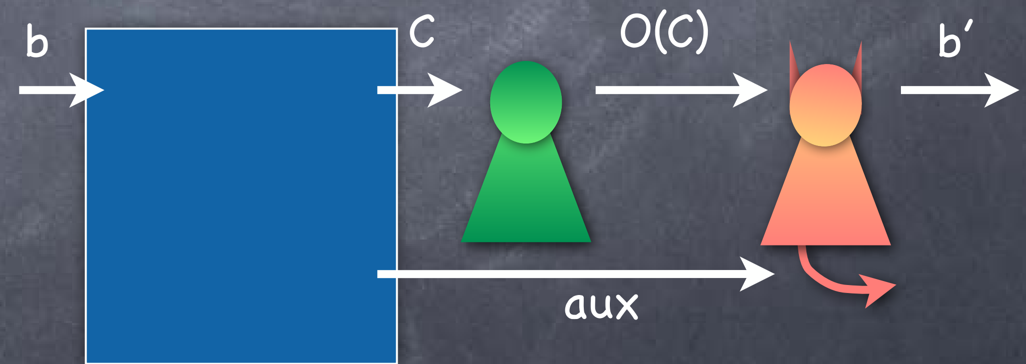
∀ Pr[b′=b] = ½ ± negl.



IDEAL

is REAL-Hiding if

∀ PPT Pr[b′=b] = ½ ± negl.



REAL

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

IDEAL-hiding statistically ⇒ REAL-hiding

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

Best-Possible Obfuscation

- iO as good at hiding information as any (perfectly correct) obfuscation
- (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
- Recent breakthroughs (for weaker definitions)
    - Using Multi-linear Maps
    - Still being cryptanalyzed