Analyzing Functional Entropy of Software Intent Protection Schemes

J. Todd McDonald
Eric Trias
Alan Lin

Center for Cyberspace Research
Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright Patterson AFB, OH

Why Do We Try to Protect Software?

- Because Programs are Attacked….
- Protect Integrity
  - Decomposing/reusing code
  - Adding new functionalities
- Protect Intent
  - Alter existing functionality
  - Prevent "gaming" functionality
  - Prevent countermeasures
- Protect Ownership/Intellectual Property
- Protect Troops/Mission

We are concerned primarily with software-only means of protection

Underlying Goals

- Given the hardware/physical environment:

  make it hard for an adversary to reliably or predictably recover an intermediate or original form (Netlist, source level program code)

- Given recovery of some or all of the intermediate / original description of a circuit or program:

  make it hard for an adversary to recover, predict, subvert, or copy functionality

Ideal Software Protection…

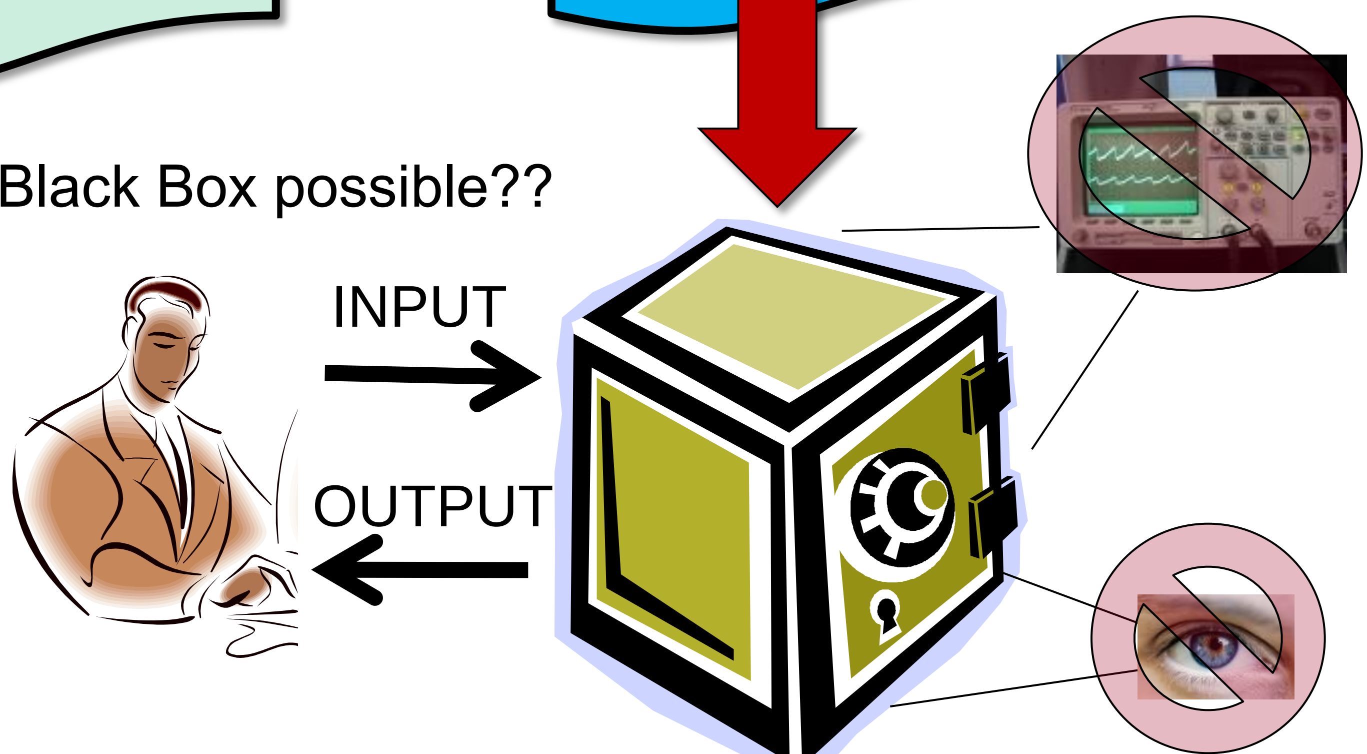
Program Source Code (Java/C++/C)

Circuit Netlist (VHDL/Verilog/BENCH)

Assembly

Realized Circuit/FPGA

Is a "Virtual" Black Box possible??



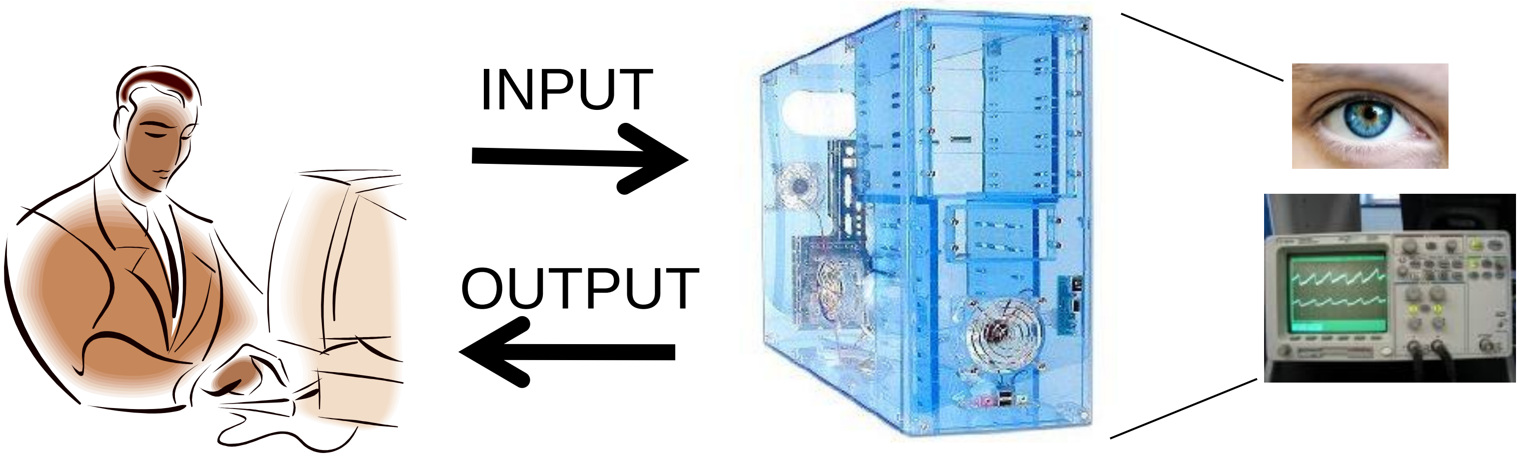
Real-world Software Protection…

Program Source Code (Java/C++/C)

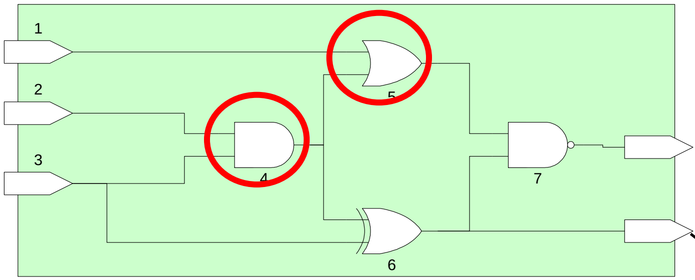
Circuit Netlist (VHDL/Verilog/BENCH)

Assembly

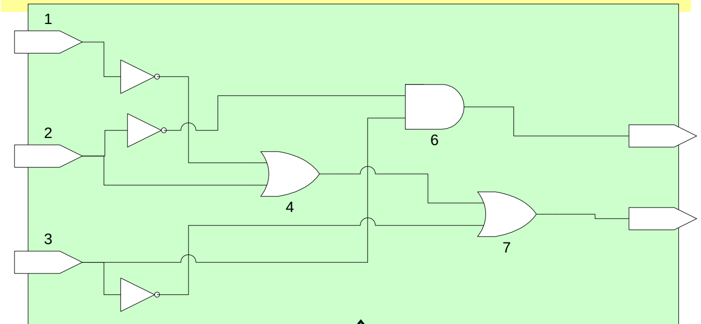
Realized Circuit/FPGA



General Intuition and Hardness of Obfuscation



The ONLY true "Virtual Black Box"



X1	X2	X3	4	5	Y6	Y7
			AND(3,2)	OR(4,1)	XOR(4,3)	NAND(5,6)
0	0	0	0	0	0	1
0	0	1	0	0	1	1
0	1	0	0	0	0	1
0	1	1	1	1	0	1
1	0	0	0	1	0	1
1	0	1	0	1	1	0
1	1	0	0	1	0	1
1	1	1	1	1	0	1

"The How"

X1	X2	X3	Y6	Y7
			XOR(4,3)	NAND(5,6)
0	0	0	0	1
0	0	1	1	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	0	1

Semantic Behavior

Program Understanding

- Adversary's ability to **anticipate a program's operational manifestation(s)**
- Adversary's ability to gain **intent indications** by **comparing** the obfuscated code, or segments, to **known code libraries**
- Adversary's knowledge gained relative to the theoretical **Virtual Black Box**
- Adversary's ability to extract the **information content** as manifested in the black box and white box aspects of program code

This is not the same as VBB or hiding all information…

Program Understanding

- **Our Context:** Prevent program understanding by limiting the amount of information gained by an adversary from either the blackbox or whitebox characteristics of a program/circuit
  - Programs are no more than **a special information class** with well-defined syntax and semantics
  - **Scrambling techniques are limited** because final form of program must adhere to rigid syntax and semantics
  - Program code **information content** is otherwise equivalent to information content in any other type of bit stream

- **Our Premise:** Program code that is statistically indistinguishable from a random bit stream has negligible information content

Defining Intent Protection

Is there an alternate (or better) way to measure security or protection?

Adversarial Observation:

Black Box Analysis

White Box Analysis

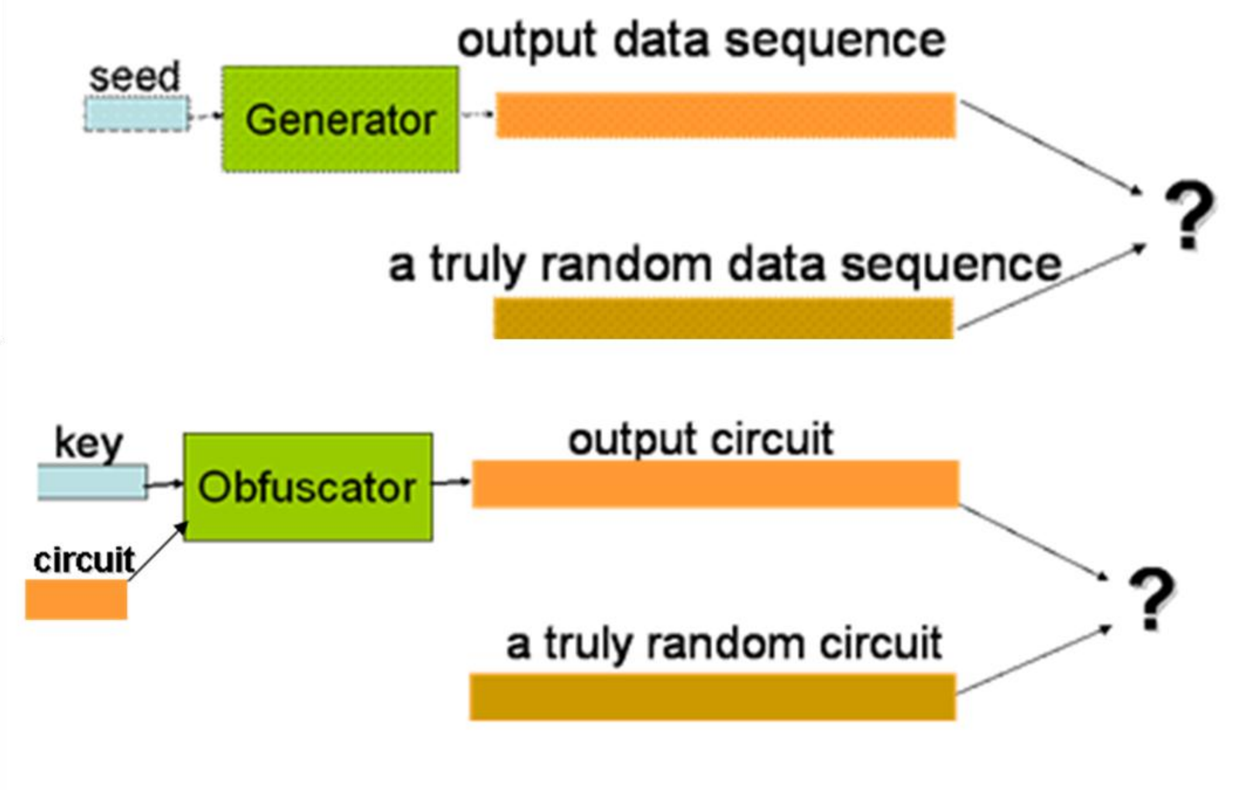
If the adversary cannot determine the function/intent of the device by input/ output analysis, we say it is black-box protected

If the adversary cannot determine the function/intent of the device by analyzing the structure of the code, we say it is white-box protected

Intent Protected: Combined black-box and white-box protection does not reveal the function/intent of the program

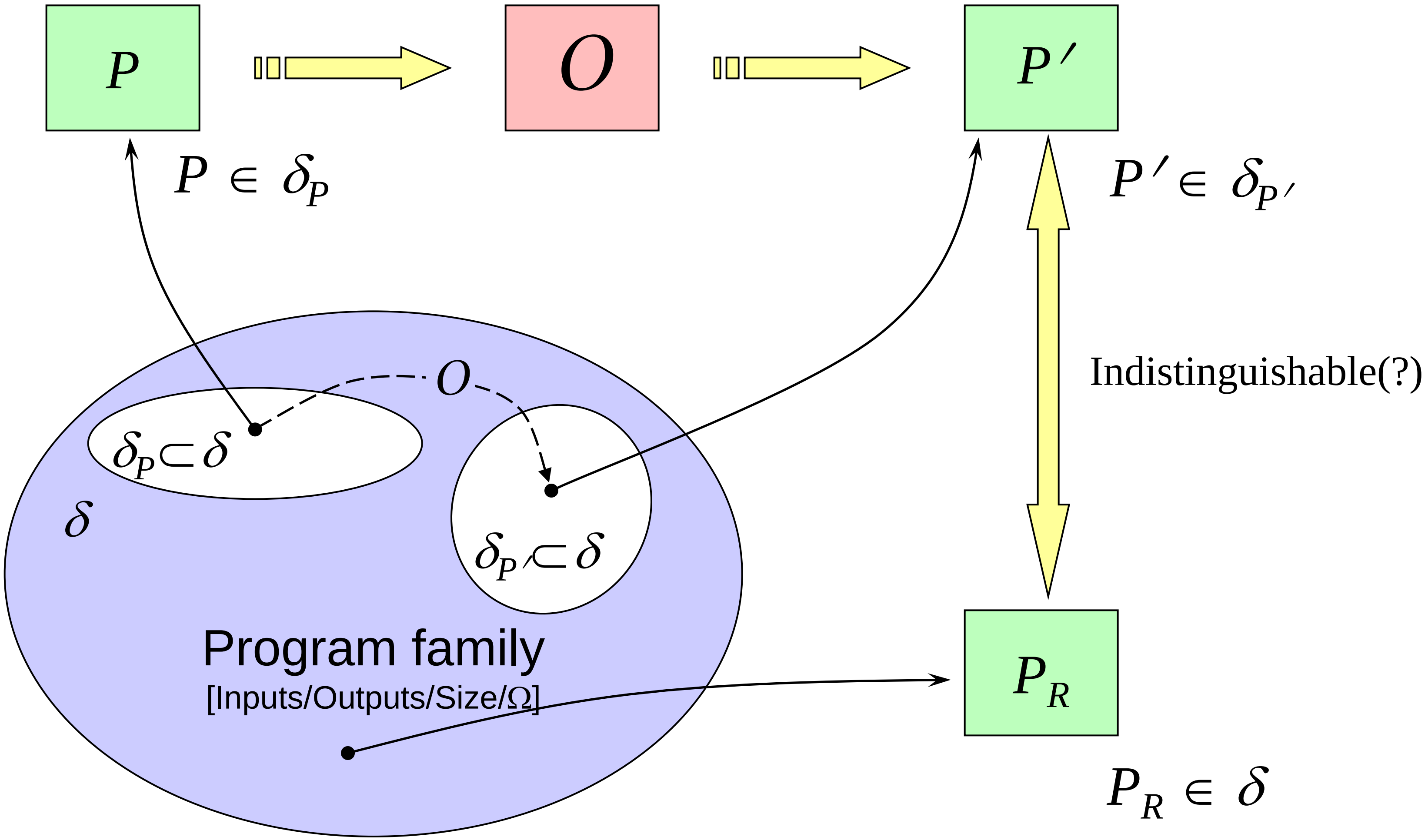
Random Programs/Circuits

Goals: Can we make **input/output** look random?
Can we make **structure** look random?



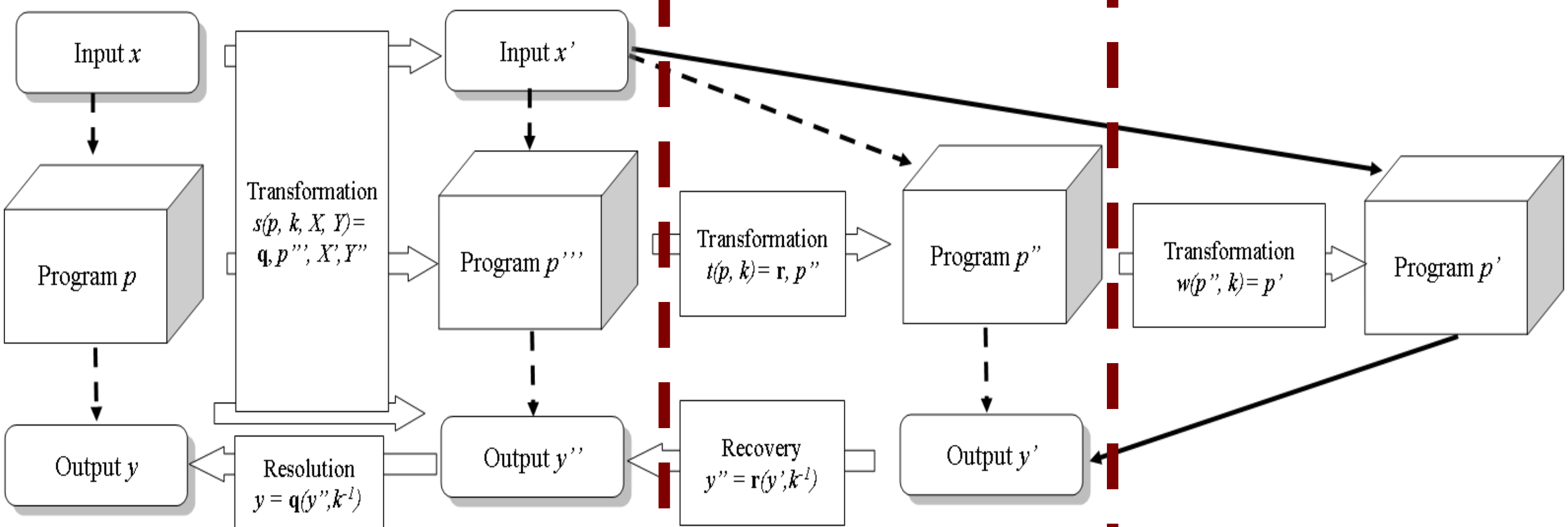
Instead of measuring security based on *leakage of information from the obfuscated program*, can we appeal to entropy or randomness as a measure for confusion in the obfuscated program?

RPM: Random Program Model



Combined Experimental Framework

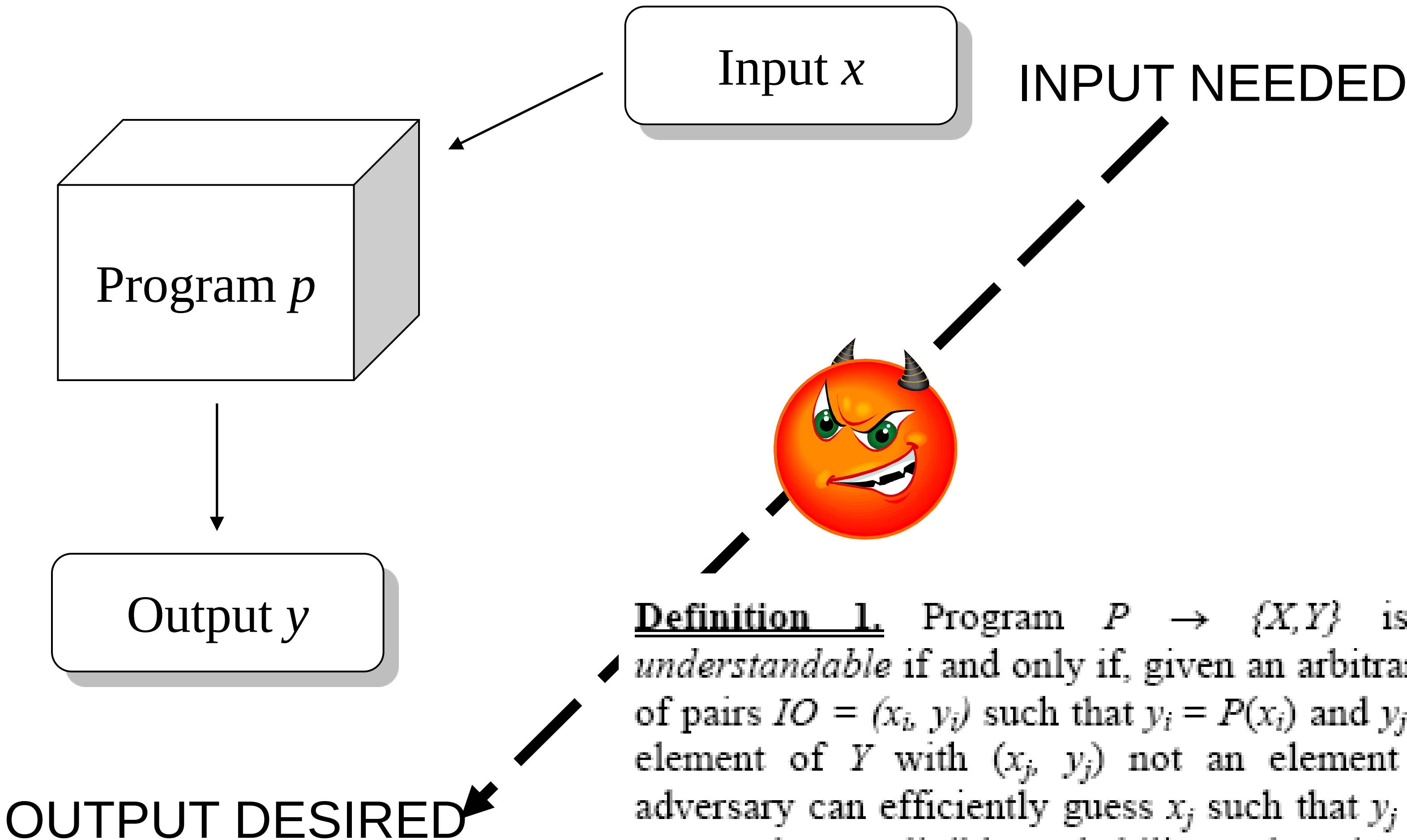
Black-box Refinement | **Semantic Transformation** | **White-box Randomization**



Black-Box I/O | **Black-Box I/O** | **White-Box Structure**

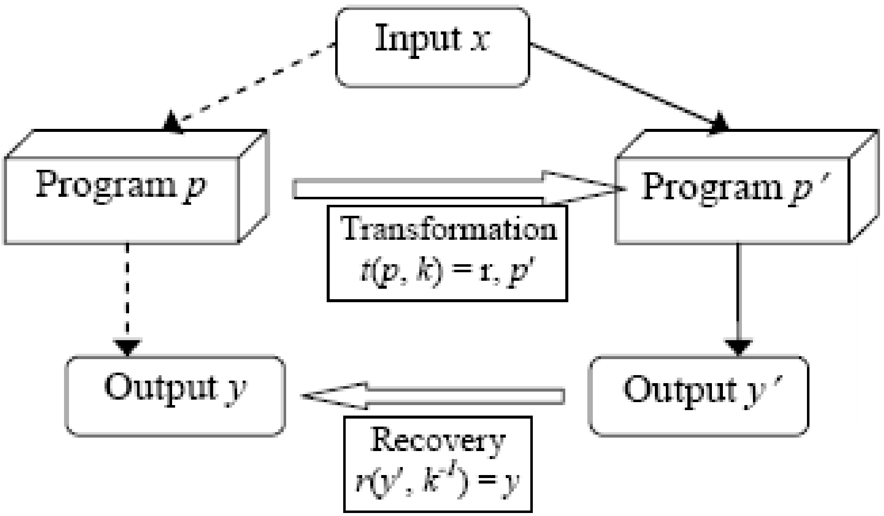
Goals: Can we make **input/output** look random?
Can we make **structure** look random?

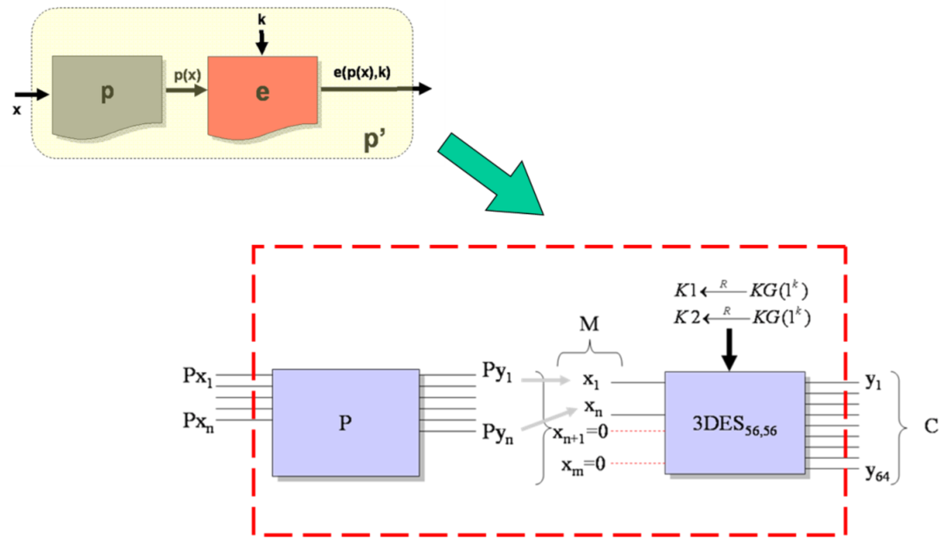
Protecting Black-Box Intent



Definition 1. Program $P \rightarrow \{X,Y\}$ is *black-box understandable* if and only if, given an arbitrarily large set of pairs $IO = (x_i, y_i)$ such that $y_i = P(x_i)$ and y_j an arbitrary element of Y with (x_j, y_j) not an element of IO), an adversary can efficiently guess x_j such that $y_j = P(x_j)$ with greater than negligible probability. Otherwise, we say P is *black-box obfuscated*.

Semantic Transformation

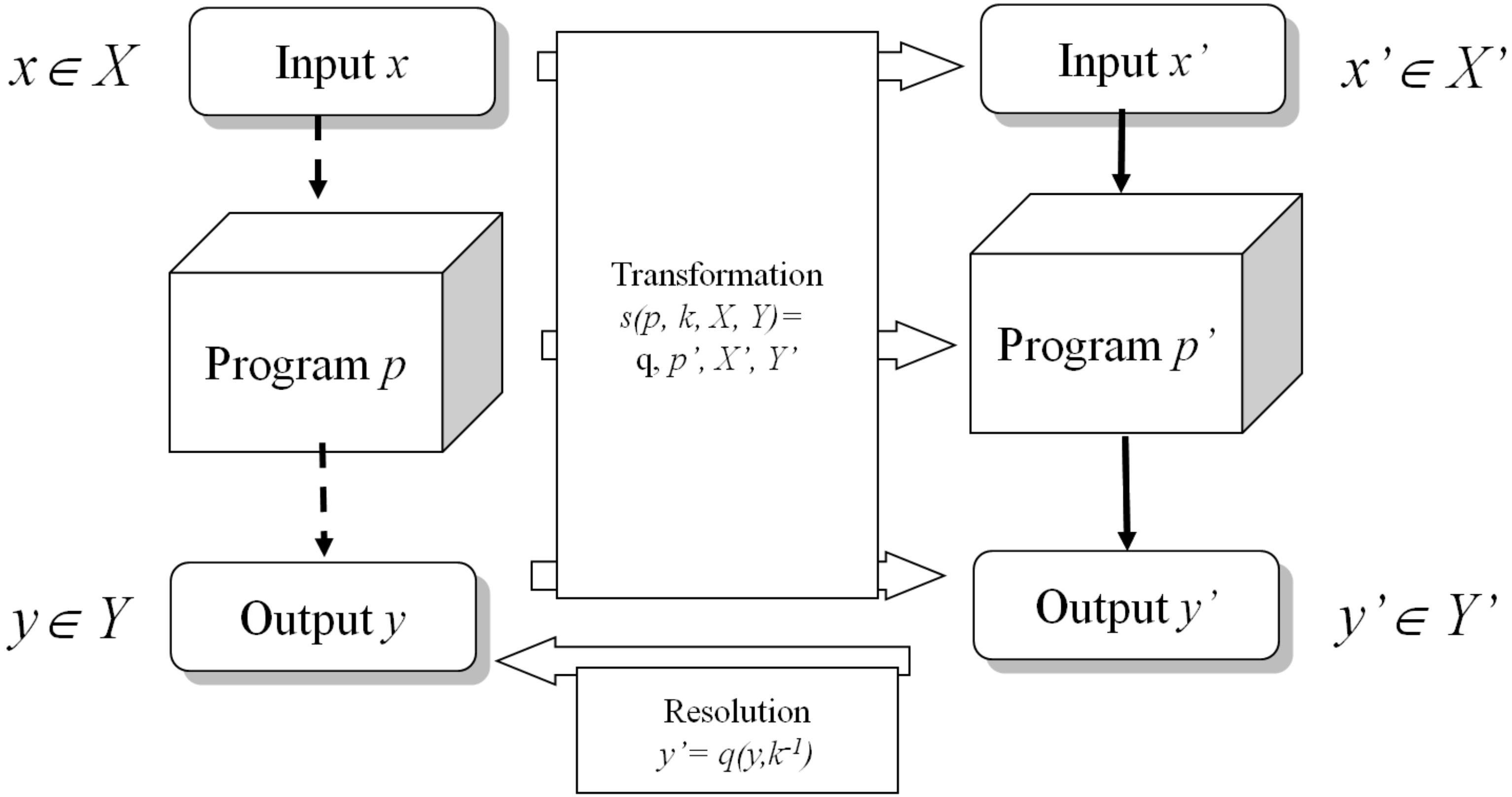




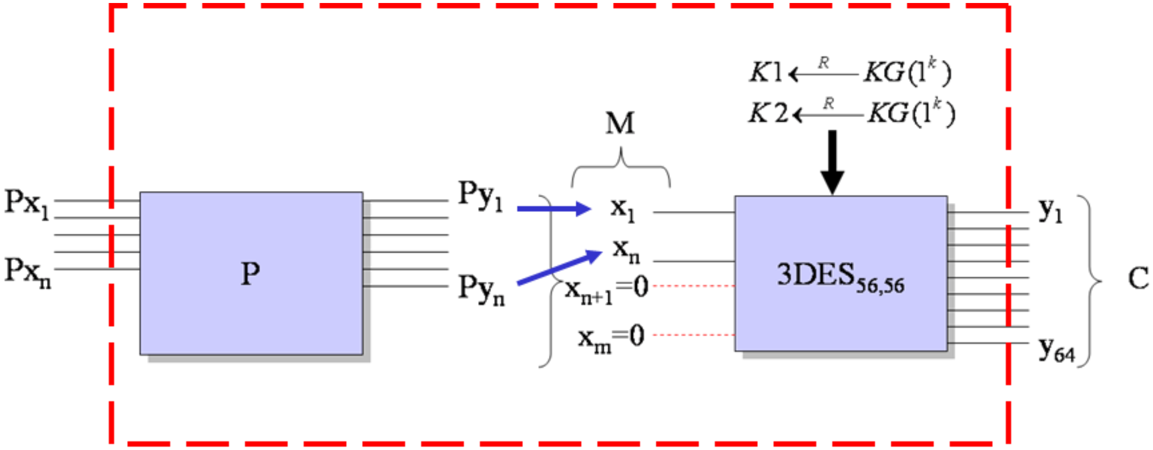
Semantically secure data encryption algorithms are black-box intent protected (BBIP)

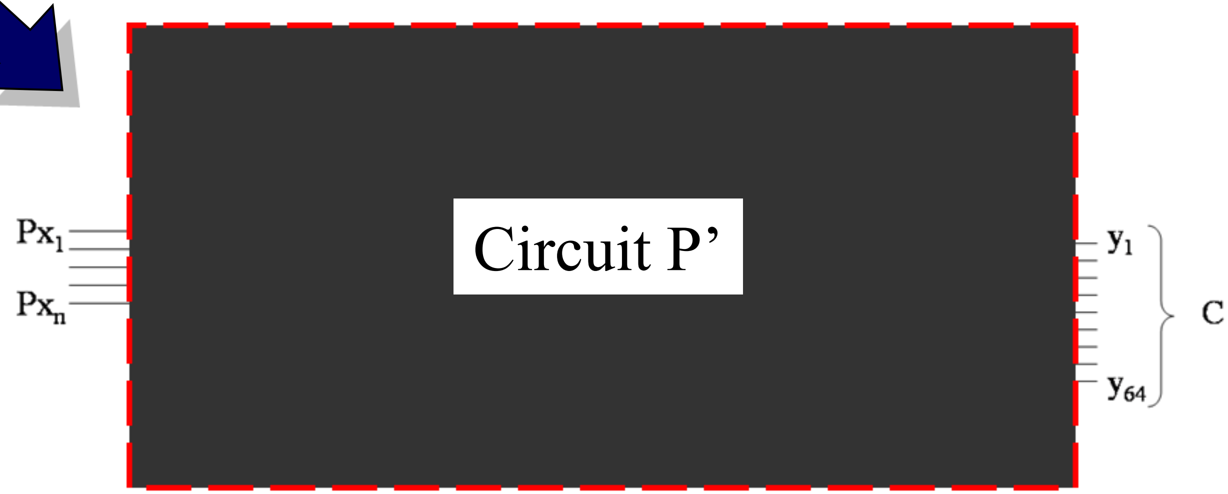
Compositions of programs with semantically strong algorithms are likewise BBIP

Black Box Refinement



Protecting White-Box Intent



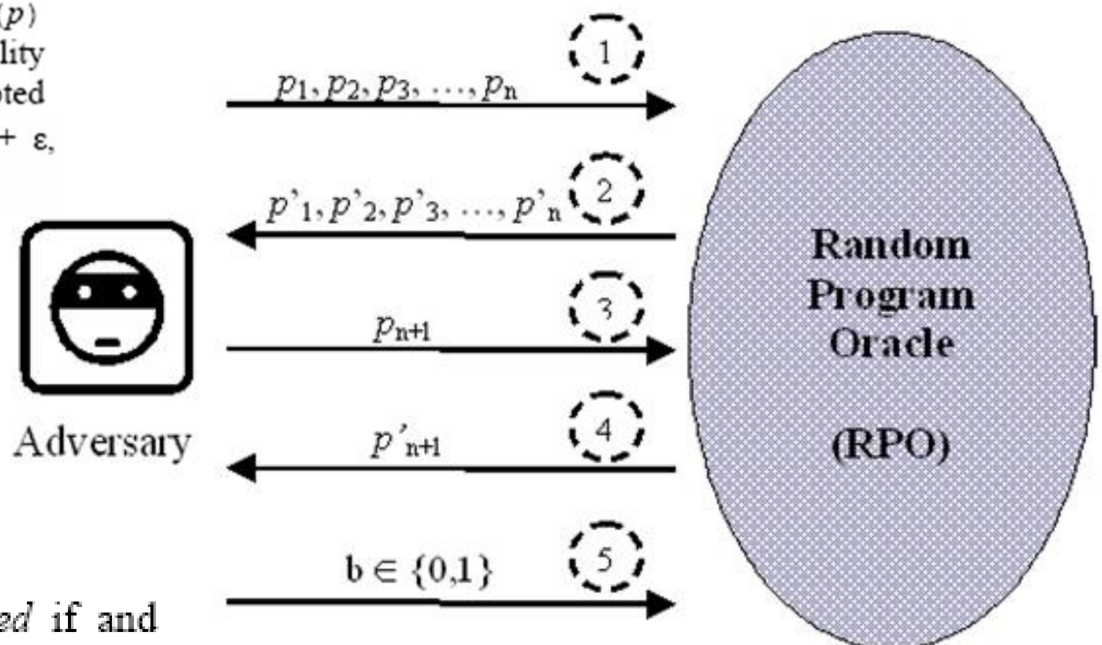


Ideal for AT applications where we shield hardware internals with some (reasonably) trusted AT method

Protecting White-Box Intent

Definition 2. Given access to a random program oracle which transforms any program p using algorithm $E(p)$ into an encrypted version p', and given full access to any encrypted program p'_x. After knowing any n pairs of original and encrypted programs $\{(p_1, p'_1), (p_2, p'_2), \ldots, (p_{n-1}, p'_{n-1}), (p_n, p'_n)\}$, an adversary that supplies a subsequent program p_{n+1} will receive p_{n+1}' from the oracle which is either: a random program (P_R) or the encrypted version of the program $p_{n+1}' = E(p_{n+1})$. The program $E(p)$ provides white-box protection if and only if the probability that an adversary is able to distinguish the encrypted program (p_{n+1}') from a random program (P_R) is ½ + ε, where ε is a negligible constant.

$$p_{n+1}' = \begin{cases} P_R & \Pr[p_{n+1}' = P_R] \leq \frac{1}{2} + \varepsilon \\ E(p_{n+1}) & \Pr[p_{n+1}' = E(p_{n+1})] \leq \frac{1}{2} + \varepsilon \end{cases}$$



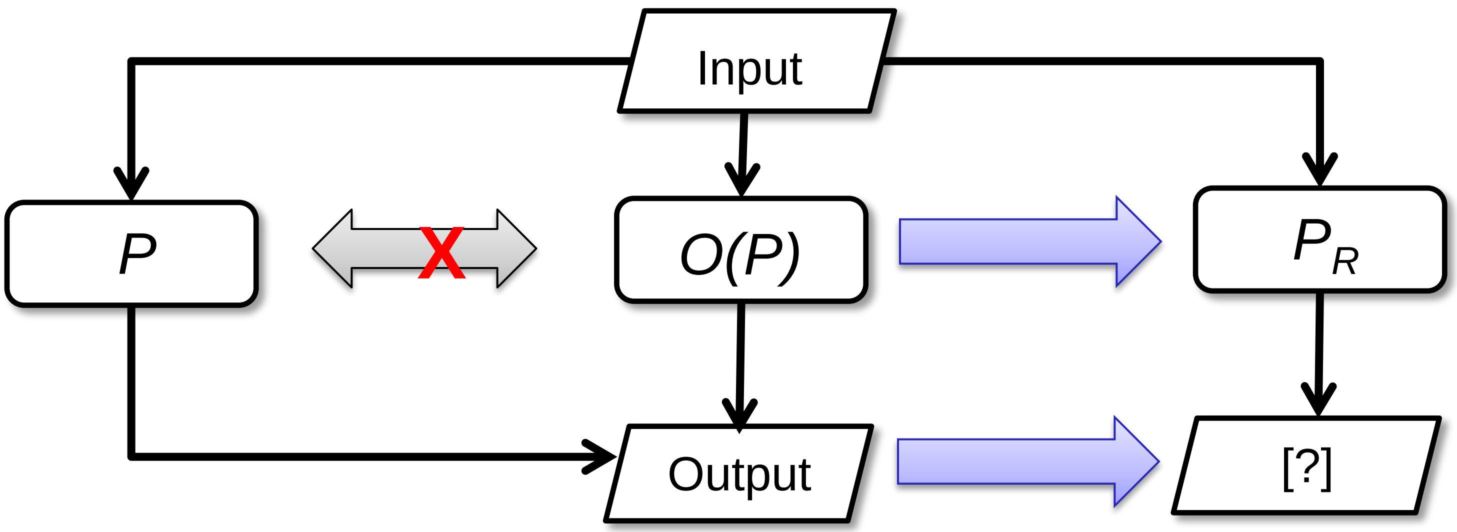
Definition 3. Program P is *intent protected* if and only if it is protected against black-box analysis and white box analysis.

Characterizing Intent Protection

- Conjecture when using Semantic Transformation:
  - If the output bits are predictable, then the output may be predictable
  - Treat each output position as a bit string generator
    - Run statistical randomness tests on each bit
- Questions of Interest to the Random Program Model:
  - **Does *structural* randomness produce *functional* randomness?**
    - Frequency of signature collisions (identical output patterns)
    - Approximate entropy of output bits
  - How random are the output bits?
    - Randomness values for specific statistical test

Methodology

- Unable to evaluate structure with agreed security metrics[1]
- Random Oracle used in absence of a defined security model
  - Sanity check for implementation
- Goal
  - O(P) structurally and functionally looks like P_R

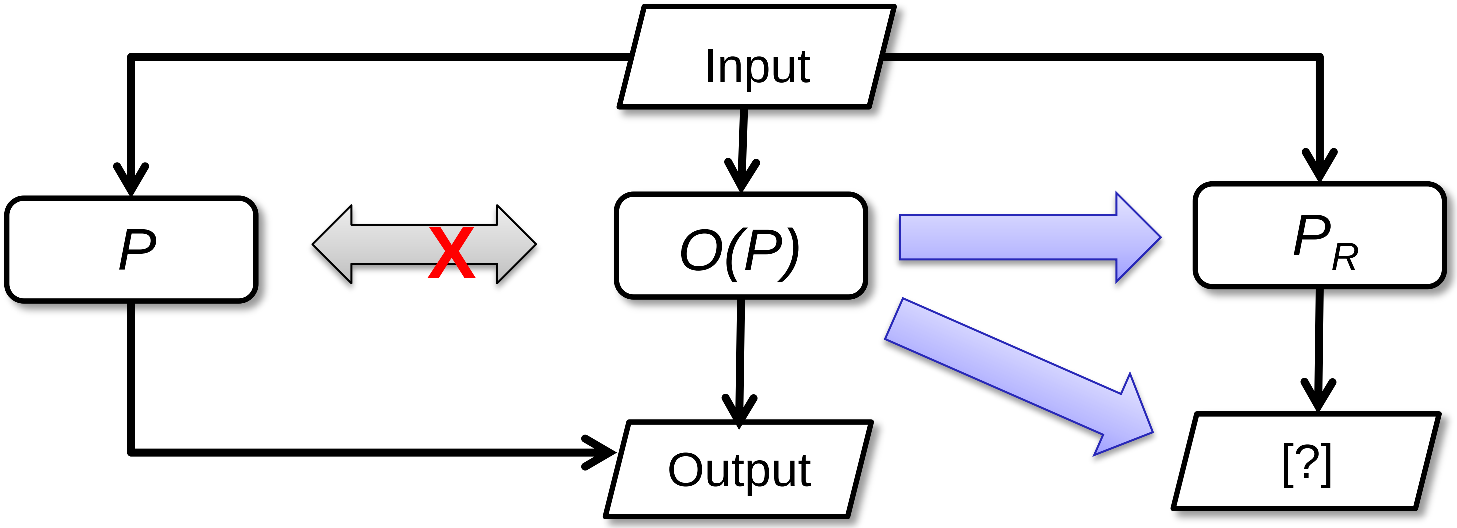


[1]National Institute of Standards and Technology

Experimental Design

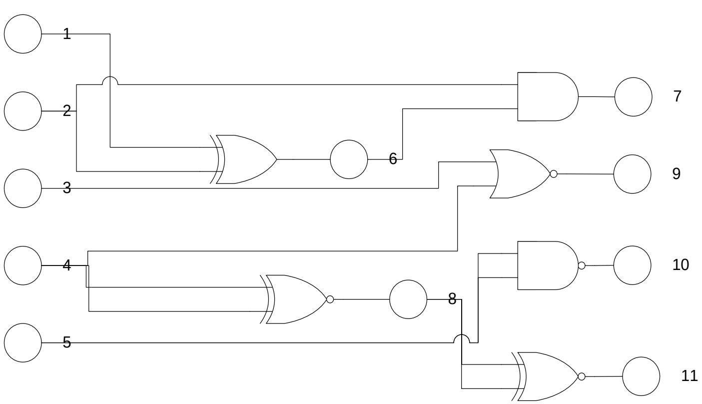
- Required components
  1. Design control /benchmark programs (deterministic)
  2. Generate P_R
  3. Black-box protect P

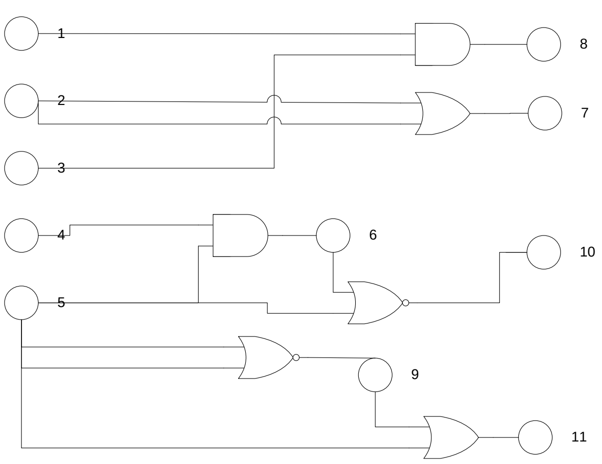
- Analysis
  - Compare P, black-box protected P, and P_R



Experimental Design

- Emulate deterministic functions w/combinational circuits
  - Abstracts high-level structure (ISCAS-85)
  - Build random circuits to analyze random circuit properties
  - Parameters (from benchmarks)
    - Input size (in bits)
    - Output size (in bits)
    - Number of intermediate nodes (represents structure)
    - Gate basis: AND, OR, XOR, NAND, NOR, NXOR





Experimental Design

- Goal
  - *O(P)* produces random functionality
- Add black-box protection to *P*
  - Weaken VBB function preservation
  - Strengthen overall security
    - Accounts for clean-room reverse engineering[1]
- Black-box w/ symmetric key cryptography
  - Produces blocks of pseudo-random bits
  - Pseudo-randomness measures exist[2]

Statistical tests
Frequency
Frequency Within a Block
Longest Runs of 1's in a Block
Runs of 0's and 1's
Cumulative Sum
Random Excursions
Random Excursions Variant
Approximate Entropy

[1]Schwartz "Reverse Engineering"

[2]National Institute of Standards and Technology

Experimental Design

- Configurations
  - Trusted-host output recovery
    - Secure execution on malicious host
    - Full structure; no functionality
  - Output recovery for malicious user
    - Partial execution on malicious host
    - Partial structure; full functionality
    - Loss of generality in function type (y = a * b + c)
  - Full ownership by malicious user/host
    - Secure structural *components*
    - Full structure; full functionality
    - Software Watermark
- Common design
  - Two-level structural configurations
    - Function Table (FT)
    - Boolean Equation Sets (BES)

x	y = f(x + x) y = f(2x) y = f(x << 1)
0	0
1	2
2	4
3	6
…	…
x_m	y_m

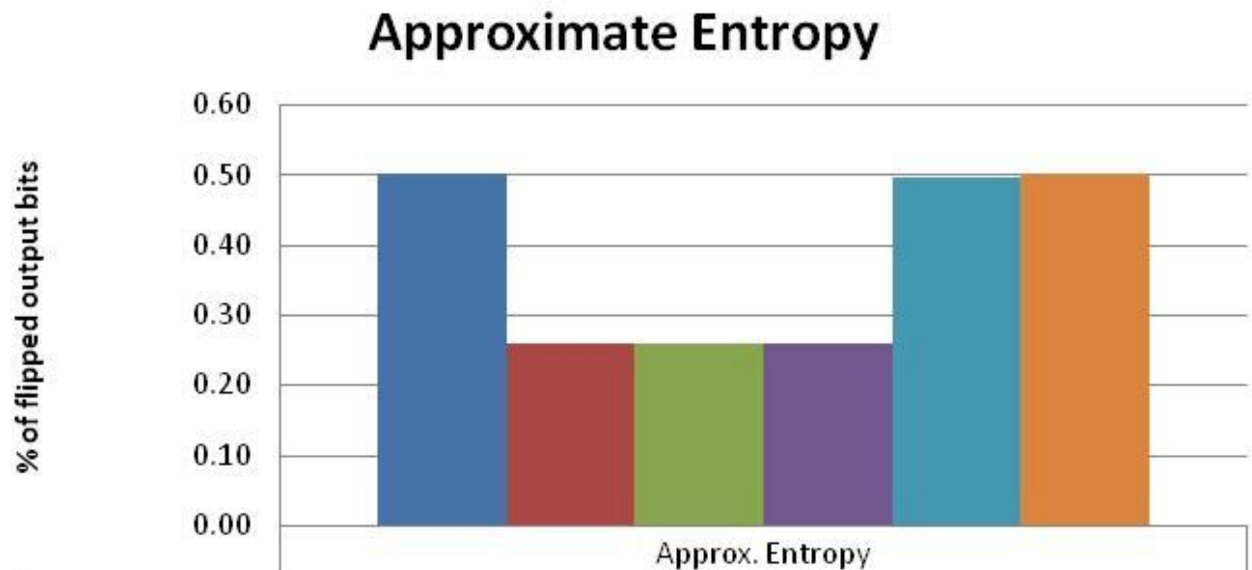
x	y = e(x, k)
0	66E94BD…89E0
1	58E2FCC…455A
2	F795AAA…C1E0
3	0388DAC…FE78
4	8ADE7D8…0291
5	95B84D1B…89E0
6	C94DA219…88F2
…	…
x_n	y_n

Experimental Design

- Complexity based brute-force attack on I/O size
  - Compute all function tables of size *m*-inputs, *n*-outputs
    - Super-exponential process, $O(m^n)$
  - Pair combinations of generated function tables in *m,n*
    - Factorial process, $O(n!)$
  - All operations of a lookup are the same
    - Index search, O(*1*) [or O(n) for BES]
    - No side-channel (performance/cost) leakage

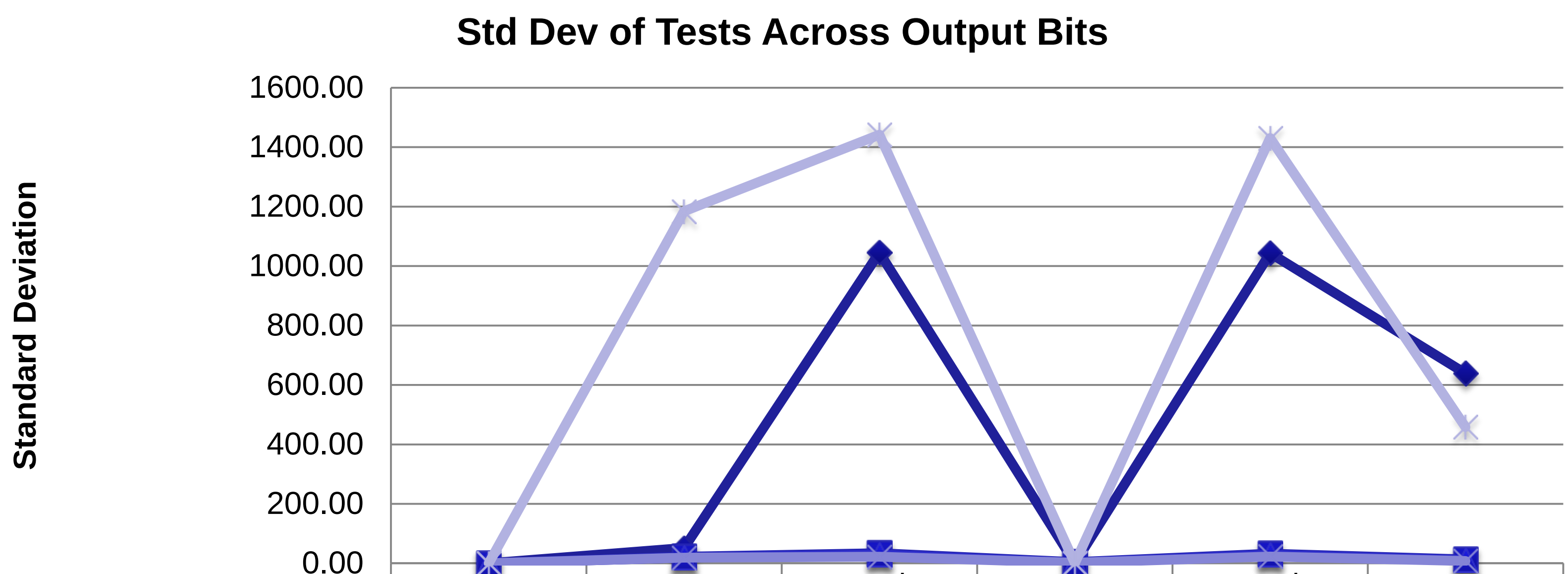
- Memory size
  - Function tables are at least (*n*-input) * (*m*-output) bits
  - Boolean equation sets are at least (*m-output) * p* terms
    - *m* equations stored in text form of *p* terms
    - Exponential size increase n^{pm}

Results and Analysis



	Approx. Entropy
AES	0.50
5-128-500 (GC)	0.26
5-128-1000 (GC)	0.26
c17 no pad	0.26
c17 CTR pad	0.50
c17 INPUT pad	0.50

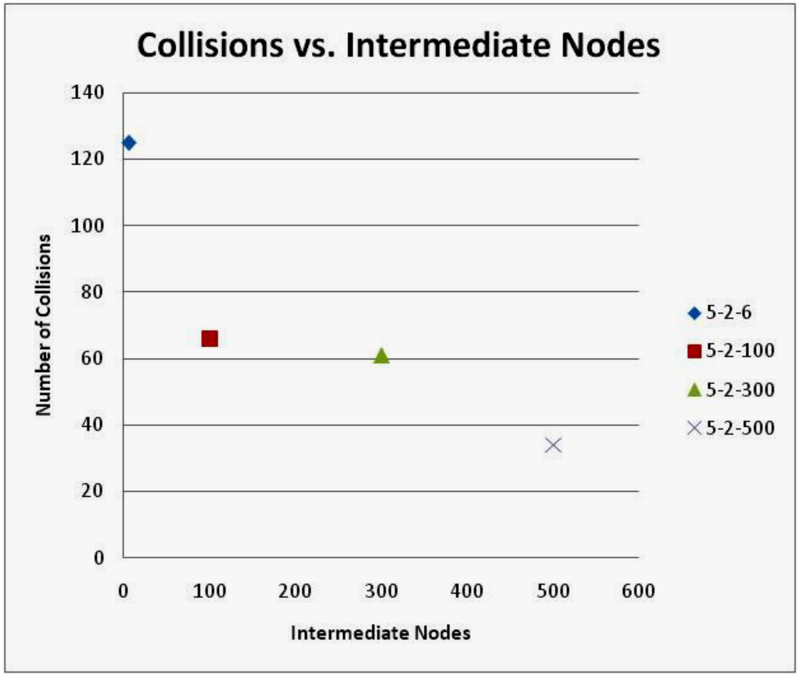
Results and Analysis

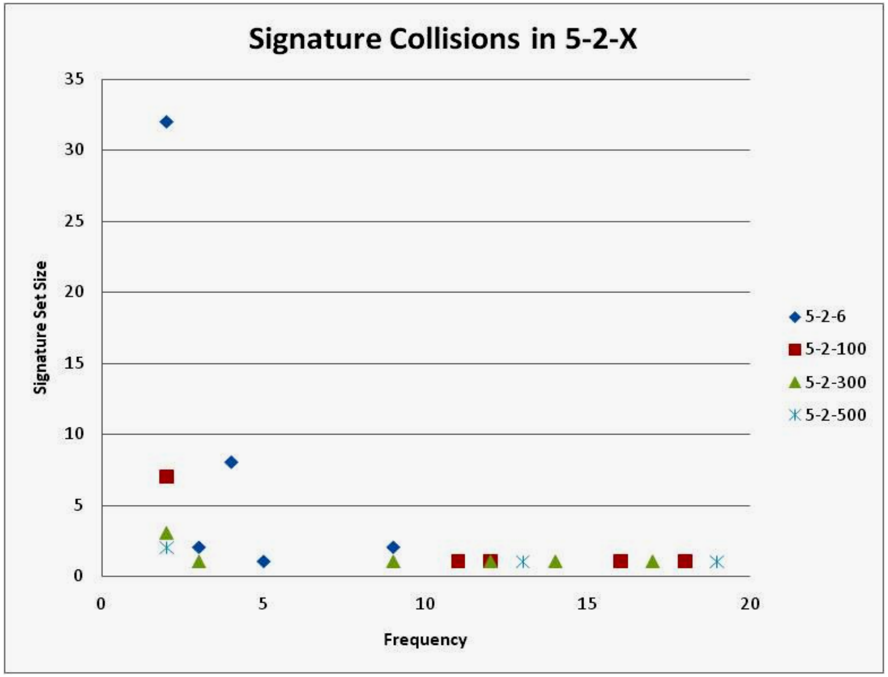


	% of 1's	Longest 1's Runs	Excursions from Zero	Zero Cycles	Excusion States	Runs of 1's
y = a * b + c (CTR)	0.12	51.47	1045.35	0.00	1043.59	639.31
12-8-100 (CTR)	0.01	22.18	33.75	4.18	32.45	13.27
12-8-300 (CTR)	0.01	19.05	28.94	1.77	28.56	10.42
12-8-500 (CTR)	0.01	19.06	22.27	1.66	22.72	7.94
12-8-100 (CTR) 0001	0.31	1184.28	1441.95	0.00	1430.44	458.58

Results and Analysis

- Possible signatures is (2^{output_bits}) ^ (2^{input_bits})
  - Collisions occur at ↑ frequency with ↓ intermediate node size
  - Collisions occur at ↑ concentration with ↑ intermediate node size





Conclusions

- Black-box metrics indicate bits where structural entropy is most needed if we keep function preservation property
  - Structural entropy may be insufficient depending on output pattern
  - Smaller circuits are better choices for random selection

- Enumeration is required—larger *n* requires greater resources upon generation, not execution
  - Advantage to developers with large computational resources
  - Reuse encryption function tables
  - Brute-force attack limited to adversaries with sufficient resources
  - Input/output size is easier to determine than function family

Sponsor

Research sponsorship by:

Air Force Office of Scientific Research (AFOSR)
Information Operations

Questions

?????