CSE 5095 & ECE 4451 & ECE 5451 – Spring 2017

Lecture 9a

- HaTCh follows http://arxiv.org/abs/1605.08413 and https://eprint.iacr.org/2014/943.pdf

HaTCh: State-of-the-Art in Hardware Trojan Detection

Marten van Dijk

Syed Kamran Haider, Chenglu Jin, Phuong Ha Nguyen

Department of Electrical & Computer Engineering

University of Connecticut

UCONN

Secure Computation Laboratory

HaTCh: Advancing the State-of-the-Art in Hardware Trojan Detection

Syed Kamran Haider[†], Chenglu Jin[†], Masab Ahmed[†], Devu Manikantan Shila[‡], Omer Khan[†] and Marten van Dijk[†]

[†]University of Connecticut

[‡]United Technologies Research Center

http://scl.uconn.edu/

UCONN

Outline

- Hardware Trojans and Problem Statement
- Existing Hardware Trojan Detection Techniques
- Characterization of Hardware Trojans
  - Advanced Properties
- HaTCh: Hardware Trojan Catcher
  - Algorithm
  - Comparisons with other schemes
- Evaluation
- Conclusion

What is a Hardware Trojan?

- A malicious logic embedded inside a larger circuit resulting in data leakage or harm to the normal functionality

- Hardware Trojans have two major classes
  - *Trigger Activated*: Activates upon some special internal/external event
  - *Always Active*: Remain always active to deliver the payload

- Several possible payloads
  - Denial of Service
  - Leakage of Sensitive Information
  - Reducing the battery life of the device
  - Weakening of Security mechanisms
    - E.g. bypassing protection circuitry, discard counter measures etc.

Hardware Trojans Examples [1][2]

<table>
<tr><th rowspan="2">Types of Trojan</th><th colspan="2">Trigger</th><th colspan="2">Actors</th><th rowspan="2">Payload / Consequence of attack</th></tr>
<tr><th>Actor</th><th>Action</th><th>Input Channel</th><th>Output/Leaking channel</th></tr>
<tr><td rowspan="3">Trigger Activated</td><td rowspan="2">Attacker with physical access to the device</td><td>▪ Particular legitimate input sequence
▪ Particular illegitimate input sequence</td><td>Standard Input
▪ I/O pins
▪ Keyboard
▪ Serial/Parallel protocols</td><td rowspan="3">Standard / Unused Outputs
▪ I/O pins
▪ LCD
▪ LEDs
▪ Serial/Parallel protocols

Side Channels
▪ EM Waves
▪ Hidden in standard output</td><td rowspan="3">Leaking sensitive information
▪ Encryption Key
▪ Plain text

Denial of service
▪ Generating incorrect results
▪ Make the device stop working

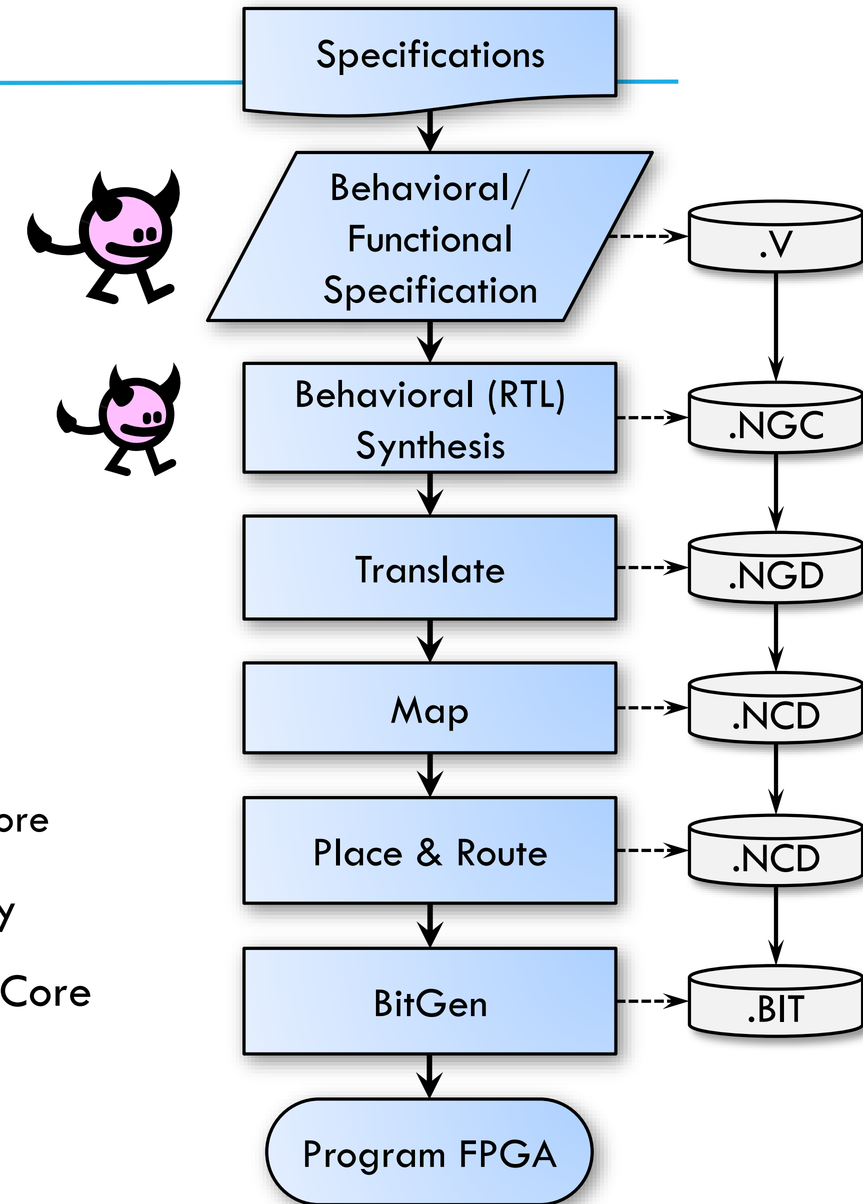
Reduce the reliability of the device
▪ Drain the battery</td></tr>
<tr><td>▪ Taking control through unused functional units or interfaces</td><td>Unused Inputs
▪ I/O pins
▪ Serial/Parallel protocols</td></tr>
<tr><td>Legitimate User</td><td>▪ Normal operation for certain n>N
▪ Particular legitimate input sequence
▪ Illegitimate input sequence by mistake
▪ Certain time interval between two legal inputs</td><td>Standard Input
▪ I/O pins
▪ Keyboard
▪ Serial/Parallel protocols</td></tr>
<tr><td>Always Active</td><td>N/A</td><td>N/A</td><td>Internal IP Core</td><td>Side Channels
▪ EM Waves</td><td>Leak the Encryption Key</td></tr>
</table>

[1] Y. Jin, "Experiences in Hardware Trojans Design and Implementation"
[2] G. Becker, "Implementing Hardware Trojans"

FPGA Design Flow

- IP Core Design Steps
  1) System level design is modeled in C/C++, MATLAB etc.
  2) RTL design is modeled in some HDL e.g. Verlilog, VHDL
  3) Xilinx FPGA Design Flow takes HDL Design entry
     a) **Synthesis:** Creates Xilinx-specific Netlist → NGC file
     b) **Translate:** Reduces logical design to Xilinx primitives
     c) **Map:** Maps the design on target FPGA
     d) **Place & Route:** Places & routes the design to meet timing
     e) **BitGen:** Produces a BIT file to program the FPGA

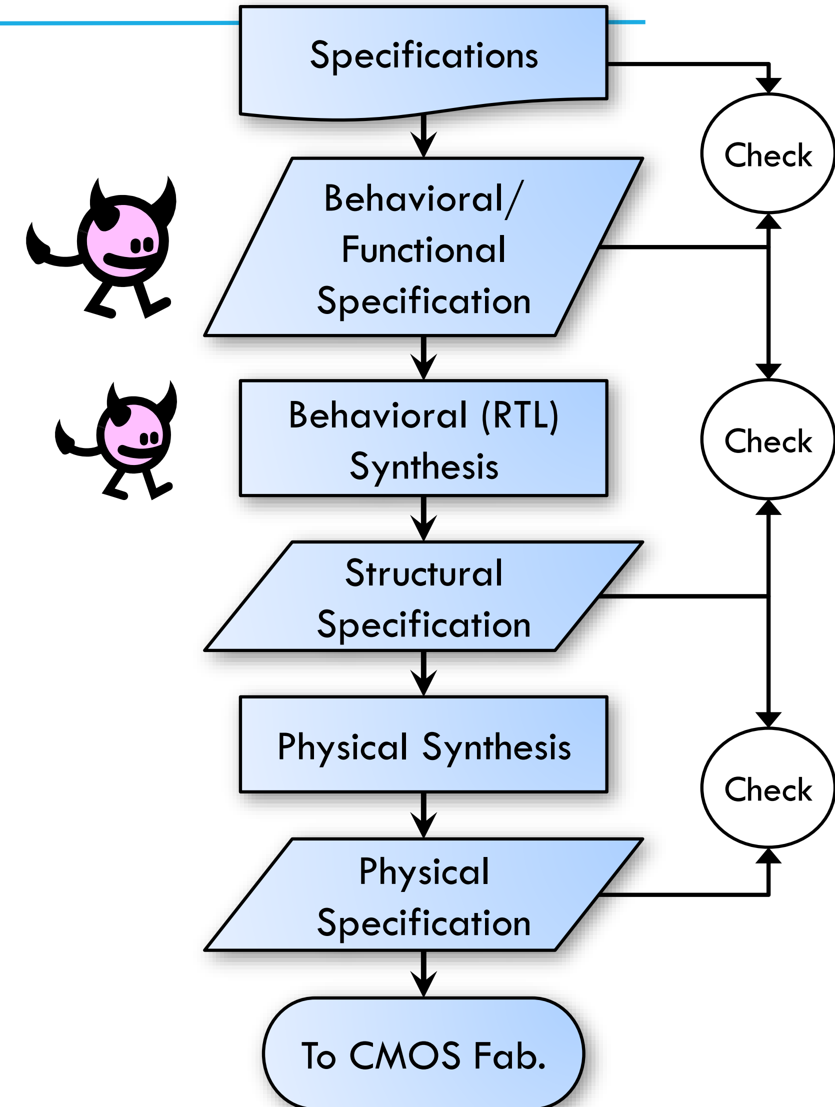
- In practice the NGC netlist of the IP Core is provided to the customer
  - It still hides the source code
  - Possibility to include the NGC netlist in a larger design
  - Note: This means that rest of the Toolchain is in control of the customer and can therefore be trusted
- We define access to a Closed Source IP Core as access to the NGC netlist only
- We assume in the remainder that the customer has access to Closed Source IP Core
- Hardware Trojans can be embedded in the IP Core



ASIC Design Flow

- Generalized ASIC Design FLow
  - High Level Design
    - Specification Capture
    - Design Capture in C, C++, SystemC or SystemVerilog
  - RTL Design
    - Verilog/VHDL
  - System, Timing and Logic Verification
    - Is the logic working correctly?
  - Physical Design
    - Floorplanning, Place and Route, Clock insertion
  - Performance and Manufacturability Verification
    - Extraction of Physical View
    - Verification of timing and signal integrity
    - Design Rule Checking/ LVS

- In practice the RTL Synthesized netlist of the IP Core is provided to the customer
- Similar to FPGAs, Hardware Trojans can be embedded in the IP Cores



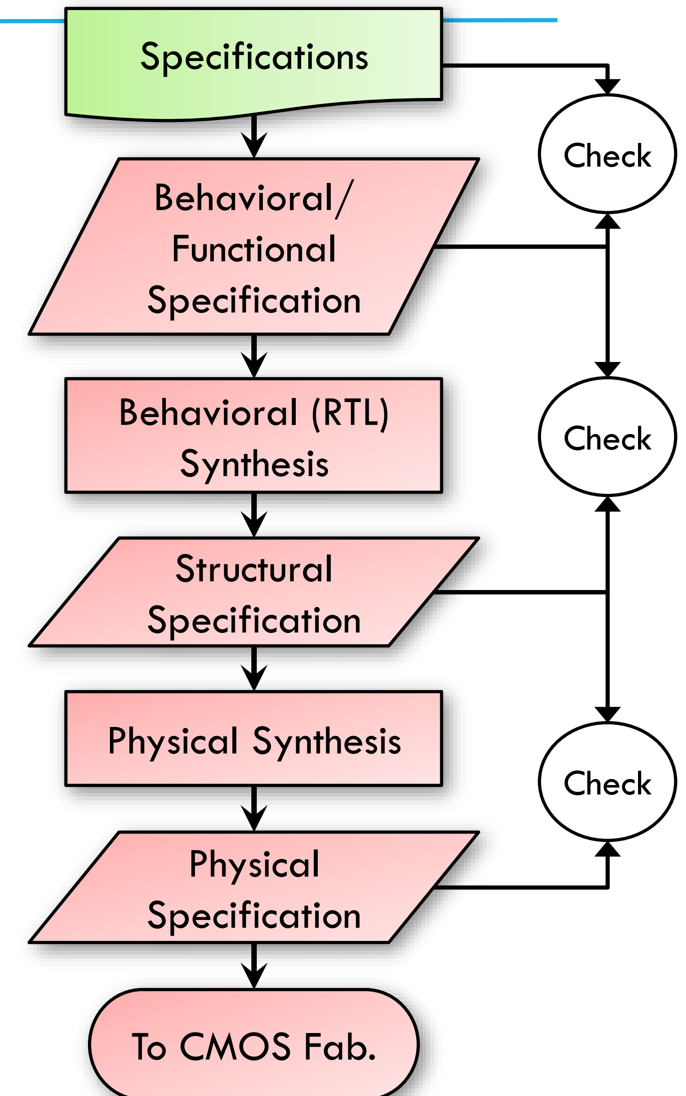
Design Flow Vulnerabilities

Untrusted Source Code:

- Third party only provides netlist file (e.g. NGC) of the IP Core
  - A Trojan could have been implanted in the source code
  - Netlist file obfuscates HDL source code
  - Hard to detect an embedded Trojan

Untrusted Toolchain:

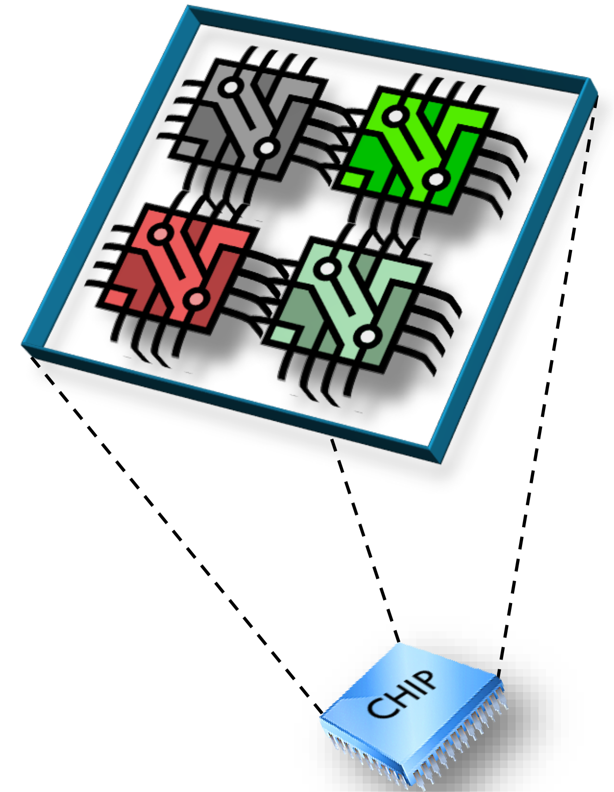
- Toolchain used to generate Netlists could be malicious†
  - User trusts only a finite set of trustworthy Tools
    - E.g. Cadence, Synopsis, Xilinx etc
  - IP Core provider may not use these trustworthy tools



† TURING AWARD LECTURE "Reflections on Trusting Trust" by Ken Thompson

Problem Statement & Motivation

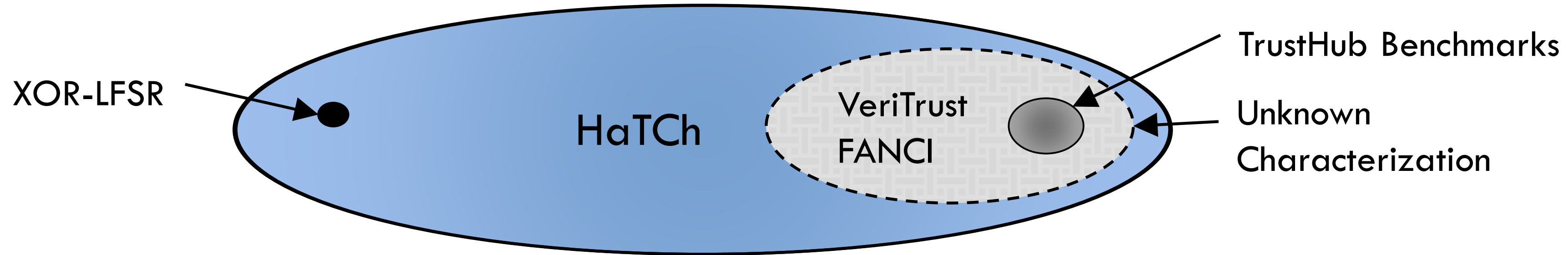
- IP cores are heavily used in modern systems
  - IP cores are vulnerable to insertion of Hardware Trojans (HTs)
- State of the art HT detection schemes have either of the following two limitations
  1. *They can be defeated by new 'sophisticated' HTs.*
  2. *They have infeasibly high computational complexity.*

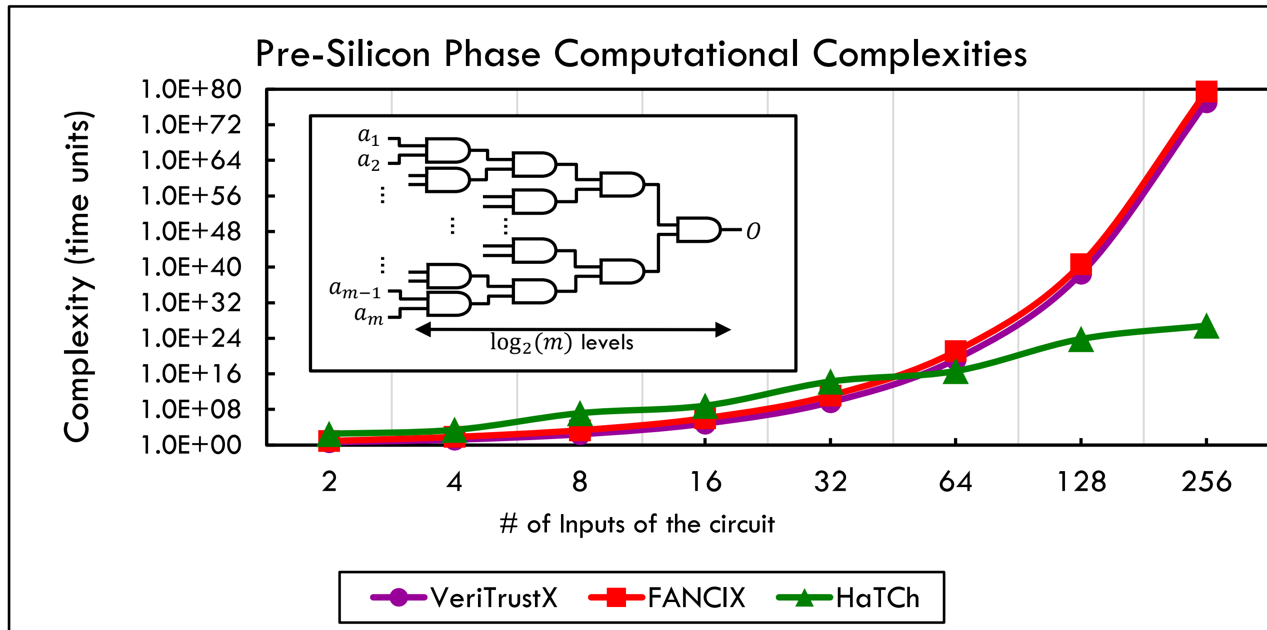


This leads to the following two questions:

1. *Which exponentially large class of HTs a tool can detect with negligible false negative rate?*
2. *How to design an efficient detection tool with controlled false positive rate which is computationally feasible for this large class of HTs?*

The Big Picture





	False Positives	False Negatives
VeriTrust FANCI	Uncontrollable	Unknown Characterization
HaTCh	Controllable ρ	Controllable $2^{-\lambda}$ for $H_{d,t,\alpha}$

Outline

- Hardware Trojans and Problem Statement
- Existing Hardware Trojan Detection Techniques
- Characterization of Hardware Trojans
  - Advanced Properties
- HaTCh: Hardware Trojan Catcher
  - Algorithm
  - Comparisons with other schemes
- Evaluation
- Conclusion

Existing Trojan Detection Schemes

- **Unused Circuit Identification (UCI)**
  - Distinguishes minimally used logic from the other parts of the circuit.
  - Intuition that a HT almost always remains inactive in the circuit to pass the functional verification.
- **VeriTrust**
  - Detects HTs by identifying redundant inputs for the normal functionality of the output wire.
  - First the activation history of the inputs is recorded in SOP and POS form.
  - Further analysis of SOPs and POSs yields the redundant inputs.
- **FANCI**
  - Applies Boolean function analysis.
  - Flags suspicious wires which have weak input-to-output dependency determined by *Control Value*.

Extended VeriTrust & FANCI

- **DeTrust** introduces a new HT design methodology that defeats VeriTrust & FANCI [CCS'14]

Problem 1: Current schemes can be bypassed by "Sophisticated" Trojans

- DeTrust also proposes extensions to VeriTrust & FANCI to detect the new HTs
- Extended VeriTrust **(VeriTrustX)** & Extended FANCI **(FANCIX)**
  - Key idea: The circuits should be monitored up to multiple sequential stages at a time, while ignoring any FFs in between.

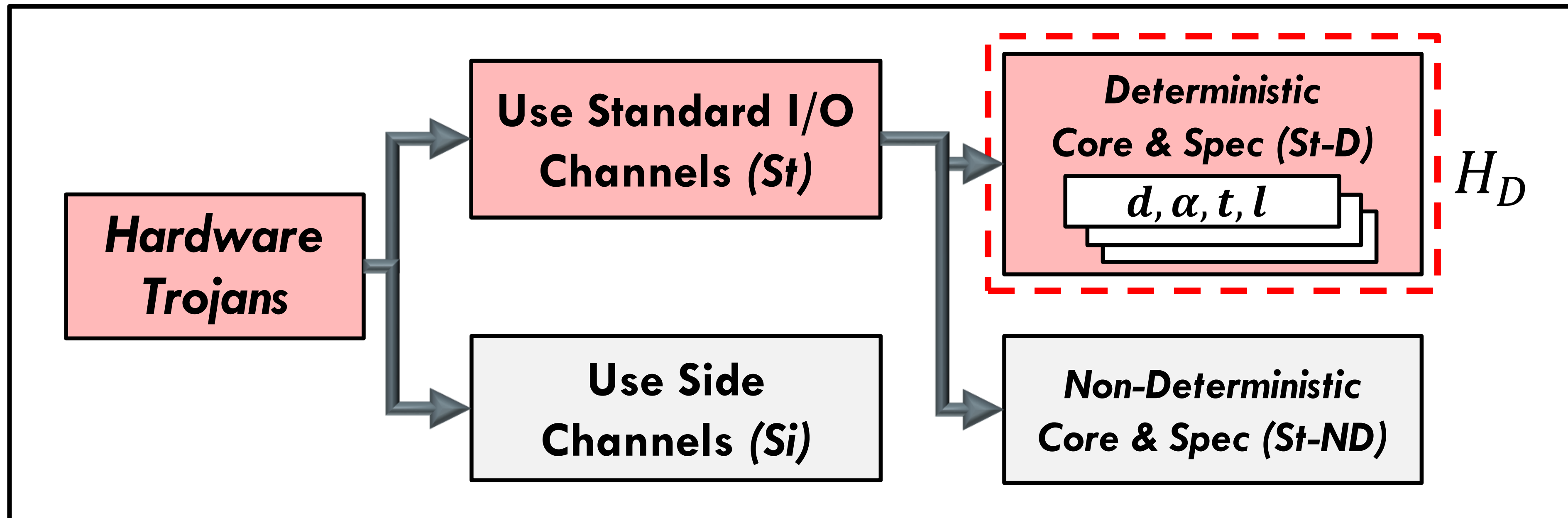
Problem 2: Current schemes can have infeasibly High Computational Complexity

Outline

- Hardware Trojans and Problem Statement
- Existing Hardware Trojan Detection Techniques
- Characterization of Hardware Trojans
  - Advanced Properties
- HaTCh: Hardware Trojan Catcher
  - Algorithm
  - Comparisons with other schemes
- Evaluation
- Conclusion

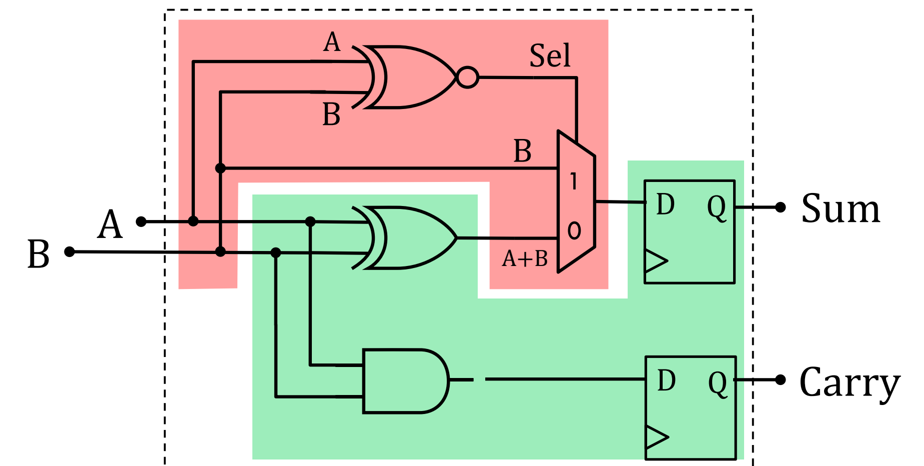
Characterization of Hardware Trojans

- A detailed characterization of Hardware Trojans that defines the scope of HaTCh at the huge landscape of Hardware Trojans.



Explicit vs. Implicit Malicious Behavior

- **Explicit malicious behavior** refers to a behavior of a HT where the HT generated output is distinguishable from a normal output.
  - $\mathrm{A} = 1,\ \mathrm{B} = 1 \rightarrow \mathrm{Sum} = \mathrm{B} = 1 \neq 0$
- **Implicit malicious behavior** refers to a behavior of a HT where the HT generated output is indistinguishable from a normal output.
  - $\mathrm{A} = 0,\ \mathrm{B} = 0 \rightarrow \mathrm{Sum} = \mathrm{B} = 0 = 0$

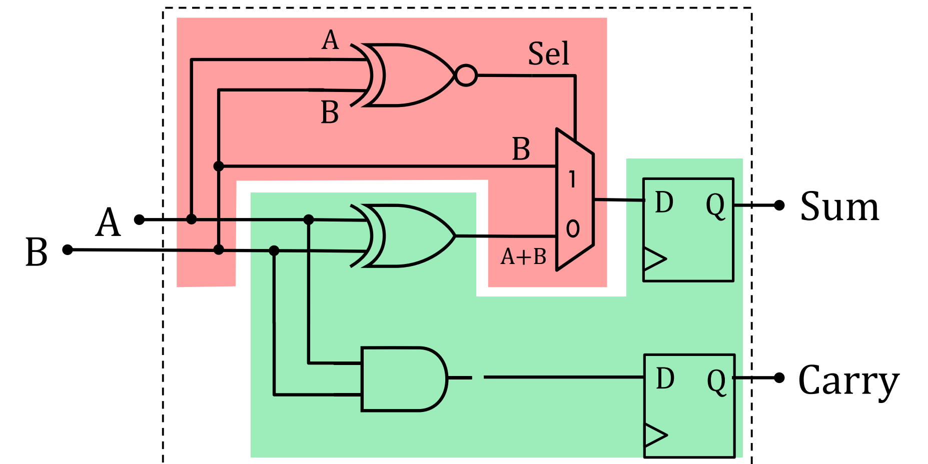


Trigger: *A=B*
Payload: *Sum=B when A=B*

Implicit Malicious behavior can be exploited to
bypass the countermeasure!!!

Properties of Deterministic HTs Group H_D

- **Trigger Signal Dimension d:**
  Number of Trigger Signal Wires
  - E.g. 1 bit trigger signal $'\mathrm{Sel}'$
- **Payload Propagation Delay t:**
  Cycles taken to propagate malicious behavior to the output port after Trigger
  - E.g. 1 cycle taken by $'\mathrm{Sum}'$ after $\mathrm{Sel} = 1$
- **Implicit Behavior Factor α:**
  Probability of Implicit Malicious Behavior given that the Trojan is triggered.
  - 50% for the example Trojan, since
    $\mathrm{A} = \mathrm{B} = 0$ → $\mathrm{Sum} = \mathrm{B} = 0 = 0$ and
    $\mathrm{A} = \mathrm{B} = 1$ → $\mathrm{Sum} = \mathrm{B} = 1 \neq 0$



Trigger: *A=B*
Payload: *Sum=B when A=B*

Properties of Deterministic HTs Group H_D

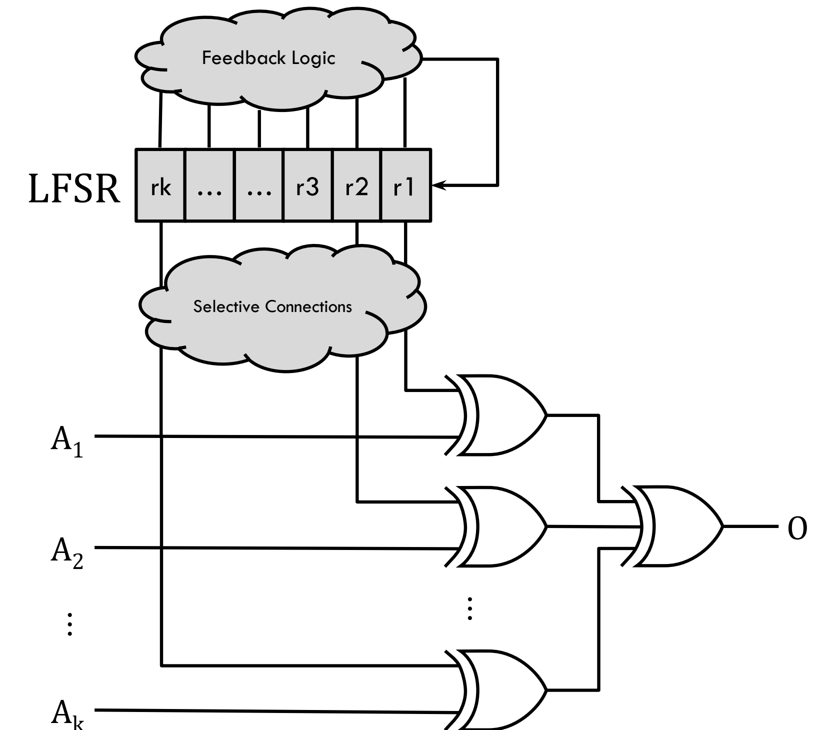
A set T of trigger states represents a HT if the HT always passes through one of the states in T in order to express implicit of explicit malicious behavior.

- **Trigger Signal Dimension $\boldsymbol{d(T)}$** of a HT is defined as $d(T) = \max_{Trig \in T} |Trig|$
- **Payload Propagation Delay $\boldsymbol{t(T)}$** of a HT represented by a set of trigger states T is defined as the maximum number of clock cycles taken to propagate the malicious behavior after entering a trigger state in T.
- **Implicit Behavior Factor $\boldsymbol{\alpha(T)}$** of a HT represented by the set of trigger states T is defined as $\alpha(T) = \max_{Trig \in T} \alpha(Trig)$ where $\alpha(Trig)$ shows the probability that, given the trigger state $Trig$ occurs, it will lead to *implicit* malicious behavior.

- $H_{d,t,\alpha}$ is the set of all H_D type Trojans which can be represented by a set of trigger states T with $d(T) \leq d$, $t(T) \leq t$, and $\alpha(T) \leq \alpha$.

k-XOR-LFSR Hardware Trojan

- A counter based trojan with the counter implemented as an LFSR
- Let $r^i \in \{0,1\}^k$ denote its register content at clock cycle i represented as a binary vector of length *k*.
- Suppose that *u* is the maximum index for which the linear space *L* generated by vectors $r^0, \ldots, r^{u-1}$ (modulo 2) has dimension $k-1$
- Since $\dim(L) = k-1 < k = \dim(\{0,1\}^k)$, there exists a vector $v \in \{0,1\}^k$ such that
  - $\langle v, r^i \rangle = 0$ (modulo 2) for all $0 \leq i \leq u-1$ and
  - $\langle v, r^u \rangle = 0$ (modulo 2)
- Only the register cells corresponding to $v_j = 1$ are being XORed with inputs A_j .



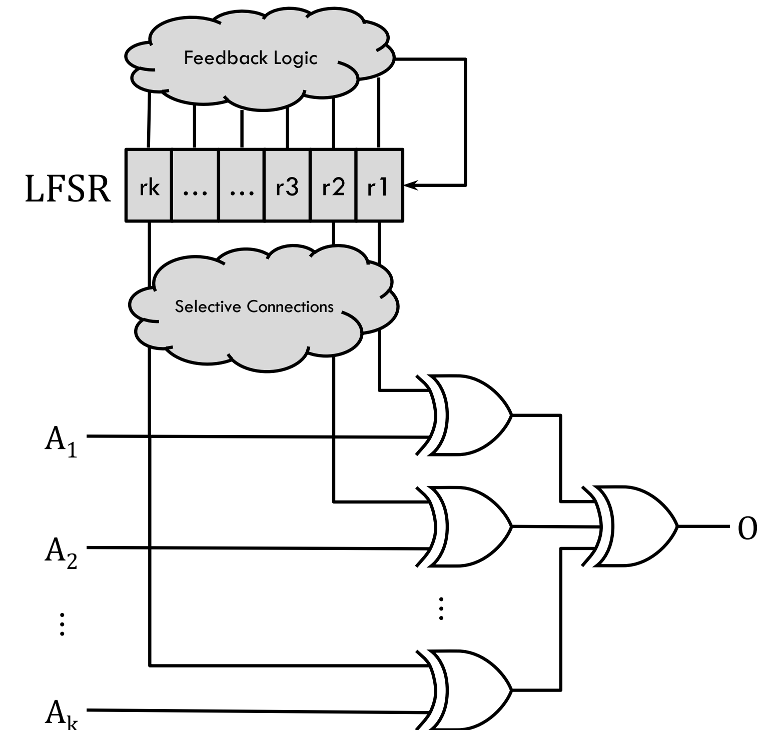
k-XOR-LFSR Hardware Trojan

- Since the A_j are all XORed together in the specified logical functionality to produce the sum $\sum_j A_j$ the Trojan changes this sum to

$$\sum_j A_j \oplus \sum_{j:v_j=1} r_j^i = \sum_j A_j \oplus \langle v, r^i \rangle$$

I.e., the sum remains unchanged until the u-th clock cycle when it is maliciously inverted

- Notice that the dimension d of this Trojan is independent of the inputs A_j
- Therefore in this sense, the *k-XOR-LFSR* trojan is universally applicable to cores that implement an XOR over k inputs.



Suppose that all vectors r_i behave like random vectors from a uniform distribution.

- *Then k-XOR-LFSR has register size k and triggers after $u \approx k$ LFSR transitions (can be clocked at slow rate).*
- *Furthermore, if k-XORLFSR is in $H_{d,t,\alpha}$ then $\alpha = 0$ and with significant probability $d \geq \log(k-t) - \log(\log(k-t)\log k)$.*

Outline

- Hardware Trojans and Problem Statement
- Existing Hardware Trojan Detection Techniques
- Characterization of Hardware Trojans
  - Advanced Properties
- HaTCh: Hardware Trojan Catcher
  - Algorithm
  - Comparisons with other schemes
- Evaluation
- Conclusion

HaTCh: Hardware Trojan Catcher

Hardware Trojan Catcher (HaTCh) processes an IP Core in two phases;

- ***Learning Phase*** puts the core (represented by a netlist) through functional testing and returns a blacklist *B* of unused wire combinations.
  - If no malicious behavior is observed during the learning phase, then the tagging phase starts
  - Otherwise the IP core potentially contains a hardware Trojan and is rejected straightaway
- ***Tagging Phase*** adds extra logic for each entry in the blacklist for runtime detection
  - Whenever any of the blacklisted wires is activated, an exception is raised to indicate the activation of a Trojan.

HaTCh Algorithm

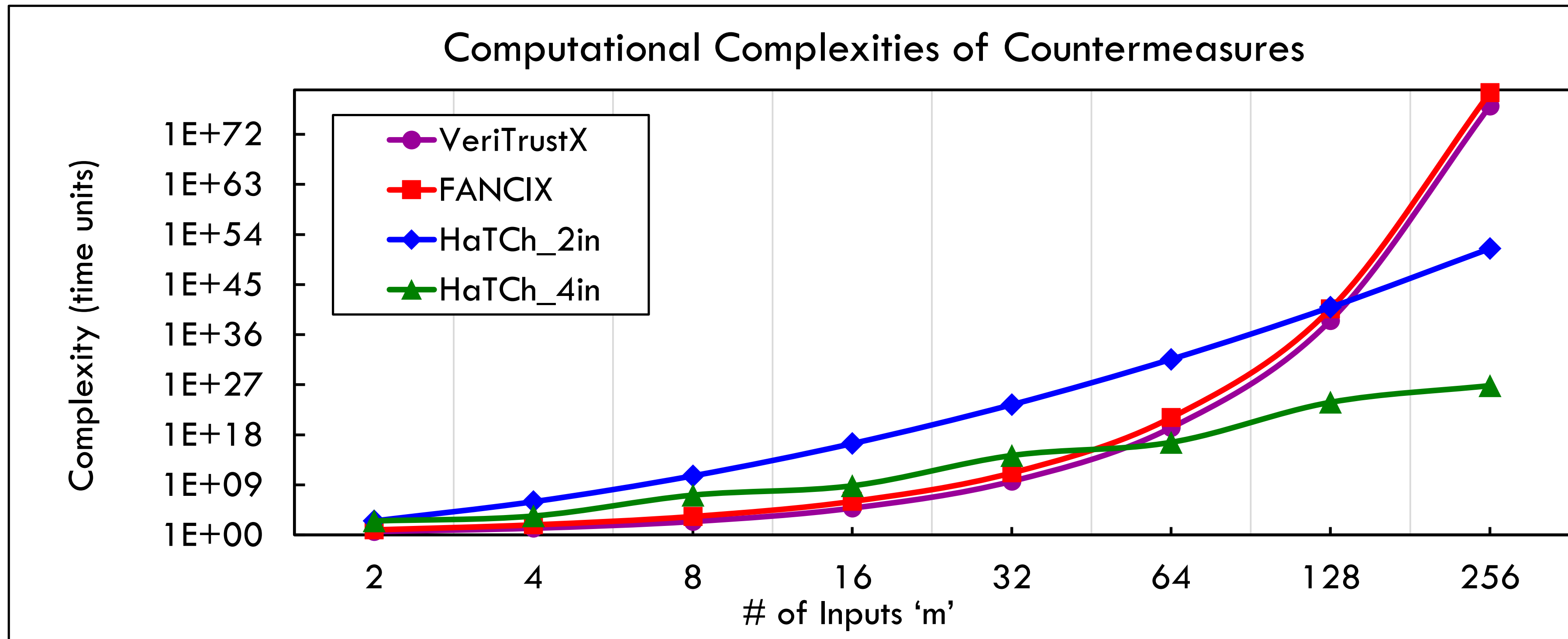
- Learning Phase
  - A simulator is used to produce expected outputs
  - An emulator runs actual IP core circuit
  - k independent blacklists are created
  - Final blacklist is a union of k blacklists
- Tagging Phase
  - Additional circuitry is added
  - Blacklisted wires are tracked
  - Run-time detection
- Complexity
  - $O\left(\frac{\lambda}{\log_2\left(\frac{1}{\alpha}\right)} \cdot \frac{\left(2n^2\right)^d}{\rho/\Delta}\right)$

Learning Phase (lines "for all" through "end for")

procedure HaTCh$(Core, U, t, d, \alpha, \lambda, \rho)$
  $k = \left\lceil \frac{\lambda}{\log_2(1/\alpha)} \right\rceil, B = \phi$
  for all $1 \leq i \leq k$ *do*
    $B_i \leftarrow LEARN(Core, U, t, d, \rho)$
    if $B_i =$ Trojan Detected *then*
      return Trojan Detected
    else
      $B = B \cup B_i$
    end if
  end for
  $Core_{Protected} = TAG(Core, B)$
  return $Core_{Protected}$
end procedure

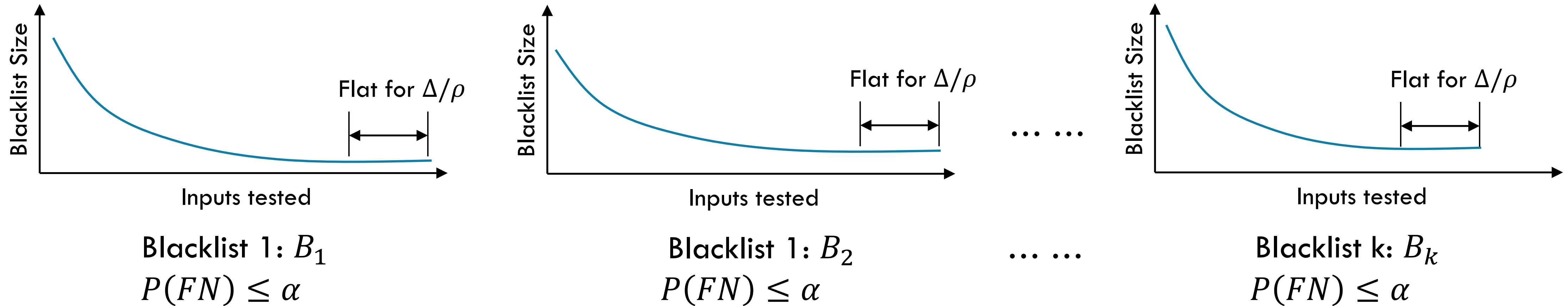
Computational Complexity Comparison

- VeriTrust: $O(2^m)$ where m indicates the number of inputs of the Trojan circuit.
- FANCI: $O(m2^m)$ where m indicates the number of inputs of the Trojan circuit.
- HaTCh: $O((2n^2)^d)$ where:
  - $n = 2m - 1$ and $\mathrm{d} = \log_2(m) + 1$ for 2-input implementation
  - $n = m + \left\lceil \frac{m-1}{3} \right\rceil$ and $\mathrm{d} = \lceil \log_4 m \rceil$ +1 for 4-input implementation



False Positives

- VeriTrust & FANCI → Uncontrollable
- HaTCh → Controllable, i.e. ρ



- Final Blacklist $B = B_1 \cup B_2 \cup \cdots \cup B_k$
  - Probability of False Negative $P(FN) \leq \alpha^k \leq 2^{-\lambda}$
  - False positives rate $= \rho$

Statistical assumptions: (1) With probability at least 0.5 testing another Δ/ρ inputs would not reduce any of the k blacklists. (2) States corresponding to the same test input that are separated by Δ cycles are statistically independent. (3) The state distribution is statistically independent of the cycle number at which the state occurs. (4) The learning phase samples the real input distribution closely.

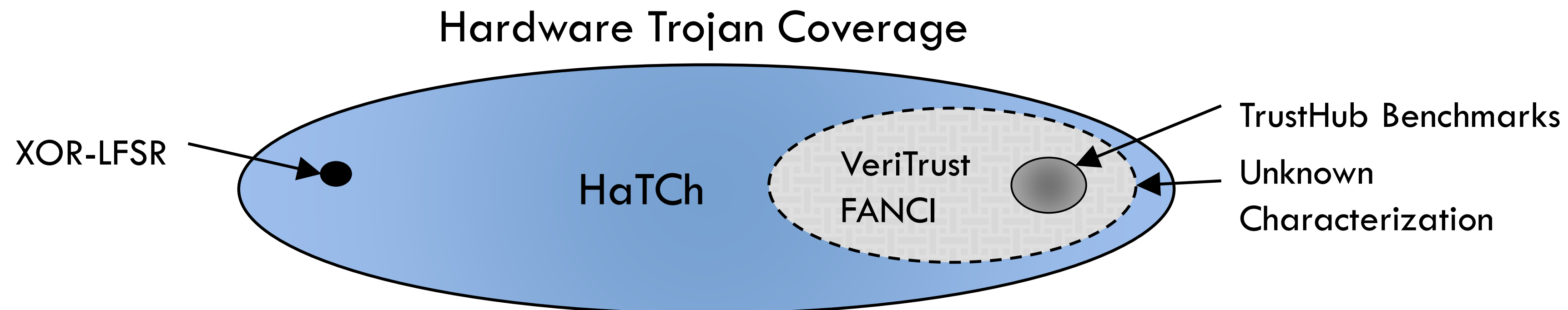
False Negatives

- False Negative Rate of a set of Hardware Trojans
  - $FNR(H) = \frac{1}{|H|}\sum_{h \in H} Prob(\text{h is not detected when triggered})$

- VeriTrust & FANCI
  - TrustHub → $FNR(\text{TrustHub}) = 0$
  - Other Hardware Trojans → No Characterization

- HaTCh
  - TrustHub → $FNR(\text{TrustHub}) = 0$
  - Other Hardware Trojans → Controllable, $FNR(\text{H}_{\text{d,t},\alpha}) \leq 2^{-\lambda}$

Hardware Trojan Coverage



Outline

- Hardware Trojans and Problem Statement
- Existing Hardware Trojan Detection Techniques
- Characterization of Hardware Trojans
  - Advanced Properties
- HaTCh: Hardware Trojan Catcher
  - Algorithm
  - Comparisons with other schemes
- Evaluation
- Conclusion

Evaluation

- We first characterize the benchmarks from TrustHub[‡] w.r.t. the Hardware Trojan characterization introduced in our framework
- Then we evaluate HaTCh for the following benchmarks:
  - S-Series Benchmarks from TrustHub: s15850, s35932 and s38417
  - RS232 Benchmarks from TrustHub
  - New Hardware Trojans presented by DeTrust which defeat FANCI & VeriTrust
  - A newly designed XOR-LFSR Hardware Trojan
- HaTCh detects all these Trojans
  - The corresponding area overheads for relevant benchmarks are presented next.

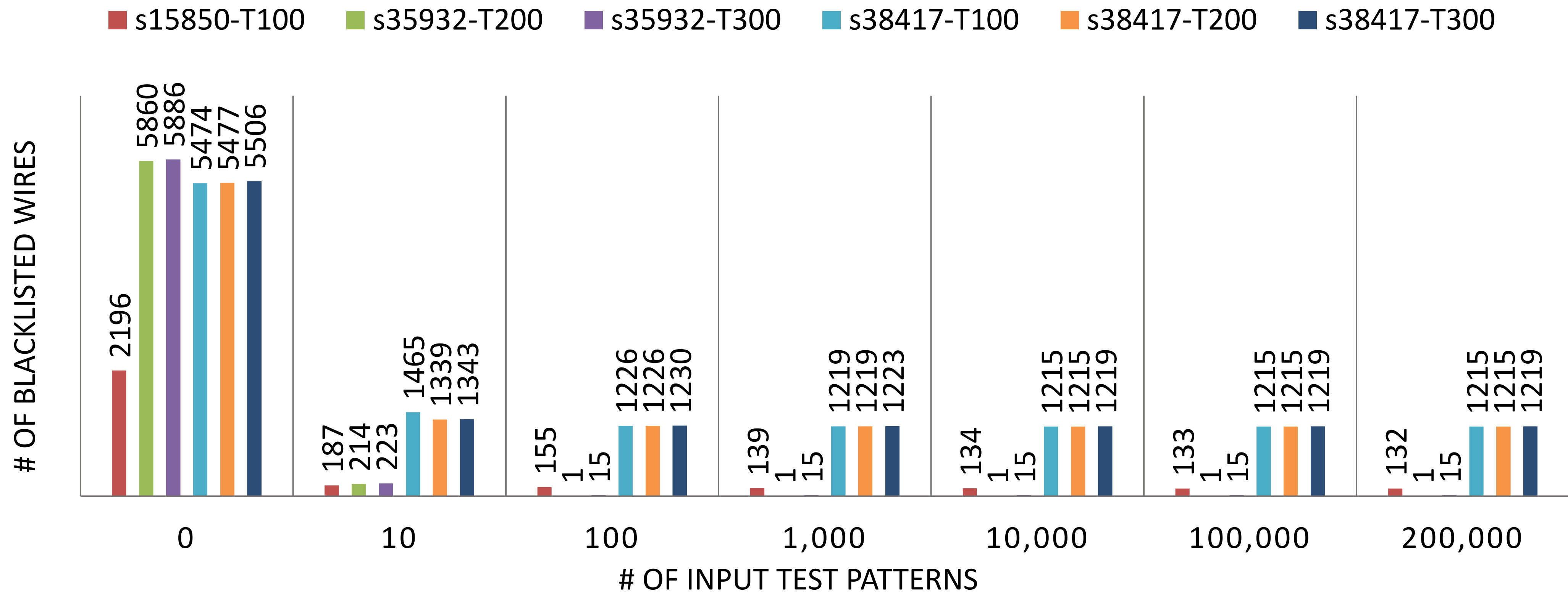
[‡] M. Tehranipoor, R. Karri, F. Koushanfar, and M. Potkonjak, "Trusthub," http://trusthub.org

Characterization of TrustHub

Type		t	α	*Benchmarks*
St	***D***	0	$1/2^{32}$	BasicRSA-T{100, 300}
			0.5	s15850-T100, s38584-T{200, 300}
			0-0.25	wb_conmax-T{100, 200, 300}
			0-0.87	RS232-T{100, 800, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1900, 2000}
		1	0.5	b15-T{300,400}
			0.5-0.75	s35932-T{100, 200}
			0-0.06	RS232-T{400, 500, 600, 700, 900, 901}
		2	0.5	vga-lcd-T100, b15-T{100, 200}
			0.87	s38584-T100
		3	$1/2^{32}$	BasicRSA-T{200, 400}
			0.5	s38417-T100
		5	0.99	s38417-T200
		7	0.5	RS232-T300
		8	0.5	s35932-T300
	ND	N/A		MC8051-T{200, 300, 400, 500, 600, 700, 800}, PIC16F84-T{100, 200, 300, 400}
Si		N/A		AES-T{400, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 2000, 2100}, s38417-T300, AES-T{100, 200, 300}

$d = 1$

Experimental Results for S-Series



Area Overhead for S-Series

TABLE 2. AREA OVERHEAD FOR S-SERIES WITH $d = 1$

Benchmark	*Size*	*Area Overhead*	
		Pipelined	*Non-Pipelined*
s15850-T100	2180	4.17%	2.11%
s35932-T200	5442	0.02%	0.02%
s35932-T300	5460	0.16%	0.09%
s38417-T100	5341	15.22%	7.62%
s38417-T200	5344	15.21%	7.62%
s38417-T300	5372	15.25%	7.63%
Average		8.34%	4.18%

Area Overhead for RS232

TABLE 3. AREA OVERHEAD FOR RS232 WITH $d = 2, l = 1$

Benchmark	*Size*	*Area Overhead (non-pipelined)*
RS232-T300	280	2.50%
RS232-T1200	273	0.73%
RS232-T1300	267	0.75%

Conclusion

- We introduce a thorough characterization and certain advanced properties of Hardware Trojans which provide crucial information for the development of detection tools
  - The benchmarked Hardware Trojans turn out to be of the simplest kind and must only reflect the tip of the iceberg

- We propose and implement HaTCh, a powerful hardware detection tool which
  - Detects all benchmarked Trigger based deterministic Hardware Trojans
  - Detects exponentially large Hardware Trojan classes with negligible probability of a false negative
  - Offers sub-exponential computational complexity as opposed to exponential complexity of existing schemes
  - Has low area overhead

Thank you!

See http://scl.uconn.edu/research/htdd.php for more details with links to http://arxiv.org/abs/1605.08413 and https://eprint.iacr.org/2014/943.pdf