Using Scan Side Channel to Detect IP Theft

Leonid Azriel, Ran Ginosar, Avi Mendelson
Technion – Israel Institute of Technology
Shay Gueron,
University of Haifa and Intel Israel
Outline

• IP theft issue in SoC
• Reverse Engineering with Scan
• Junta Learning
• Clustering and Graph Completion
• The Test Case: BitCoin SHA-256
• Conclusions
IP Piracy

- Modern SoC development mode: global and distributed
- IP passes dozens of hands
- Issue of Trust
Preventing IP theft

• **Watermarks** – allow identification without altering the function
  – State Machine Encoding
  – Constraints on physical layout
  – More...
  – Detection
  – Proof

• **Forensic techniques**
  – Direct detection
Outline

- IP theft issue
- Reverse Engineering with Scan
- Junta Learning
- Clustering and Graph Completion
- The Test Case: BitCoin SHA-256
- Conclusions
Reverse Engineering of an ASIC

- Phase 1 – Invasive Physical → Circuit
  - Delayering
  - SEM
  - Nanoscale Imaging
  - Cross-section

- Phase 2 – Algorithmic Circuit → Spec
  - FSM Extraction
  - Model Checking
  - SAT
Reverse Engineering of an ASIC

• Phase 1 – Invasive Physical ➔ Circuit
  – Delayering
  – SEM
  – Nanoscale Imaging
  – Cross-section

Scan Side Channel makes phase 1 non-invasive

• Phase 2 – Algorithmic Circuit ➔ Spec
  – FSM Extraction
  – Model Checking
  – SAT Solvers
The Scan Technique

Goal: automate production testing
The Scan Technique

Need to verify every net is functional
The Scan Technique

Sequential Cells
(FFs / Latches)
The Scan Technique

Scan Insertion
The Scan Technique

Production Tester

Shift In
The Scan Technique

Production Tester

Capture
The Scan Technique

Production Tester

Shift Out
Unfolding Sequential Circuits with Scan

- Scan turns the SoC to a stateless circuit
- Mapped to the Boolean Function Learning problem: $\{0,1\}^n \rightarrow \{0,1\}^n$
Unfolding Sequential Circuits with Scan

- Scan turns the ASIC to a stateless circuit
- Mapped to the Boolean Function Learning problem: \( \{0,1\}^n \rightarrow \{0,1\}^n \)
- Exhaustive Search: Extract the Truth Table by running queries for all inputs

\[
F = \begin{array}{cccc}
0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & . \\
. & . & . & . \\
. & . & . & . \\
. & . & . & . \\
\end{array}
\]
Unfolding Sequential Circuits with Scan

- Scan turns the ASIC to a stateless circuit
- Mapped to the **Boolean Function Learning** problem: $\{0,1\}^n \rightarrow \{0,1\}^n$
- Exhaustive Search: Extract the Truth Table by running queries for all inputs
- **Exponential Size**: $2^n$

$$F = \begin{array}{cccc}
0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 \\
0 & 1 & 1 & .
\end{array}$$
Outline

• IP theft issue
• Reverse Engineering with Scan
• Junta Learning
• Clustering and Graph Completion
• The Test Case: BitCoin SHA-256
• Conclusions
Limited Transitive Fan-in

• In practice, logic cones have limited number of inputs: Transitive Fan In = K
Dependency Graph

- Bipartite graph represents flip-flop dependencies
- The goal: Find dependencies
- Complexity: $2^n \rightarrow 2^k$: Scalable with the chip size
The K-Junta Algorithm

\[ y = f(\bar{x}), \bar{x} = \{ x_1, x_2, \ldots, x_i, x_{i+1}, \ldots, x_j, \ldots, x_n \} \]

Generate random queries \( y = f(\bar{x}) \)
The K-Junta Algorithm

\[ y = f(\vec{x}), \vec{x} = \{x_1, x_2, \ldots, x_i, x_{i+1}, \ldots, x_j, \ldots, x_n\} \]

\[ \vec{a} = \{0, 0, \ldots 0, 0, 0, 0, \ldots, 0, 0\}, f(\vec{a}) = 0 \]

Generate random queries \[ y = f(\vec{x}) \]

\[ \vec{b} = \{1, 0, \ldots 1, 0, 1, 0, 0, \ldots, 0, 1\}, f(\vec{b}) = 1 \]
The K-Junta Algorithm

\[ y = f(\bar{x}), \bar{x} = \{x_1, x_2, \ldots, x_i, x_{i+1}, \ldots, x_j, \ldots, x_n\} \]

\[ \bar{a} = \{0, 0, \ldots 0, 0, 0, 0, 0, \ldots, 0, 0\}, f(\bar{a}) = 0 \]

\[ \bar{a} = \{1, 0, \ldots 0, 0, 0, 0, 0, \ldots, 0, 0\}, f(\bar{a}) = 0 \]

\[ \bar{b} = \{1, 0, \ldots 1, 0, 1, 0, 0, \ldots, 0, 1\}, f(\bar{b}) = 1 \]
The K-Junta Algorithm

\[ y = f(\bar{x}), \bar{x} = \{x_1, x_2, \ldots, x_i, x_{i+1}, \ldots, x_j, \ldots, x_n\} \]

\[ \bar{a} = \{0,0,\ldots,0,0,0,0,\ldots,0,0\}, f(\bar{a}) = 0 \]

\[ \bar{a} = \{1,0,\ldots,0,0,0,0,\ldots,0,0\}, f(\bar{a}) = 0 \]

\[ \bar{a} = \{1,0,\ldots,1,0,0,0,0,\ldots,0,0\}, f(\bar{a}) = 0 \]

\[ \bar{b} = \{1,0,\ldots,1,0,1,0,0,\ldots,0,1\}, f(\bar{b}) = 1 \]
The K-Junta Algorithm

\[ y = f(\bar{x}), \quad \bar{x} = \{x_1, x_2, \ldots, x_i, x_{i+1} \ldots, x_j, \ldots, x_n\} \]

\[ \bar{a} = \{0, 0, 0, 0, 0, 0, 0, \ldots, 0, 0\}, \quad f(\bar{a}) = 0 \]

\[ \bar{a} = \{1, 0, 0, 0, 0, 0, 0, \ldots, 0, 0\}, \quad f(\bar{a}) = 0 \]

\[ \bar{a} = \{1, 0, 1, 0, 0, 0, 0, \ldots, 0, 0\}, \quad f(\bar{a}) = 0 \]

\[ \bar{a} = \{1, 0, 1, 0, 0, 0, 0, \ldots, 0, 0\}, \quad f(\bar{a}) = 1 \]

\[ \bar{b} = \{1, 0, 1, 0, 0, 0, \ldots, 0, 1\}, \quad f(\bar{b}) = 1 \]
The K-Junta Algorithm

\[ y = f(\bar{x}), \quad \bar{x} = \{ x_1, x_2, \ldots, x_i, x_{i+1}, \ldots, x_j, \ldots, x_n \} \]

\[
\bar{a} = \{0, 0, \ldots, 0, 0, 0, 0, 0, \ldots, 0, 0\}, \quad f(\bar{a}) = 0
\]

\[
\bar{a} = \{1, 0, \ldots, 0, 0, 0, 0, 0, \ldots, 0, 0\}, \quad f(\bar{a}) = 0
\]

\[
\bar{a} = \{1, 0, \ldots, 1, 0, 0, 0, 0, \ldots, 0, 0\}, \quad f(\bar{a}) = 0
\]

\[
\bar{a} = \{1, 0, \ldots, 1, 0, 1, 0, 0, \ldots, 0, 0\}, \quad f(\bar{a}) = 1
\]

\[
\bar{b} = \{1, 0, \ldots, 1, 0, 1, 0, 0, \ldots, 0, 1\}, \quad f(\bar{b}) = 1
\]

\[ O(n \cdot \log n \cdot k \cdot 2^k) \]
Partial Dependency Graph

If $k$ is too high \(\Rightarrow\) Partial dependency graph

Influence = sensitivity of a function to a variable

K-Junta works for Influence $>\frac{1}{2^k}$
Outline

• IP theft issue
• Reverse Engineering with Scan
• Junta Learning
• Clustering and Graph Completion
• The Test Case: BitCoin SHA-256
• Conclusions
The Adder Example

- Dependencies across many bits are not likely to appear
  - Influence too low
- Close neighbor dependencies are discovered
- Need to group all the nodes of the adder
SNN Clustering

- Shared Nearest Neighbors Clustering
  - Every pair of nodes with >threshold shared dependencies assigned to the same cluster
SNN Clustering

- Shared Nearest Neighbors Clustering
  - Every pair of nodes with \( >\text{threshold} \) shared dependencies assigned to the same cluster
Enumeration of the Adder Nodes

- Sort outputs in a cluster by their fan-in
  - Sort inputs accordingly
- Handle the plateau by iterative enumeration
  - Higher order inputs feed higher order outputs
Completing the graph

• Assuming the learner is looking for an adder
• Add dependencies of output bit \( i \) on all input bits 0 to \( i \).
Outline

- IP theft issue
- Reverse Engineering with Scan
- Junta Learning
- Clustering and Graph Completion
- The Test Case: BitCoin SHA-256
- Conclusions
SHA-256 Structure

Message Schedule

- Pre-processing
  - 512-bit chunk

IV

Selector

Compression stage 1

Compression stage 2

Compression stage 64

Hash

A, B, C, D, E, F, G, H

from extension $w_i$

$K_i$ constant

$A_{i+1}, B_{i+1}, C_{i+1}, D_{i+1}, E_{i+1}, F_{i+1}, G_{i+1}, H_{i+1}$

from extension $w_{i+1}$

$K_{i+1}$ constant

Maj

Σ

+
Learning Strategy

• The implementation is not known in advance
• But there are building blocks inherent to SHA-256
  – 7-way adder
  – 5-way adder
• We search for structures that look like adders
BitCoin SHA-256 Accelerator

- Open source design from opencores.org
- Performance oriented, heavily pipelined
- ~80,000 registers
- Used a software simulator
After K-Junta and Clustering

64-sized clusters match
2 32-bit adders
→ Compression Stage

32-sized clusters match
1 32-bit adder
→ Message Schedule

SNN Clustering Error

Number of stages suggests two SHA-256 instances, but not necessarily
Zooming in into a cluster

• Sort by enumeration

• How to detect individual operands?
Detecting operands by fanout

- Fanout components
  - Bit order
  - Number of functions
  - Function type
Returning to sequential

Flip-flop Outputs

Flip-flop Inputs

Flattened

Folded
Summary

• A novel method of IP theft detection
  – By non-invasive reverse engineering with scan
  – Boolean function analysis and graph methods
  – Works with or without watermarks

• Learned a 80,000-register SHA-256 accelerator

• What next
  – More test cases
  – Detecting Trojan hardware
Thanks!