Synthesis, Verification & Test for Secure ICs

Protecting Integrated Circuits from Piracy with Test‐aware Logic Locking

Stephen Plaza
Igor Markov

The Piracy Threat

- Problem 1: Unauthorized over-production of IC
  - $$ Lost revenue opportunities
  - Undesired exposure of proprietary technology

- Problem 2: Substandard production
  - Insertion of harmful Trojans
  - Sabotage and espionage opportunities

Driving factor
Worldwide distribution of IC production (i.e., outsourcing)

An “EPIC” Solution?

- Basic idea: hide functionality from manufacturer
- Split manufacturing, test, etc between different companies → infeasible
- EPIC active metering approach [Roy ‘08, ’10 revision]
  - Springboard for many subsequent research efforts
  - Unique chip ID (e.g., PUF)
  - Locking mechanism to prevent correct circuit operation
Our Solution

*Mux-based locking that preserves test response* → manufacturer does not activate the circuit

**Advantages**
1. Manufacturer never has a functional circuit
2. Supplied test response cannot reveal locking scheme
3. End-user can validate authenticity

Outline

• Background
  - XOR locking (and variants)
  - Types of attacks
- XOR locking vulnerability to attacks
- MUX-based locking
  - Background on simulation-based synthesis
  - Mux locking strategy
  - Practical considerations
- Results

Circuit Locking

- Add logic to the circuit to alter circuit output
- Key bits that can disable locks

Adding XOR-based Locks

**Properties**

- XORs placed randomly throughout circuit
- Key bits not obvious even with known output response

*key bit not obvious upon inspection*
Types of Attacks

• Mask modification to disable PUF

• Mask modification to disable locks
  • Hard to find on inspection
  • Hard to modify in current technologies

• “Guess” key bit values
  • Random key patterns intractable
  • Simulation based on stolen netlist [Rajendran ’12] (better placement of XORs make this attack difficult)

Supplied test patterns and response indicate expected behavior → potential attack vulnerability

Outline

• Background
  • XOR locking (and variants)
  • Types of attacks

• XOR locking vulnerability to attacks

• MUX-based locking
  • Background on simulation-based synthesis
  • Mux locking strategy
  • Practical considerations

• Results

XOR Lock Attack

**High-level Strategy**
Assume access to key bits

• Inputs: test pattern input, test pattern output, locked circuit
• Output: key bit assignment that unlocks circuit

**Algorithm**
1. Choose a key bit at random
2. Apply all test patterns, observe test response
3. Flip key bit value
4. Apply all test patterns, observe test response
5. Choose the key bit value that minimizes test response differences
6. Repeat till convergence

![Diagram of XOR Lock Attack](image-url)
XOR Lock Attack Considerations

- Output response often betrays gradient toward unlocked configuration
- Random restarts needed since algorithm gets caught in local minimum
- Adding XOR locks with downstream correlation can make a naïve attack trickier

Outline

- Background
  - XOR locking (and variants)
  - Types of attacks
- XOR locking vulnerability to attacks
  - MUX-based locking
    - Background on simulation-based synthesis
    - Mux locking strategy
    - Practical considerations
- Results

Efficient Resynthesis through Simulation-based Approximations

- Simulation can be used to approximate circuit behavior
  - Estimate average case behavior
  - Linear-time computation
- Apply values, e.g., random, to primary inputs
- Associate signatures with each node
- Use signatures to enable complex optimizations

Signatures and Bit Simulation

- Signature: partial truth table associated with each node in a circuit
- Stimulate inputs with simulation vectors

\[ \text{Sig}(x_1) = \{011\} \]
Signature-based Synthesis Optimization

- Use signatures to guide synthesis optimization
- *e.g.*, find signatures that are equal and merge nodes
  - Need to prove equivalence (*e.g.*, by using SAT)

![Circuit Diagram](image)

Signature-driven Locking Strategy

- Node Merging with Signatures
- Finding Locks with Signatures
  - Apply Test Patterns
  - Extract Signatures
  - Find Equal Signature
  - Verify Formally
  - Optimize
  - Add Lock

Finding Candidate Signatures

- Problem: few signals with equal signatures that are not logically equivalent
- Solution: create equal signatures using logic covers

Logic Cover

- $\text{Sig}_1 \leq \text{Sig}_2 \Rightarrow \text{Sig}_1 = \text{Sig}_1 \& \text{Sig}_2$
- $\text{Sig}_1 \leq \text{Sig}_2 \Rightarrow \text{Sig}_2 = \text{Sig}_1 \& \text{Sig}_2$

MUX-based Locking Algorithm

1. Apply Test Patterns
2. Create Candidate Cover for Randomly Chosen Node (most have covers)
### MUX-based Locking Algorithm

**3. Apply Random Simulation**

**5. Add MUX lock**

**4. Disprove Candidate Cover**

### Practical Considerations of MUX Locking to the Design Flow

**Area considerations**
- Each optimization requires minimal additional logic
- Only a few locks (e.g., 64, 128) are needed for the key to be uncrackable

**Timing and wiring**
- We avoid operations that increase logic levels
- More generally: use local signals when synthesizing locks to uphold timing/wiring constraints → need many candidate locking sites

### Handling Untestable Logic with Probabilistic Testing

**Problem 1:** incorrect lock key will make some logic untestable

**Problem 2:** unpredictable output response could complicate diagnosing

**Solution:** run test patterns multiple times with different key bit values

**Properties**
- If there are no interactions between locked sites, exponential fewer untested sites for each combination
- Improved testability possible because covers can increase observability of internal signals
## Yield-aware Testing

**Problem:** test inputs change as a function of yield

**Solution:** maintain large pool of test patterns compatible with locking

- Derive/Insert mux locks
- Derive test vectors
- Test Patterns
- Choose subset of test patterns
- Design Under Test
- Current Yield

## Outline

- Background
  - XOR locking (and variants)
  - Types of attacks
- XOR locking vulnerability to attacks
- MUX-based locking
  - Background on simulation-based synthesis
  - Mux locking strategy
  - Practical considerations
- Results

## Experimental Setup

- Benchmark info
  - ISCAS89 and IWLS ’05 benchmarks
  - Combinational portions considered (latches treated as PIs and POs)
- Circuits structurally hashed by ABC
- All analysis at logic level (no place or route)

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#Gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>c880</td>
<td>383</td>
</tr>
<tr>
<td>usb_phy</td>
<td>1197</td>
</tr>
<tr>
<td>sasc</td>
<td>1651</td>
</tr>
<tr>
<td>c3540</td>
<td>1669</td>
</tr>
<tr>
<td>i2c</td>
<td>2902</td>
</tr>
<tr>
<td>pci_spoci_ctrl</td>
<td>3483</td>
</tr>
<tr>
<td>systemcdes</td>
<td>9008</td>
</tr>
<tr>
<td>spi</td>
<td>10109</td>
</tr>
<tr>
<td>tv80</td>
<td>22575</td>
</tr>
<tr>
<td>systemcaes</td>
<td>26717</td>
</tr>
</tbody>
</table>

## Cracking XOR Locks

- 4 versions of each circuit with different locking configurations
- Timeout: 1,000,000 combinations

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#Test patterns</th>
<th># Key Comb (64 bit)</th>
<th>% circuits cracked</th>
<th>#Key Comb (128 bit)</th>
<th>% circuits cracked</th>
</tr>
</thead>
<tbody>
<tr>
<td>c880</td>
<td>53</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>usb_phy</td>
<td>67</td>
<td>37709</td>
<td>75</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sasc</td>
<td>67</td>
<td>1334</td>
<td>100</td>
<td>174517</td>
<td>25</td>
</tr>
<tr>
<td>c3540</td>
<td>149</td>
<td>22624</td>
<td>100</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>i2c</td>
<td>164</td>
<td>11722</td>
<td>100</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>pci_spoci_ctrl</td>
<td>246</td>
<td>13196</td>
<td>100</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>systemcdes</td>
<td>123</td>
<td>4767</td>
<td>100</td>
<td>55005</td>
<td>50</td>
</tr>
<tr>
<td>spi</td>
<td>554</td>
<td>2290</td>
<td>100</td>
<td>131112</td>
<td>75</td>
</tr>
<tr>
<td>tv89</td>
<td>878</td>
<td>1727</td>
<td>75</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>systemcaes</td>
<td>426</td>
<td>2247</td>
<td>100</td>
<td>489</td>
<td>50</td>
</tr>
</tbody>
</table>

Smaller designs have more interactions between locks \(\to\) harder to crack

Larger circuits like spi easily cracked even with 128 keys
Adding MUX Locks

- Inserts 64 locks for each circuit

<table>
<thead>
<tr>
<th>circuit</th>
<th>% area overhead</th>
<th>% signals with cover candidates</th>
<th>lock insertion runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c880</td>
<td>33.4</td>
<td>67.4</td>
<td>1.8</td>
</tr>
<tr>
<td>usb_phy</td>
<td>10.7</td>
<td>54.0</td>
<td>1.0</td>
</tr>
<tr>
<td>sasc</td>
<td>7.8</td>
<td>57.4</td>
<td>0.2</td>
</tr>
<tr>
<td>c3540</td>
<td>7.6</td>
<td>58.4</td>
<td>116.9</td>
</tr>
<tr>
<td>i2c</td>
<td>4.4</td>
<td>53.0</td>
<td>4.7</td>
</tr>
<tr>
<td>pci_spoci_ctrl</td>
<td>3.7</td>
<td>73.4</td>
<td>12.3</td>
</tr>
<tr>
<td>systemcdes</td>
<td>1.4</td>
<td>67.0</td>
<td>6.9</td>
</tr>
<tr>
<td>spi</td>
<td>1.2</td>
<td>82.6</td>
<td>499.2</td>
</tr>
<tr>
<td>tv89</td>
<td>0.6</td>
<td>85.5</td>
<td>506.1</td>
</tr>
<tr>
<td>systemcaes</td>
<td>0.5</td>
<td>34.3</td>
<td>32.8</td>
</tr>
</tbody>
</table>

Small area overhead, fixed cost independent of circuit size

Most signals have cover candidates (insertion is effectively random)

Simulation dominates runtime: function of circuit size and observability of signals

Test Coverage

<table>
<thead>
<tr>
<th>circuit</th>
<th>% fault coverage (unlocked)</th>
<th>% fault coverage (locked)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c880</td>
<td>100</td>
<td>99.6</td>
</tr>
<tr>
<td>usb_phy</td>
<td>95.3</td>
<td>95.3</td>
</tr>
<tr>
<td>sasc</td>
<td>98.7</td>
<td></td>
</tr>
<tr>
<td>c3540</td>
<td>96.9</td>
<td></td>
</tr>
<tr>
<td>i2c</td>
<td>96.9</td>
<td>96.8</td>
</tr>
<tr>
<td>pci_spoci_ctrl</td>
<td>87.8</td>
<td>88.0</td>
</tr>
<tr>
<td>systemcdes</td>
<td>95.0</td>
<td>95.0</td>
</tr>
<tr>
<td>spi</td>
<td>91.1</td>
<td>91.2</td>
</tr>
<tr>
<td>tv89</td>
<td>90.8</td>
<td>90.8</td>
</tr>
<tr>
<td>systemcaes</td>
<td>91.9</td>
<td>91.9</td>
</tr>
</tbody>
</table>

Only one random assignment of key bits still leads to high test coverage

It is possible that testing coverage is higher after adding MUX locks

Conclusions

- Demonstrated vulnerabilities to random XOR locking simply using test patterns and simulation
- Introduced a test-aware circuit locking strategy that removes the manufacturer from the chip activation loop (customer and rights owner can directly interact)
- Demonstrated the scalable identification and injection of MUX locks with minimal impact on area and timing
- Showed negligible impact to circuit testability and diagnosability

Questions?