Reversible Denial-of-Service by Locking Gates Insertion for IP Cores Design Protection

Brice Colombier*, Lilian Bossuet*, David Hély+

*Laboratoire Hubert Curien
Université Jean Monnet
Saint-Étienne — France

+LCIS
Grenoble Institute of Technology
Valence — France
Design-and-reuse paradigm

**Problem**
The designer cannot control **how many times** the IP is instantiated.
- Overusing,
- Illegal cloning.

**One solution**
Make the IP unusable unless it has been previously **activated**.
⇒ Illegal copies are useless.
Protection scheme architecture

Each has its role:

- **Security**: relies on a **cryptographic primitive**.
- **Uniqueness**: ID, NVM, PUF...
- **Disabling**: specific masking/locking module.

### Locking
- FSM,
- Clock,
- Logic.
In 2008, Roy et al.\textsuperscript{1} proposed to \textit{randomly} insert XOR/XNOR gates in the netlist.

\textsuperscript{1}Roy, Koushanfar, Markov EPIC: Ending Piracy of Integrated Circuits Design, Automation and Test in Europe, 2008
Logic masking

In 2013, Rajendran et al.\textsuperscript{2} improved the **node selection** method.

**Fault analysis-based node selection**

- Requires a fault simulator,
- Computationally expensive.

\textsuperscript{2} Rajendran, Zhang, Rose, Pino, Sinanoglu, Karri *Fault analysis-based logic encryption* IEEE Transactions on Computers, 2013

\textsuperscript{3} Plaza, Markov *Protecting Integrated Circuits from Piracy with Test-aware Logic Locking* International Conference on Computer Aided Design, 2014
Logic masking

In 2013, Rajendran et al.\(^2\) improved the **node selection** method.

### Fault analysis-based node selection
- Requires a fault simulator,
- Computationally expensive.

### Security flaw
There is a **gradient** towards the correct key.

A **hill climbing** attack\(^3\) can be mounted:
Choose a random key, flip bits **one after the other** to gradually reduce \(HD(output, test vectors)\) to 0.
Link between key bits and masked outputs is **too obvious**.

---


What is logic locking?

![Logic Locking Diagram](image-url)
What is logic locking?

Principle
- Propagating a locking value through a sequence of netlist's nodes.
- Forcing an internal node in the netlist locks a primary output.

Condition
- For all the nodes in the sequence (green nodes):
  - They are forced to a logic value that locks the following logic gate.
What is logic locking?

[Diagram of logic locking]

Principle
Propagating a locking value through a sequence of netlist's nodes.

Forcing an internal node in the netlist locks a primary output.

Condition
For all the nodes in the sequence (green nodes):
- They are forced to a logic value that locks the following logic gate.
What is logic locking?

Principle

Propagating a locking value through a sequence of netlist’s nodes.

Forcing an internal node in the netlist locks a primary output.

Condition

For all the nodes in the sequence (green nodes):
They are forced to a logic value that locks the following logic gate.
What is logic locking?

**Principle**
Propagating a **locking value** through a sequence of netlist’s nodes. Forcing an internal node in the netlist locks a primary output.

**Condition**
For all the nodes in the sequence (green nodes): They are **forced** to a logic value that locks the following logic gate.
Which node should be forced?

1st step: Build a graph from the netlist.

Conversion

<table>
<thead>
<tr>
<th>Nodes</th>
<th>→</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Vectors</td>
</tr>
<tr>
<td>G1</td>
<td>Input</td>
</tr>
<tr>
<td>G2</td>
<td>G</td>
</tr>
<tr>
<td>G3</td>
<td>...</td>
</tr>
<tr>
<td>G4</td>
<td>NAND</td>
</tr>
<tr>
<td>G5</td>
<td>...</td>
</tr>
<tr>
<td>G6</td>
<td>...</td>
</tr>
<tr>
<td>G7</td>
<td>...</td>
</tr>
<tr>
<td>G8</td>
<td>...</td>
</tr>
<tr>
<td>G9</td>
<td>...</td>
</tr>
<tr>
<td>G10</td>
<td>...</td>
</tr>
<tr>
<td>G11</td>
<td>...</td>
</tr>
<tr>
<td>G12</td>
<td>...</td>
</tr>
<tr>
<td>G13</td>
<td>...</td>
</tr>
<tr>
<td>G14</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gates</th>
<th>→</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>Output</td>
</tr>
<tr>
<td>G2</td>
<td>G</td>
</tr>
<tr>
<td>G3</td>
<td>...</td>
</tr>
<tr>
<td>G4</td>
<td>...</td>
</tr>
<tr>
<td>G5</td>
<td>...</td>
</tr>
<tr>
<td>G6</td>
<td>...</td>
</tr>
<tr>
<td>G7</td>
<td>...</td>
</tr>
<tr>
<td>G8</td>
<td>...</td>
</tr>
<tr>
<td>G9</td>
<td>...</td>
</tr>
<tr>
<td>G10</td>
<td>...</td>
</tr>
<tr>
<td>G11</td>
<td>...</td>
</tr>
<tr>
<td>G12</td>
<td>...</td>
</tr>
<tr>
<td>G13</td>
<td>...</td>
</tr>
<tr>
<td>G14</td>
<td>...</td>
</tr>
</tbody>
</table>
Graph labelling

2nd step: Label vertices with $V_{\text{forced}}$ and $V_{\text{locks}}$ values.

Labelling

- $V_{\text{forced}}$ depends on the preceding logic gate.
- $V_{\text{locks}}$ depends on the following logic gate.

<table>
<thead>
<tr>
<th>Node</th>
<th>$V_{\text{forced}}$</th>
<th>$V_{\text{locks}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>G2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G3</td>
<td>1</td>
<td>$\sim V_{\text{locks}}$(G4)</td>
</tr>
<tr>
<td>G4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G5</td>
<td>0</td>
<td>{0, 1}</td>
</tr>
<tr>
<td>G6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3rd step: Delete incoming edges of nodes for which $V_{\text{forced}} \neq V_{\text{locks}}$. Those nodes cannot propagate the locking value.
Graph cleaning

4th step: Delete connected components that contain no output.

Simplified graph

Cleaned graph
Nodes selection

In the disconnected final graph, which nodes should be locked?

One source vertex

Select the source vertex.
Nodes selection

In the disconnected final graph, which nodes should be locked?

One source vertex

Multiple source vertices

Select the source vertex.

Select the furthest node spanning all the outputs.
Nodes selection

In the disconnected final graph, which nodes should be locked?

One source vertex: Select the source vertex.

Multiple source vertices: Select the furthest node spanning all the outputs.

Multiple source vertices not all outputs spanned: Sort the nodes w.r.t to the number of outputs they span and select them greedily.
Locking gates insertion

So far, we have:
- list of nodes to lock,
- associated $V_{locks}$ values.

$V_{locks} = 0$ : add AND gate
$V_{locks} = 1$ : add OR gate
Area overhead

**Overhead metric:** percentage of locking gates to add.
Implemented on ITC’99 benchmarks (1k to 225k logic gates)

---

$\sim 2x$ lower overhead than logic masking$^2$ (+5.7%)

---

## Analysis time

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#logic gates</th>
<th>Fault analysis-based logic masking</th>
<th>Graph analysis-based logic locking</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>160</td>
<td>20min</td>
<td>0.03s</td>
</tr>
<tr>
<td>c7552</td>
<td>3512</td>
<td>4h30min</td>
<td>0.87s</td>
</tr>
<tr>
<td>b19_C</td>
<td>225k</td>
<td>X</td>
<td>1h15min</td>
</tr>
</tbody>
</table>

---

Security margin

In the final graph, all nodes are available for logic locking.

More nodes can be forced to increase locking strength. It can make hill-climbing attack and reverse-engineering harder. The designer tunes the resources overhead/locking strength ratio.
Conclusion

Key points:

- Logic locking is a **powerful** way to make the circuit **unusable**:
  - Very **low** overhead,
  - **Tunable** security margin.
- Graph analysis-based selection method:
  - **Computationally efficient**,
  - Simple **EDA integration**.
- Logic masking/locking alone is **not secure**,
- A **cryptographic primitive** is necessary for security.

Presented at ISVLSI 2015\(^4\).

All Python scripts are available on the SALWARE project webpage\(^5\).


\(^5\) http://www.univ-st-etienne.fr/salware/FOGP.html
Questions

Contact: b.colombier@univ-st-etienne.fr