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

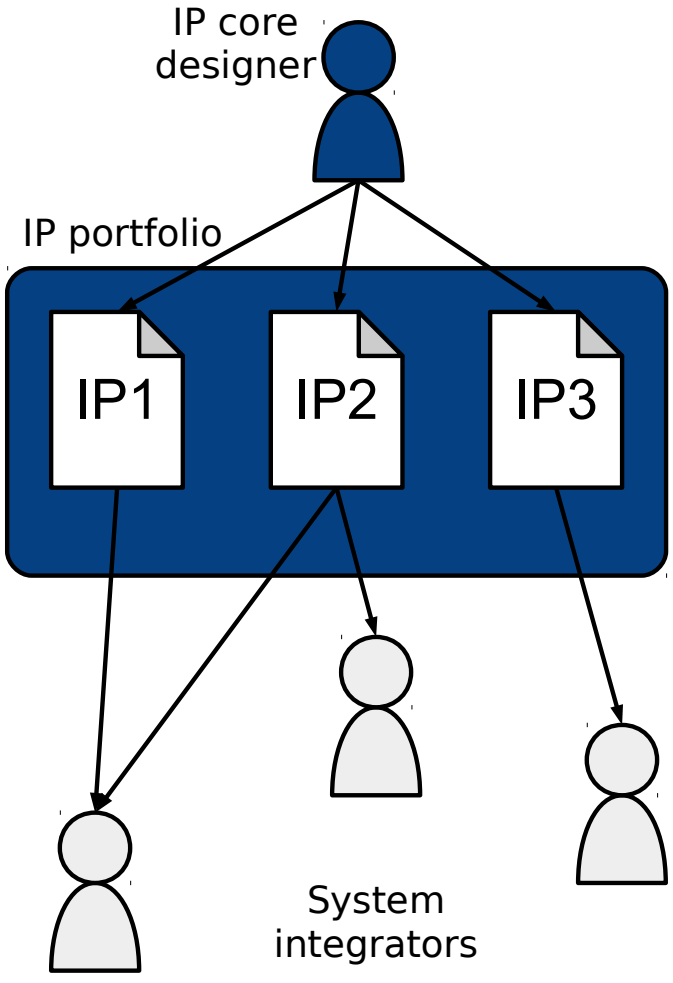
LABORATOIRE Hubert CURIEN
UMR • CNRS • 5516 • ST-ETIENNE

SALWARE
French ANR Project

RhôneAlpes Région

Outline

Design-and-reuse paradigm



Problem

The designer cannot control **how many times** the IP is used.

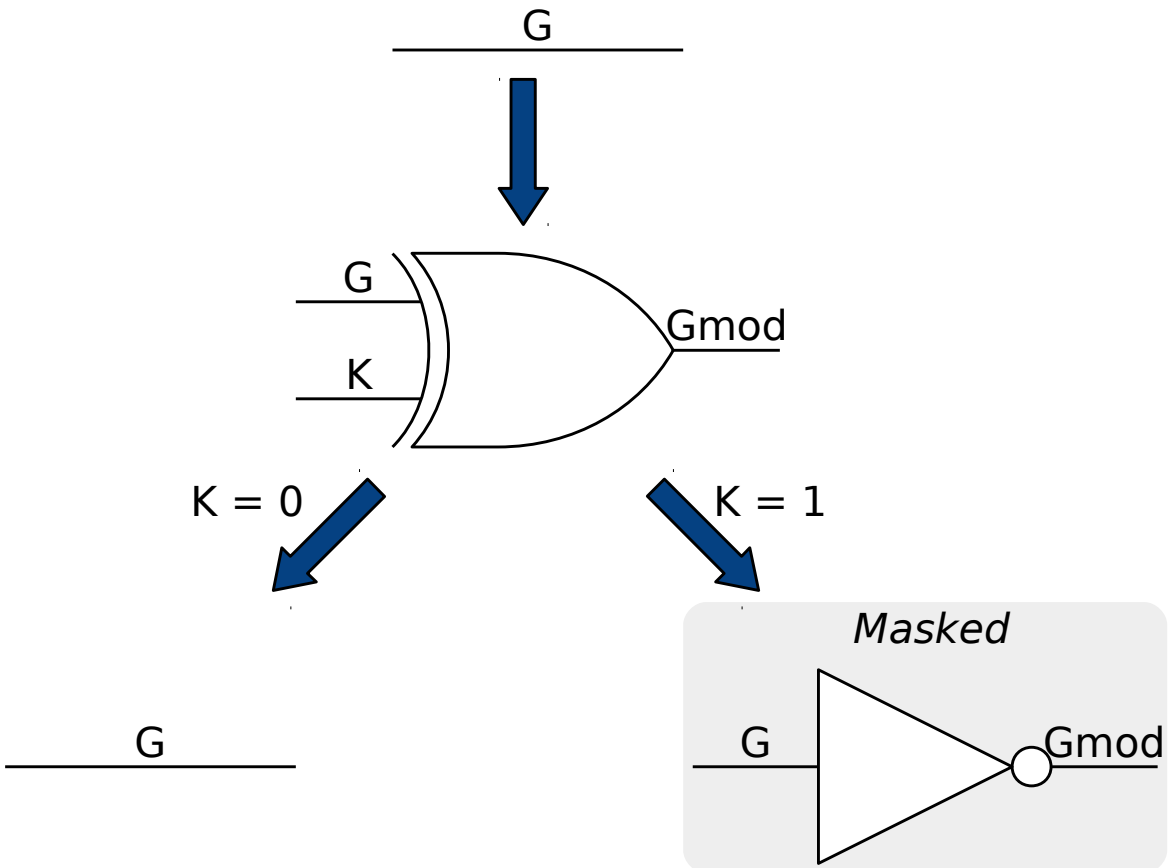
- Overusing,
- Illegal copying.

One solution

Make the IP unusable unless it has been previously **activated**.
$\Rightarrow$ Illegal copies are useless.

Logic masking

In 2008, Roy et al.[1] proposed to **randomly** insert XOR/XNOR gates in the netlist.



[1] **Roy, Koushanfar, Markov** EPIC: Ending Piracy of Integrated Circuits
Design, Automation and Test in Europe, 2008

Logic masking

In 2013, Rajendran et al.[2] improved the **node selection** method.

Fault analysis-based node selection

- Requires a fault simulator,
- Computationally expensive.

[2] **Rajendran, Zhang, Rose, Pino, Sinanoglu, Karri** *Fault analysis-based logic encryption* IEEE Transactions on Computers, 2013

[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.

---

[2] **Rajendran, Zhang, Rose, Pino, Sinanoglu, Karri** *Fault analysis-based logic encryption* IEEE Transactions on Computers, 2013

[3] **Plaza, Markov** *Protecting Integrated Circuits from Piracy with Test-aware Logic Locking* International Conference on Computer Aided Design, 2014

Protecting IP cores by inserting extra gates

Where are we?

- Insertion methods are **expensive**,
- Security level is **very low**.

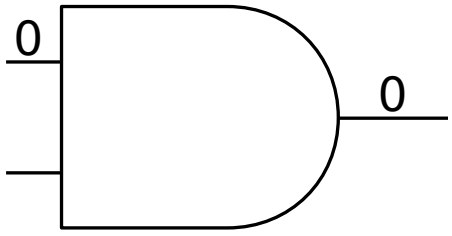
Each has its role:

Security : relies on a **cryptographic primitive**.

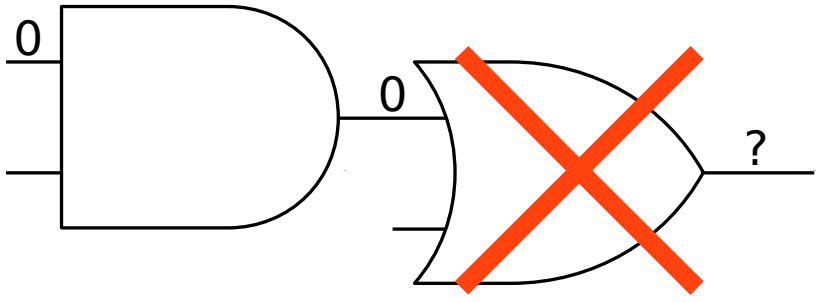
Disabling : specific masking/**locking** module.

Outline

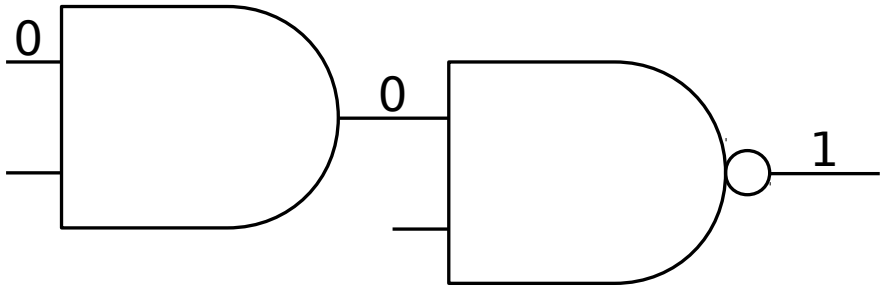
What is logic locking?



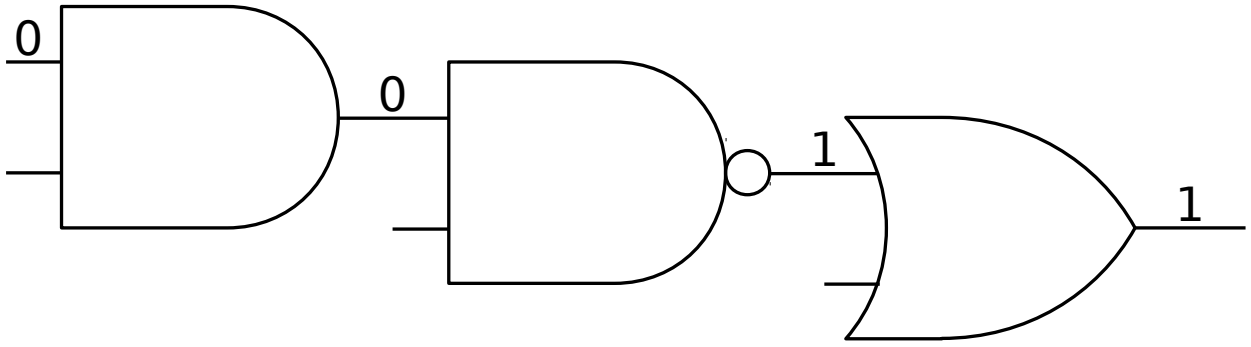
What is logic locking?



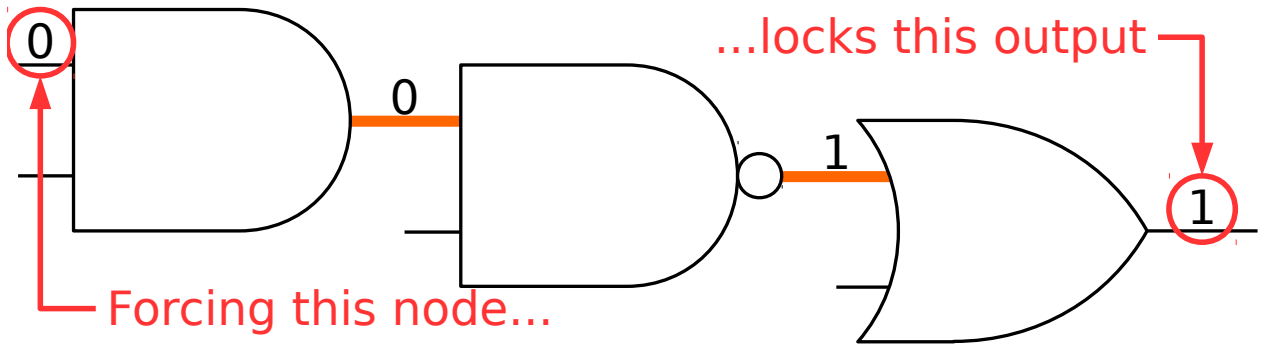
What is logic locking?



What is logic locking?



What is logic locking?



Principle

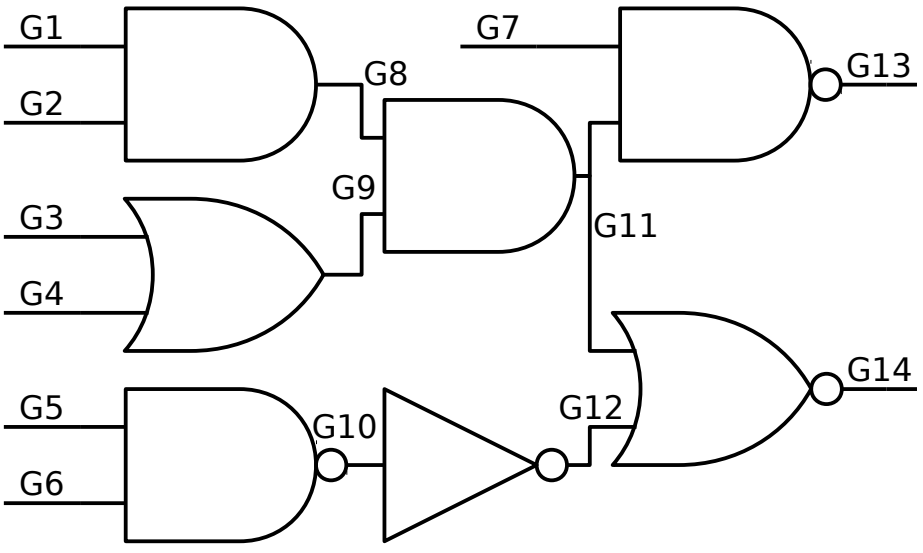
Propagating a **locking value** through a sequence of nodes.
Forcing an internal node **locks a primary output**.

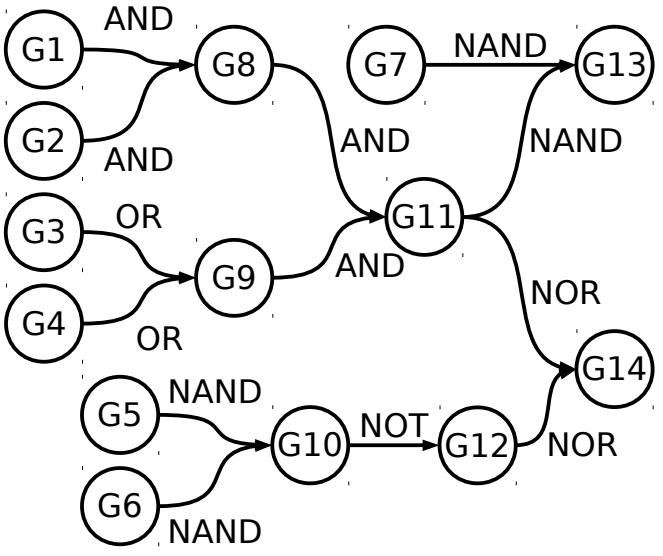
Condition

For all the nodes in the sequence (orange nodes): $V_{forced} \in V_{locks}$
(The nodes in the sequence 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

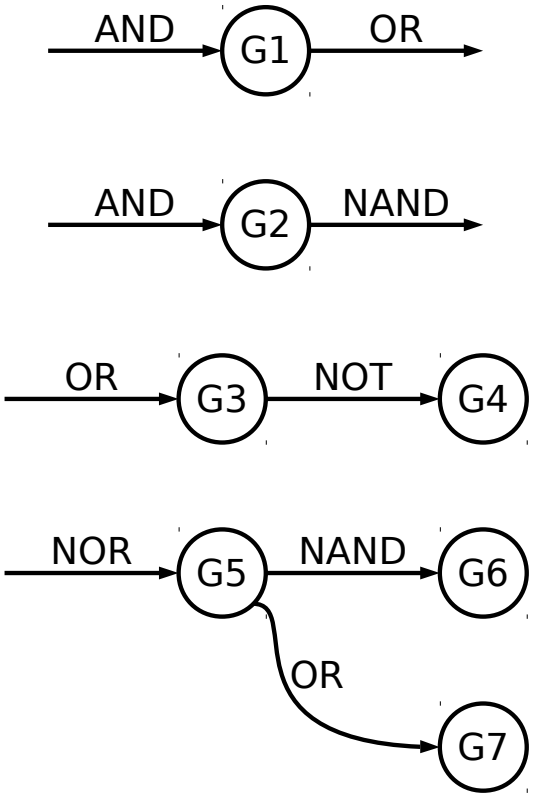
Nodes	$\rightarrow$	Edges
Gates	$\rightarrow$	Vertices

Graph labelling

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

Labelling

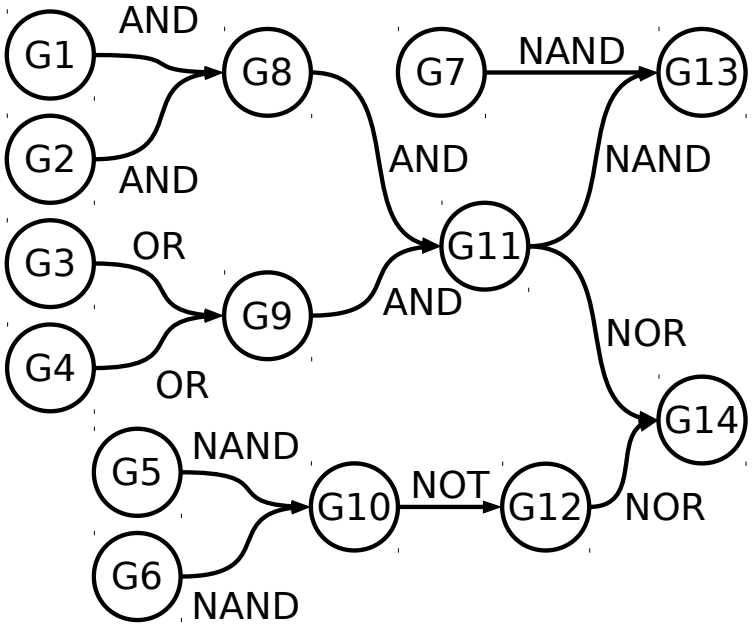
- V_{forced} depends on the **preceding** logic gate.
- V_{locks} depends on the **following** logic gate.



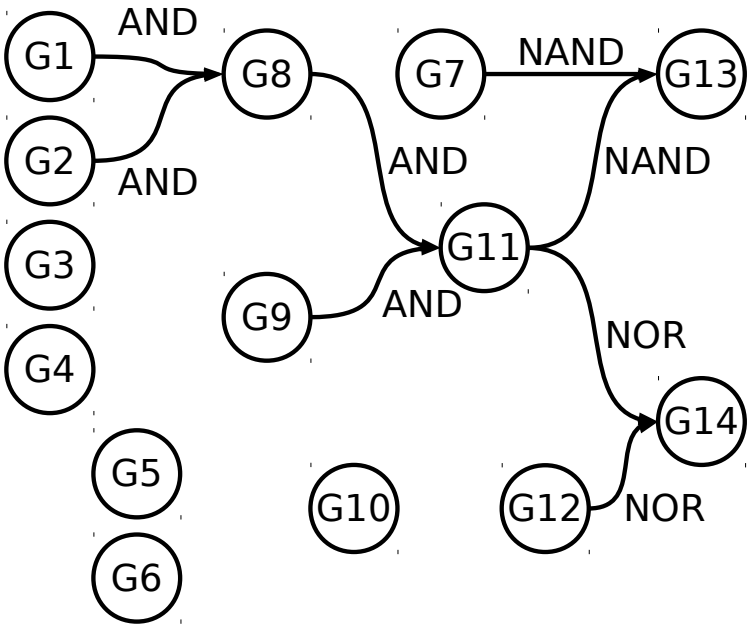
Node	V_{forced}	V_{locks}
G1	0	1
G2	0	0
G3	1	$\sim V_{locks}$(G4)
G5	0	{0, 1}

Graph simplification

3rd step: Delete **incoming edges** of nodes for which $V_{forced} \notin V_{locks}$. Those nodes **cannot propagate** the locking value.



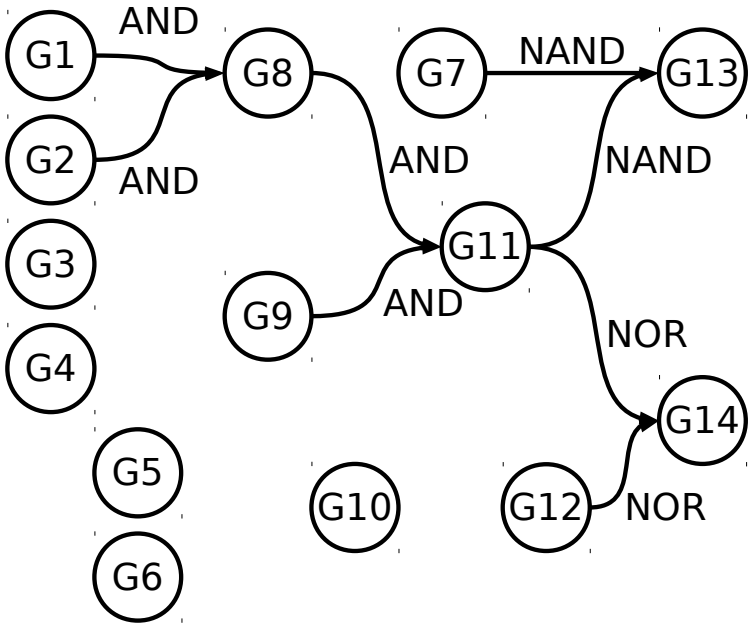
Original graph



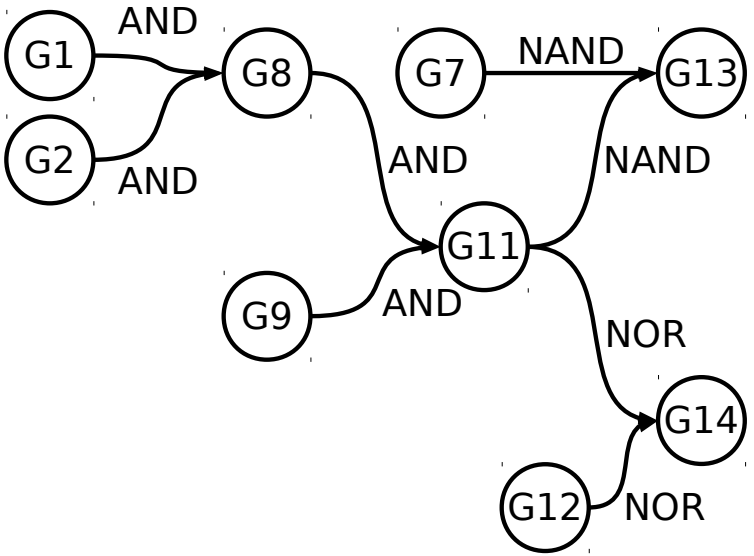
Simplified graph

Graph cleaning

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



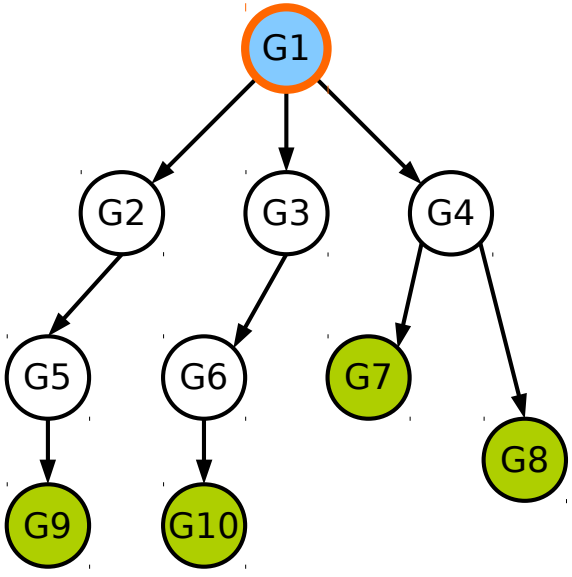
Simplified graph



Cleaned graph

Nodes selection

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

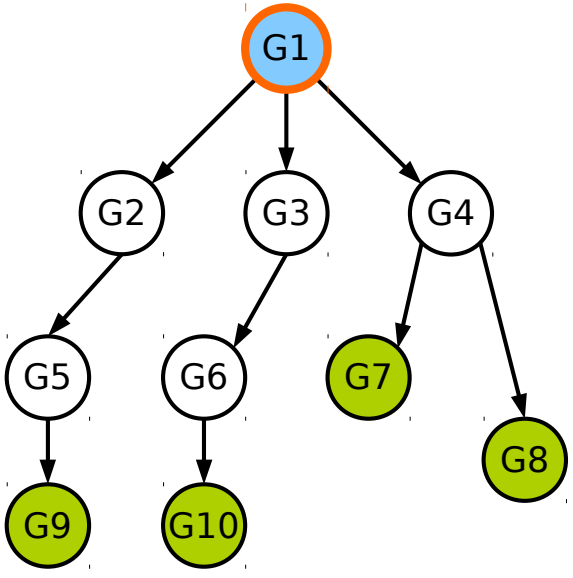


One source vertex

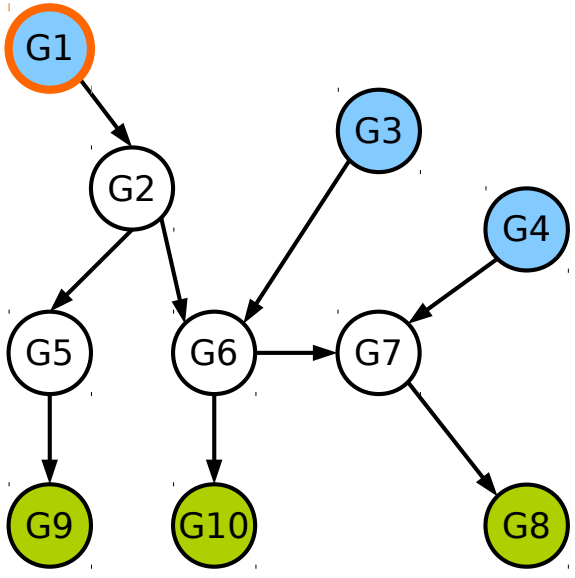
Select the source vertex.

Nodes selection

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



One source vertex



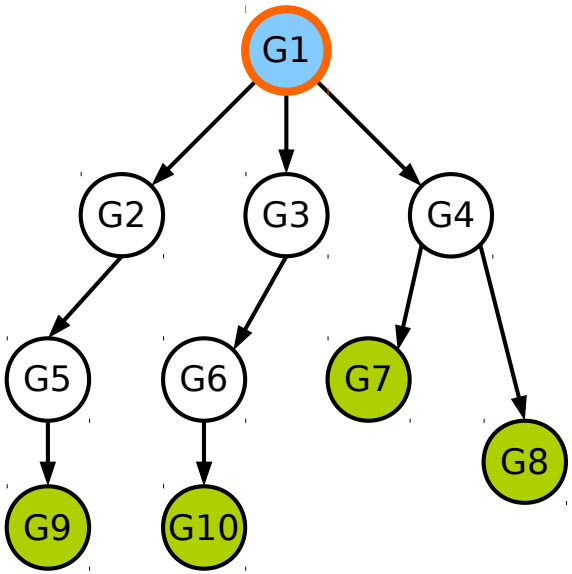
Multiple source vertices

Select the source vertex.

Select the furthest node spanning all the outputs.

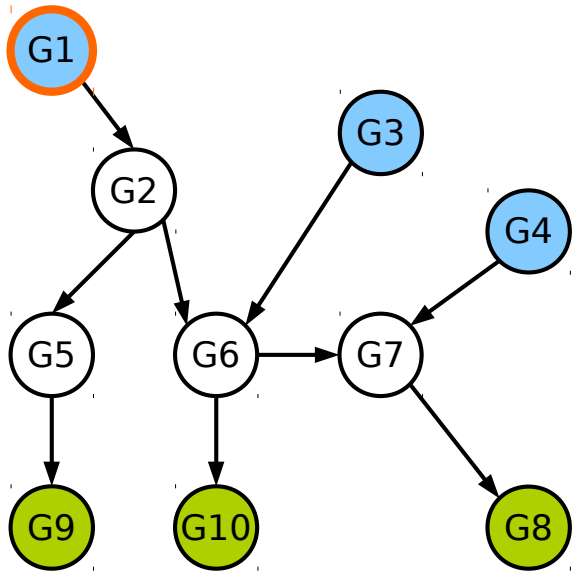
Nodes selection

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



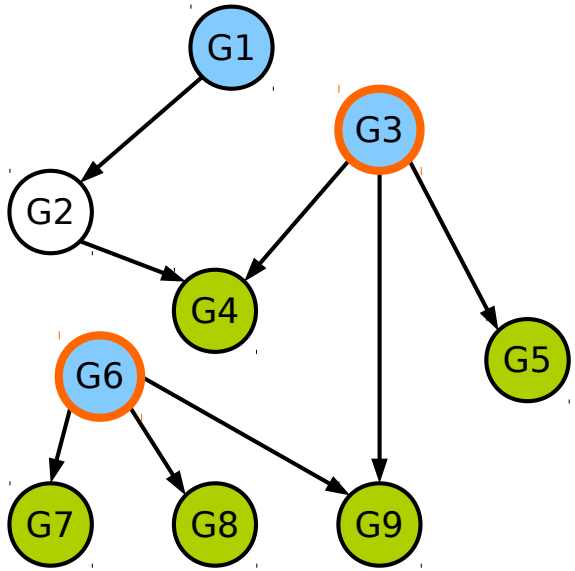
One source vertex

Select the source vertex.



Multiple source vertices

Select the furthest node spanning all the outputs.



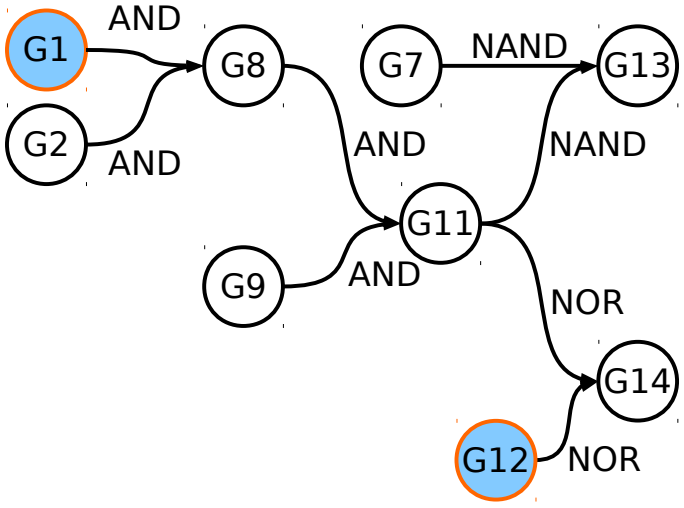
Multiple source vertices not all outputs spanned

Sort the nodes w.r.t to the number of outputs they span and select them greedily.

Locking gates insertion

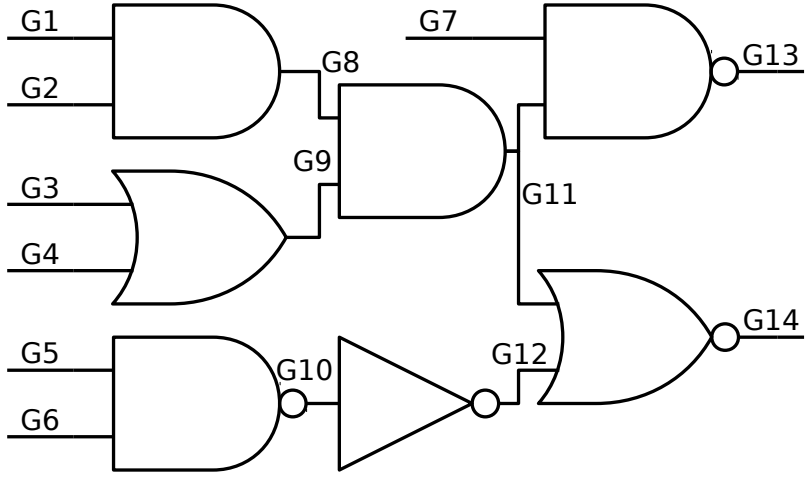
So far, we have:

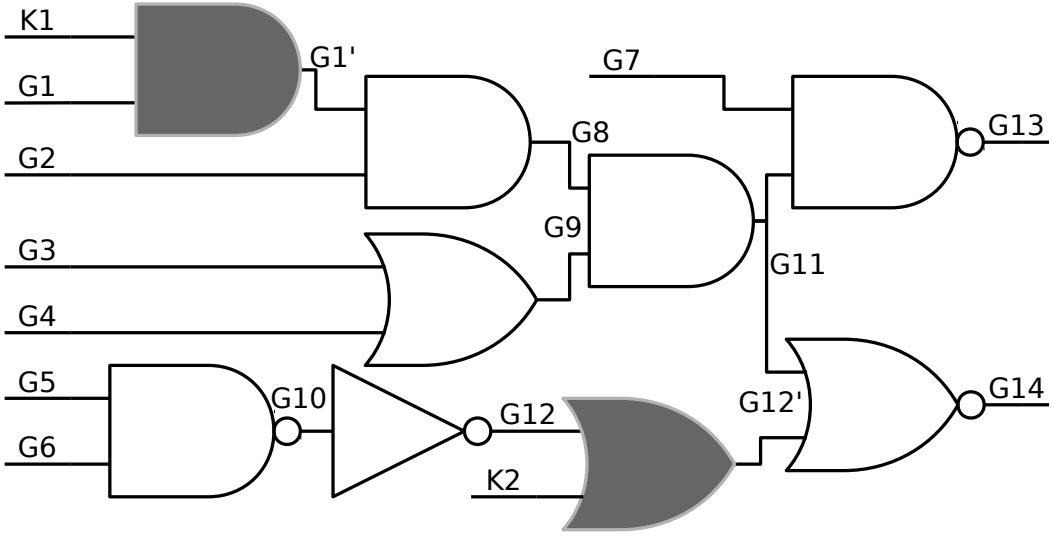
- list of nodes to lock,
- associated V_{locks} values.



$V_{locks} = 0$: add AND gate

$V_{locks} = 1$: add OR gate

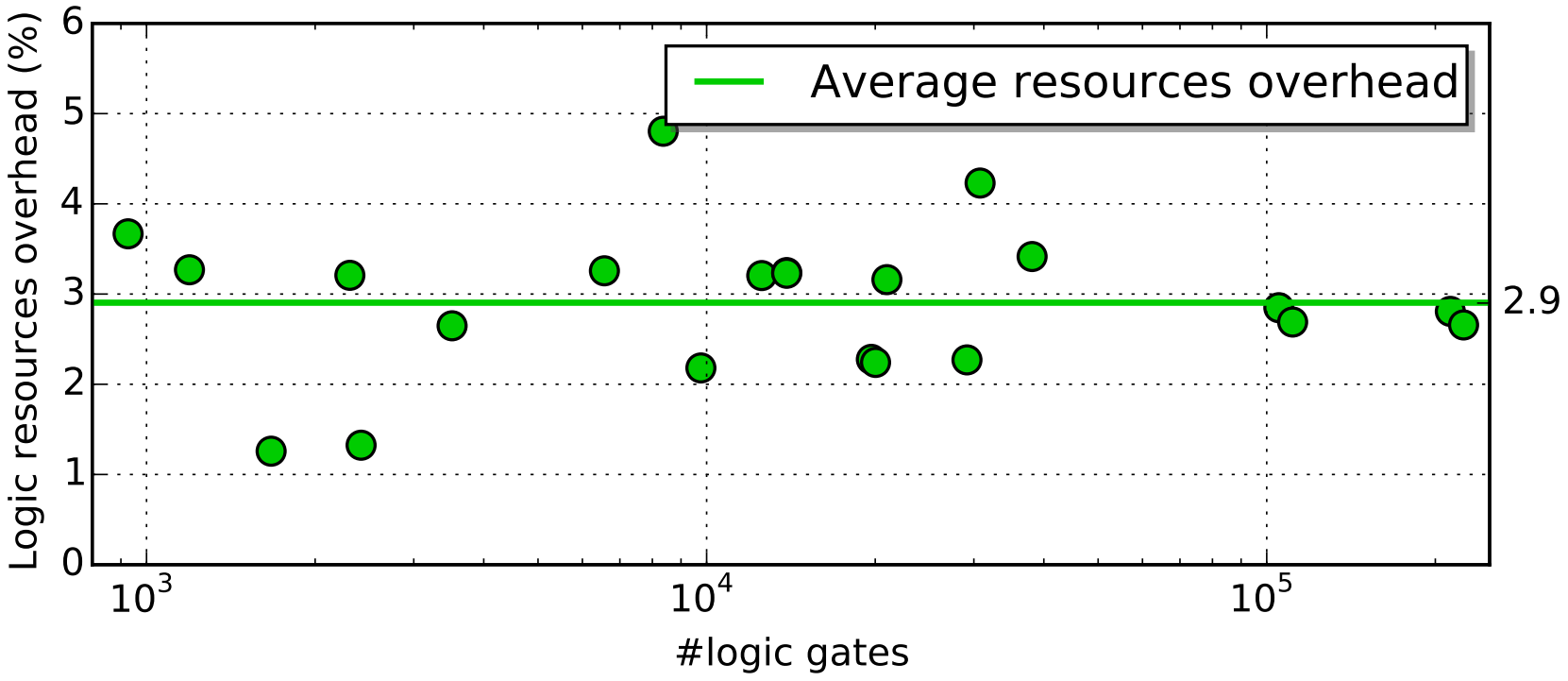




Outline

Area overhead

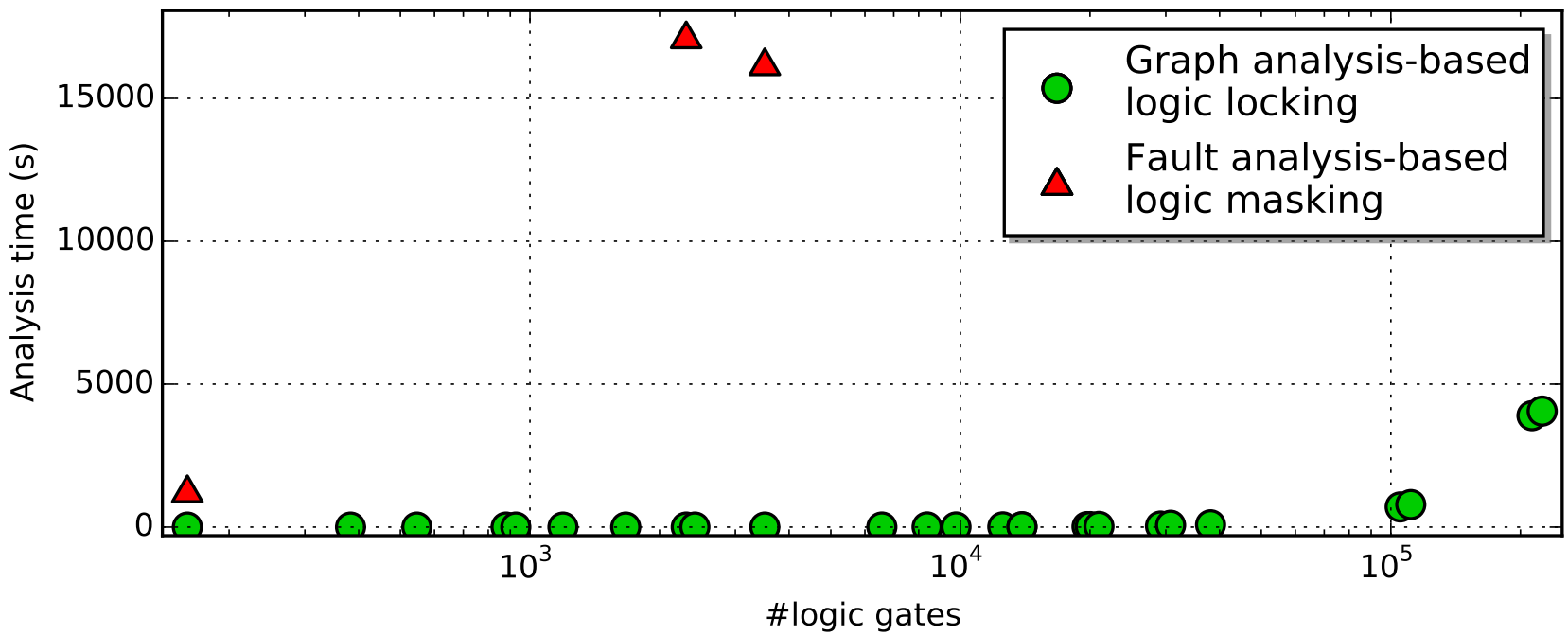
Overhead metric: percentage of locking gates to add.
Implemented on ITC'99 benchmarks (1k to 225k logic gates)



$\sim$ 2x smaller than logic masking (+5.7%)

Analysis time

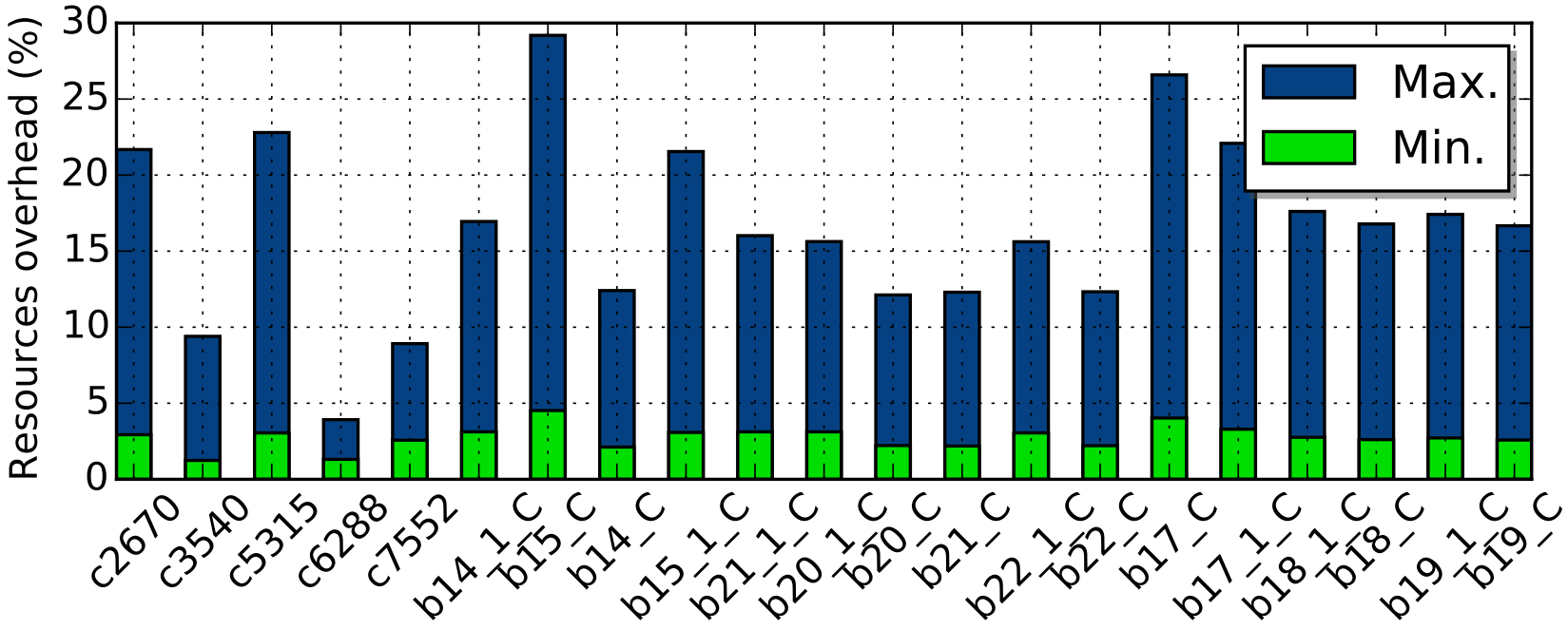
Benchmark	#logic gates	Fault analysis-based logic masking[2]	Graph analysis-based logic locking
c432	160	20min	0.03s
c7552	3512	4h30min	0.87s
b19_C	225k	X	1h15min



[2] **Rajendran, Zhang, Rose, Pino, Sinanoglu, Karri** *Fault analysis-based logic encryption* IEEE Transactions on Computers, 2013

Security margin

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



More nodes can be forced to increase locking strength.
The designer **tunes** the resources overhead/locking strength **ratio**.

Outline

Conclusion

- Graph analysis-based logic locking is a **powerful** way to make the circuit **unusable**,
- The induced resources overhead is **low**,
- **Computationally efficient** analysis makes **EDA integration** simple,
- Logic masking/locking alone is **not secure**,
- A **cryptographic primitive** is necessary for security.

All Python scripts are available on the SALWARE project webpage[4].

[4] http://www.univ-st-etienne.fr/salware/FOGP.html

Conclusion

- Graph analysis-based logic locking is a **powerful** way to make the circuit **unusable**,
- The induced resources overhead is **low**,
- **Computationally efficient** analysis makes **EDA integration** simple,
- Logic masking/locking alone is **not secure**,
- A **cryptographic primitive** is necessary for security.

Questions ?

All Python scripts are available on the SALWARE project webpage[4].

[4] http://www.univ-st-etienne.fr/salware/FOGP.html