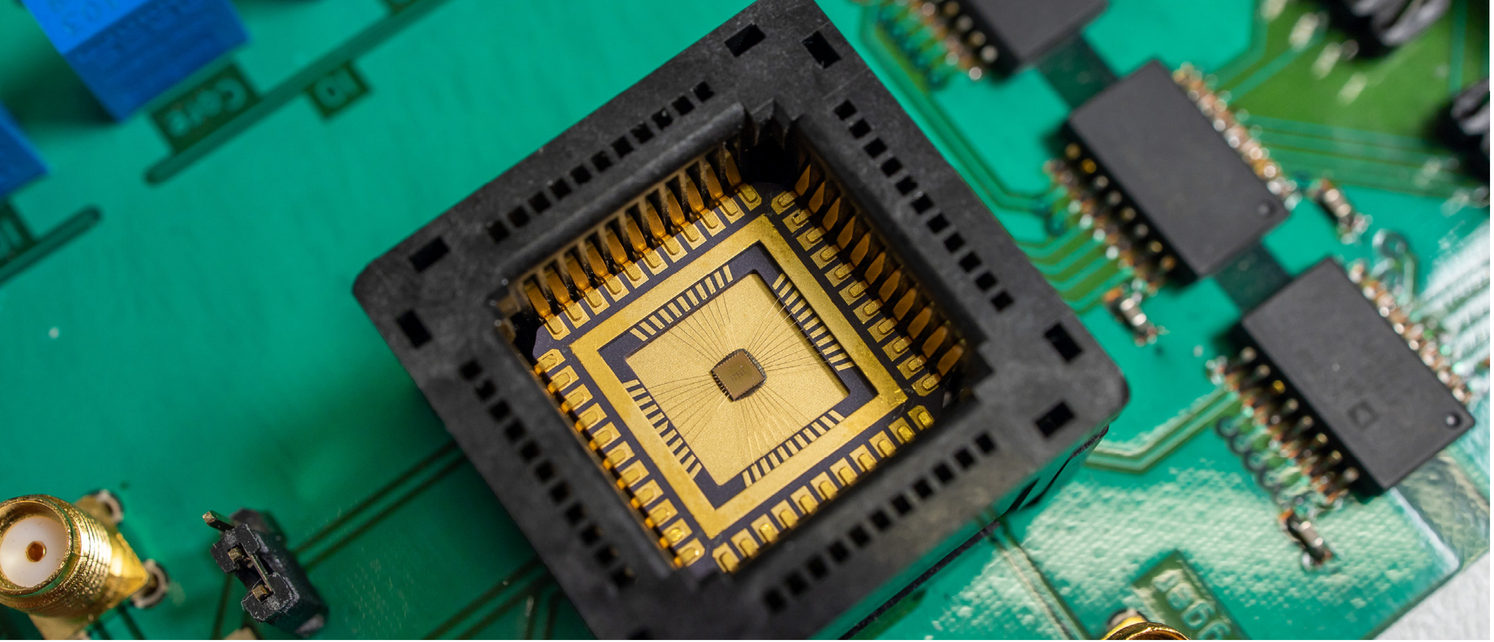


RUHR-UNIVERSITÄT BOCHUM

IMPLEMENTATION SECURITY

Automated Generation of Masked Hardware (AGEMA)

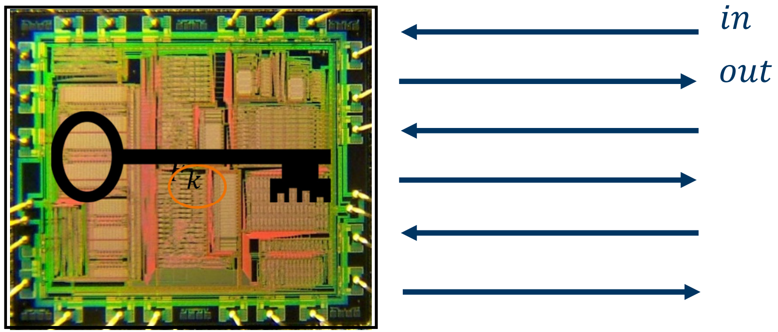
David Knichel, Amir Moradi, Nicolai Müller and Pascal Sasdrich

CHES, September 2022, Leuven, Belgium

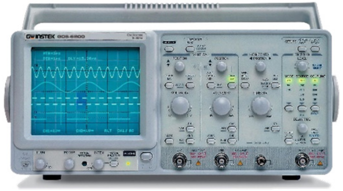
CASA
CYBER SECURITY IN THE AGE OF LARGE-SCALE ADVERSARIES

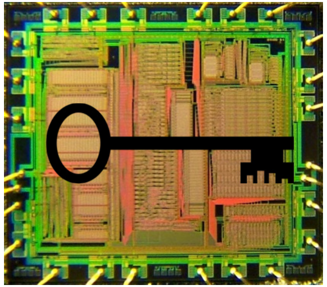
DFG

Crypto Device



Passive Physical Attacks





Physical characteristics can be exploited to extract secret information:

- Timing
- Power Consumption
- Electromagnetic radiations
- ...

Masking

Masking randomizes the intermediate values of a cryptographic computation to avoid dependencies between these values and the power consumption

It is usually applied on an algorithmic level

- Does not rely on the power consumption characteristics of the device

Each intermediate value is concealed by a random mask that is different for every execution

Basically, it corresponds to a secret sharing scheme:

- Boolean secret sharing

Boolean Secret Sharing

First order Boolean secret sharing (two shares):

- Secret: x
- Random: m

→ Shares: (x_1, x_2) $x_1 = x \oplus m$, $x_2 = m$ $x_1 \oplus x_2 = x$

- One needs to know share x_1 and x_2 to compute secret x
  - Neither of them alone provides enough information

Linear Function F

- Definition $F(x \oplus z) = F(x) \oplus F(z)$
- Boolean share before F: (x_1, x_2) with $x_1 \oplus x_2 = x$
- Boolean share after F: $\left(F(x_1), F(x_2)\right)$ with $F(x_1) \oplus F(x_2) = F\,(x_1 \oplus x_2) = F(x)$

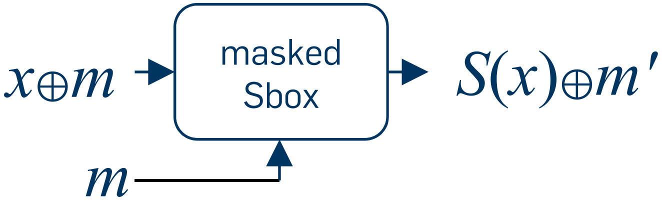
Non-linear Function?

Masking in Hardware

Pre-computing the masked tables in software

- Sequential operations, time consuming, low efficiency
- High efficiency is desired in hardware

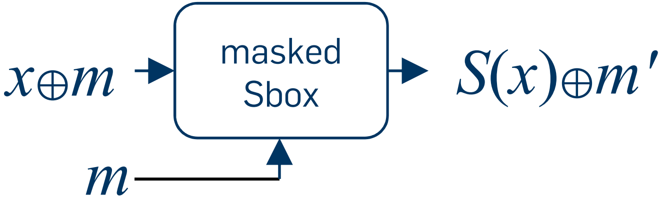
Ad-hoc/heuristic schemes



Masking in Hardware

Pre-computing the masked tables in software

- Sequential operations, time consuming, low efficiency
- High efficiency is desired in hardware



Ad-hoc/heuristic schemes

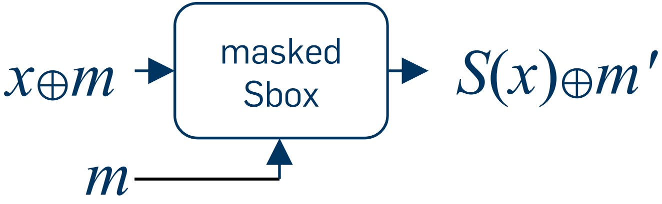
Processing the mask (m) and masked data ($x \oplus m$) simultaneously

- Joint distribution of leakages
  - It is called to be due to **glitches** [actually not always true]
  - Possible attacks

Masking in Hardware

Pre-computing the masked tables in software

- Sequential operations, time consuming, low efficiency
- High efficiency is desired in hardware



Ad-hoc/heuristic schemes

Processing the mask (m) and masked data ($x \oplus m$) simultaneously

- Joint distribution of leakages
  - It is called to be due to **glitches** [actually not always true]
  - Possible attacks

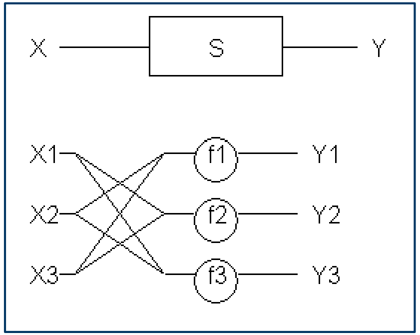
Systematic schemes

- Threshold Implementation, provable security

Threshold Implementation

Let's consider an Sbox:

$$x_1 \oplus x_2 \oplus x_3 = x$$

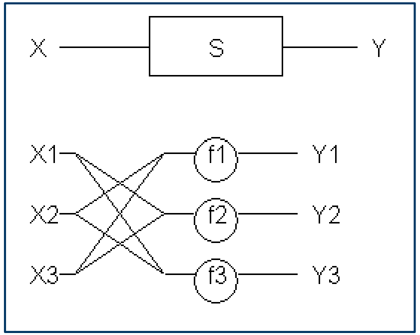


$$y_1 \oplus y_2 \oplus y_3 = y$$

Threshold Implementation

Let's consider an Sbox:

$x_1 \oplus x_2 \oplus x_3 = x$



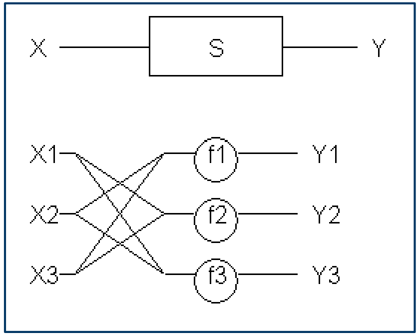
$y_1 \oplus y_2 \oplus y_3 = y$

Each f should be independent of one share

Threshold Implementation

Let's consider an Sbox:

$x_1 \oplus x_2 \oplus x_3 = x$



$y_1 \oplus y_2 \oplus y_3 = y$

Example:

Each f should be independent of one share

$x = (a, b, c, d) \quad y = (e, f, g, h)$

$S_1(a, b, c, d) = e$

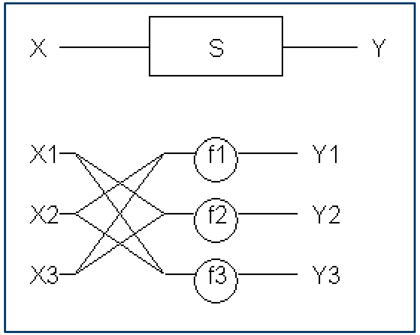
$e = a \oplus bc \oplus d$

$e = a_1 \oplus a_2 \oplus a_3 \oplus b_1c_1 \oplus b_1c_2 \oplus b_1c_3 \oplus b_2c_1 \oplus$
$b_2c_2 \oplus b_2c_3 \oplus b_3c_1 \oplus b_3c_2 \oplus b_3c_3 \oplus d_1 \oplus d_2 \oplus d_3$

Threshold Implementation

Let's consider an Sbox:

$$x_1 \oplus x_2 \oplus x_3 = x$$



$$y_1 \oplus y_2 \oplus y_3 = y$$

Example:

$x = (a, b, c, d) \quad y = (e, f, g, h)$

$S_1(a, b, c, d) = e$

$e = a \oplus bc \oplus d$

$$e = a_1 \oplus a_2 \oplus a_3 \oplus b_1c_1 \oplus b_1c_2 \oplus b_1c_3 \oplus b_2c_1 \oplus b_2c_2 \oplus b_2c_3 \oplus b_3c_1 \oplus b_3c_2 \oplus b_3c_3 \oplus d_1 \oplus d_2 \oplus d_3$$

Each f should be independent of one share

are clear where to go (to which f)

can be arbitrarily distributed among two component functions

$$\boldsymbol{f_1} = \boldsymbol{b_2c_3 \oplus b_3c_2} \oplus \boldsymbol{a_2 \oplus d_2 \oplus b_2c_2}$$

$$\boldsymbol{f_2} = \boldsymbol{b_3c_1 \oplus b_1c_3} \oplus \boldsymbol{a_3 \oplus d_3 \oplus b_3c_3}$$

$$\boldsymbol{f_3} = \boldsymbol{b_1c_2 \oplus b_2c_1} \oplus \boldsymbol{a_1 \oplus d_1 \oplus b_1c_1}$$

How to Make a Masked Design?

Unprotected
Implementation
(behavioral verilog)

How to Make a Masked Design?

Unprotected Implementation (behavioral verilog)

→

Manual Design

- Not straightforward
- Based on experience
- Algorithmic level
- Prone to errors/defeats

How to Make a Masked Design?

Unprotected Implementation (behavioral verilog)

→

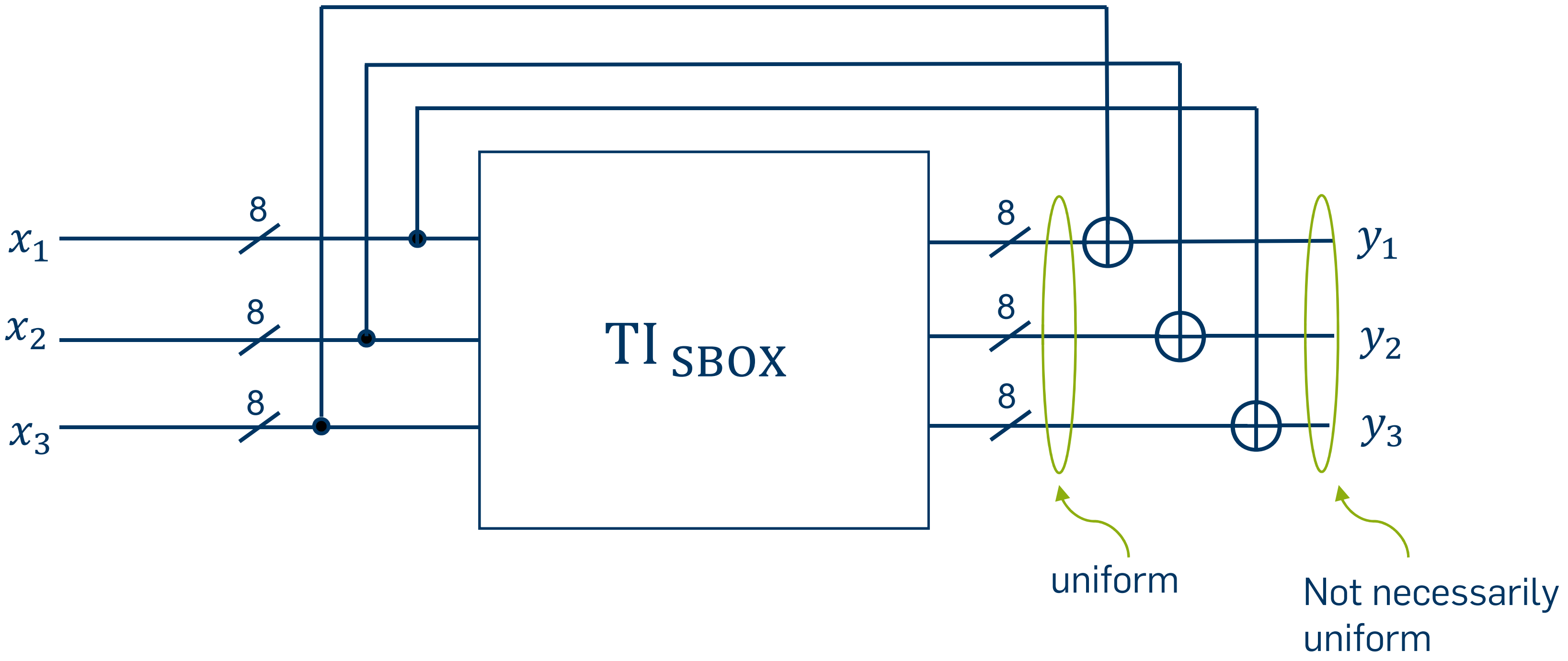
Manual Design

- Not straightforward
- Based on experience
- Algorithmic level
- Prone to errors/defeats

→

Side-Channel Protected Implementation (behavioral verilog)

Non-Composability in the TI context



Composability

Masking large and complex circuits is a hard task especially for high security orders

Composable hardware gadgets offer a systematic way to generate provable secure designs

- Arbitrary security orders possible
- Based on formal security notions
- Following divide-and-conquer approach based on fundamental building blocks
- Simply replacing unprotected gates (or larger modules) with its masked and composable counterpart

How to Make a Secure Design?

Unprotected Implementation (behavioral verilog)

→

Manual Design

- Not straightforward
- Based on experience
- Algorithmic level
- Prone to errors/defeats

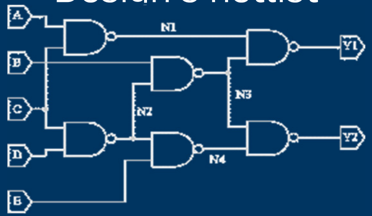
→

Side-Channel Protected Implementation (behavioral verilog)

How to Make a Secure Design?

Unprotected Implementation (behavioral verilog)

Design's netlist



Manual Design

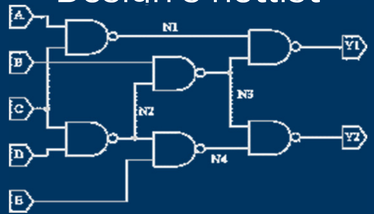
- Not straightforward
- Based on experience
- Algorithmic level
- Prone to errors/defeats

Side-Channel Protected Implementation (behavioral verilog)

How to Make a Secure Design?

Unprotected Implementation (behavioral verilog)

Design's netlist



Manual Design

- Not straightforward
- Based on experience
- Algorithmic level
- Prone to errors/defeats

Side-Channel Protected Implementation (behavioral verilog)

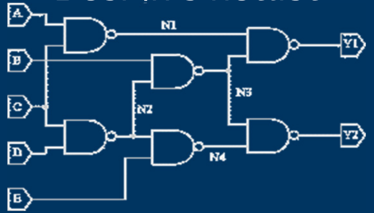
Settings

- Level of protection (order)?
- Optimize for area or speed?

How to Make a Secure Design?

Unprotected Implementation (behavioral verilog)

Design's netlist



Manual Design

- Not straightforward
- Based on experience
- Algorithmic level
- Prone to errors/defeats

Side-Channel Protected Implementation (behavioral verilog)

AGEMA

- Identifies gates/modules to be secured
- Replaces with equivalent variants
- Adjusts the control logic

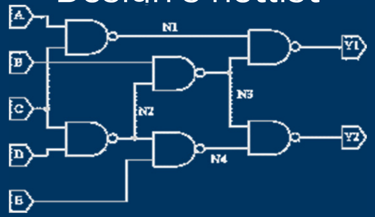
Settings

- Level of protection (order)?
- Optimize for area or speed?

How to Make a Secure Design?

Unprotected Implementation (behavioral verilog)

Design's netlist



Manual Design

- Not straightforward
- Based on experience
- Algorithmic level
- Prone to errors/defeats

Side-Channel Protected Implementation (behavioral verilog)

AGEMA

- Identifies gates/modules to be secured
- Replaces with equivalent variants
- Adjusts the control logic

Side-Channel Protected Implementation (netlist)

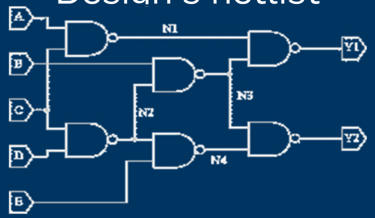
Settings

- Level of protection (order)?
- Optimize for area or speed?

How to Make a Secure Design?

Unprotected Implementation (behavioral verilog)

Design's netlist



Manual Design

- Not straightforward
- Based on experience
- Algorithmic level
- Prone to errors/defeats

Side-Channel Protected Implementation (behavioral verilog)

AGEMA

- Identifies gates/modules to be secured
- Replaces with equivalent variants
- Adjusts the control logic

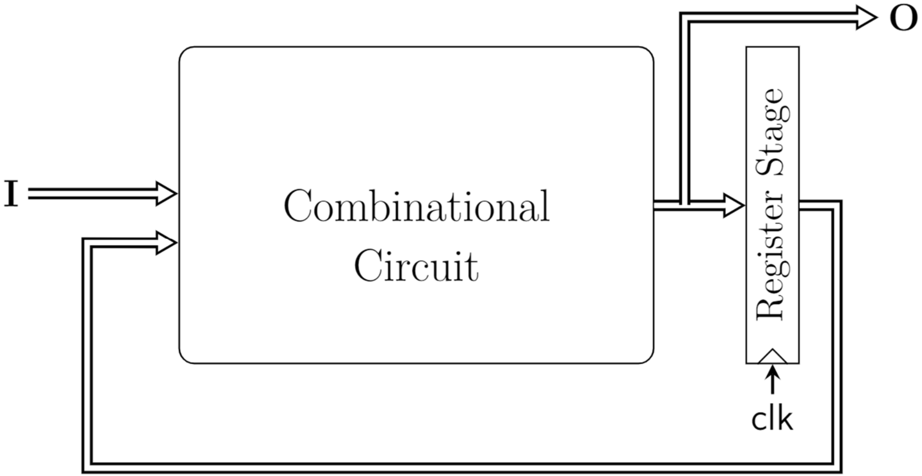
Side-Channel Protected Implementation (netlist)

- Free of heuristics
- Based on proofs
- Free of engineering's failures
- Open-source (GitHub)

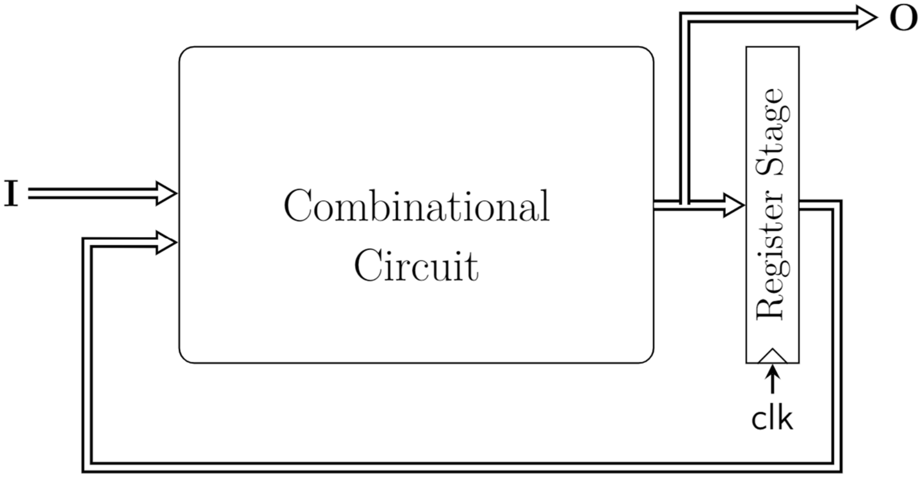
Settings

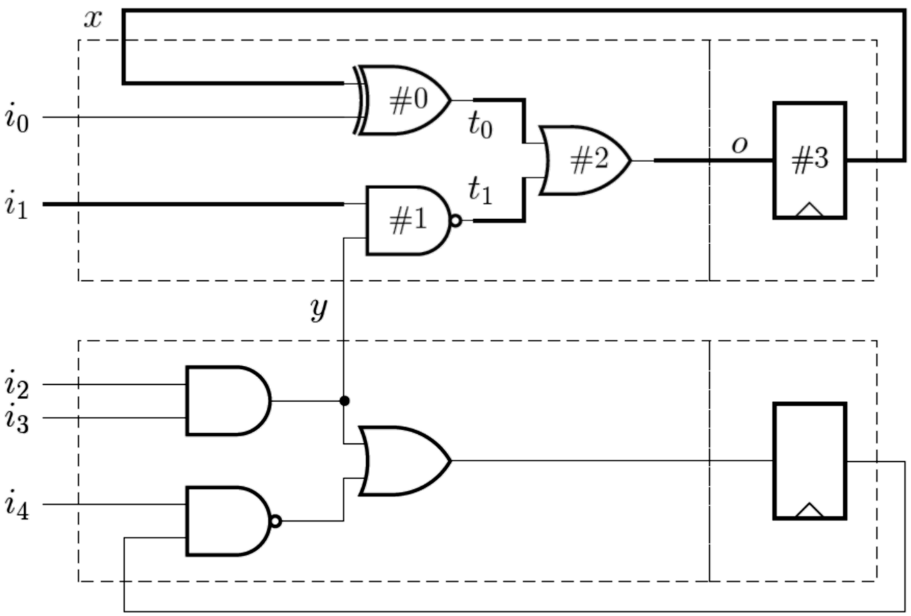
- Level of protection (order)?
- Optimize for area or speed?

Circuit Model

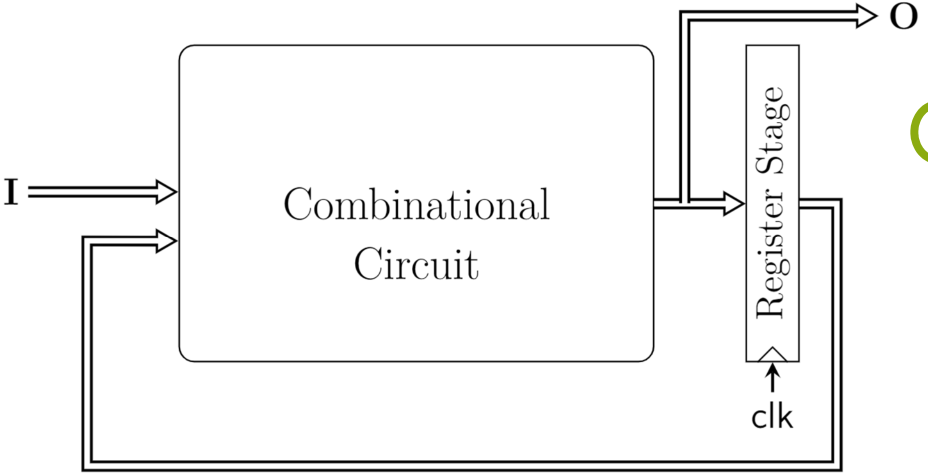


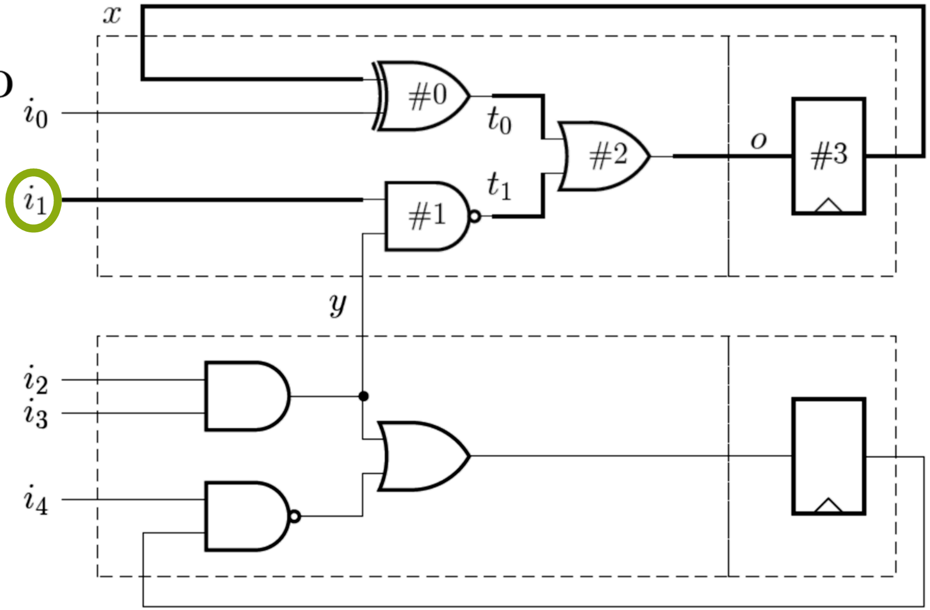
Circuit Model



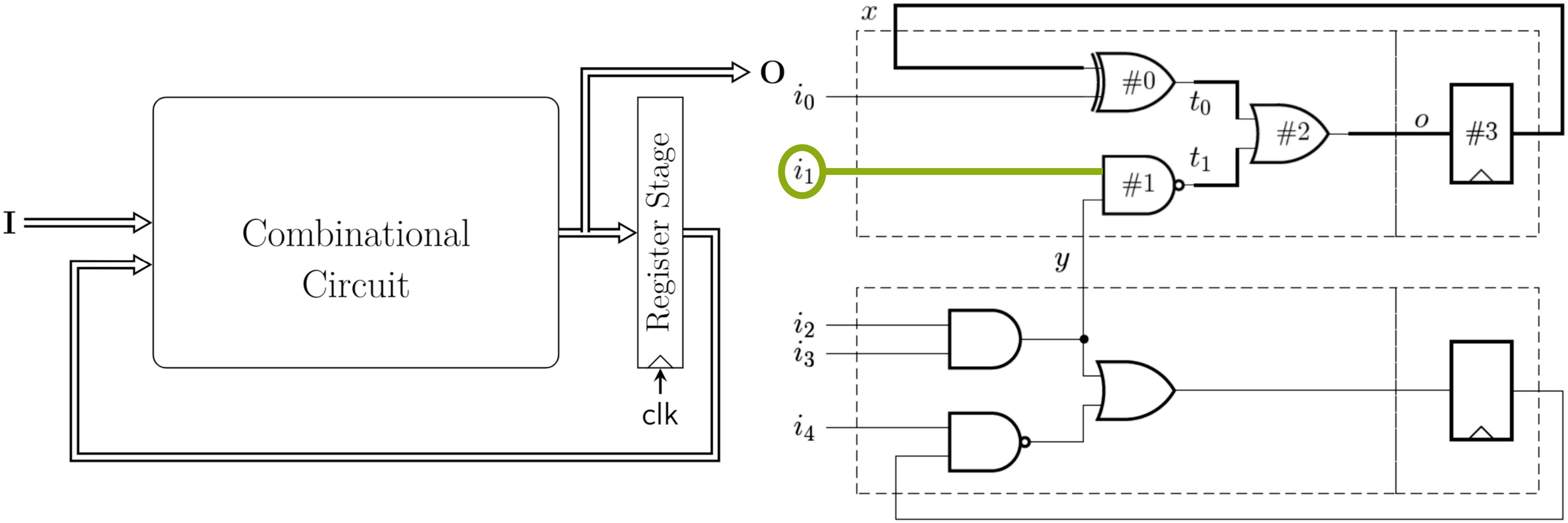


Circuit Model

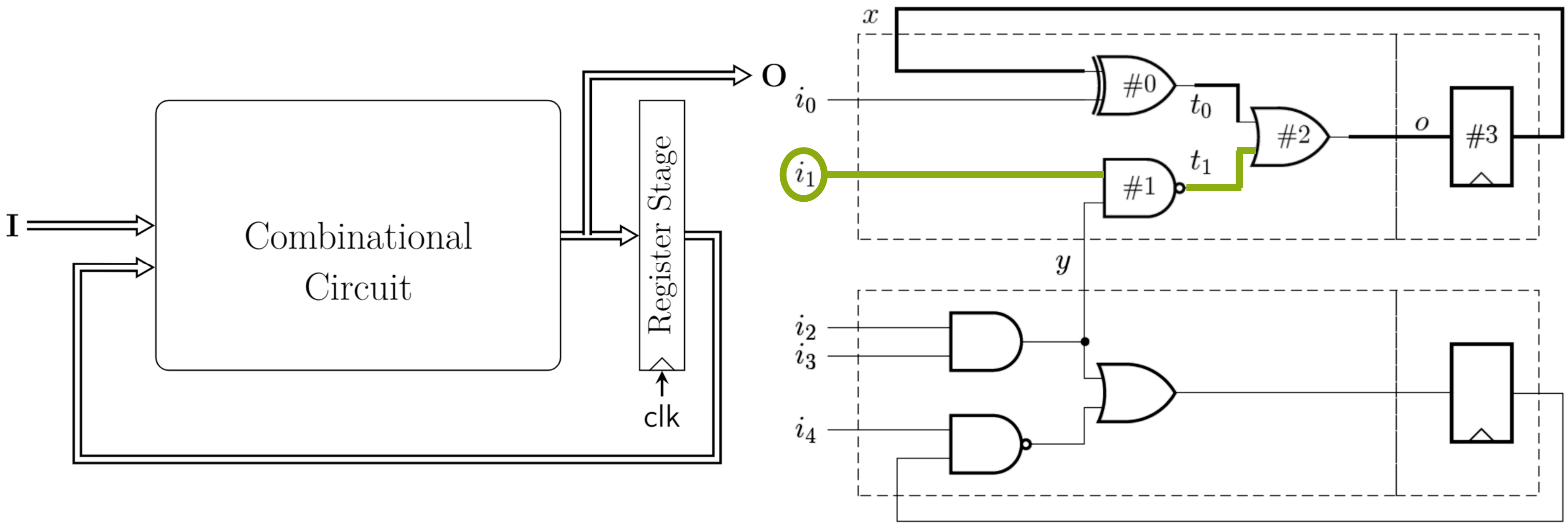




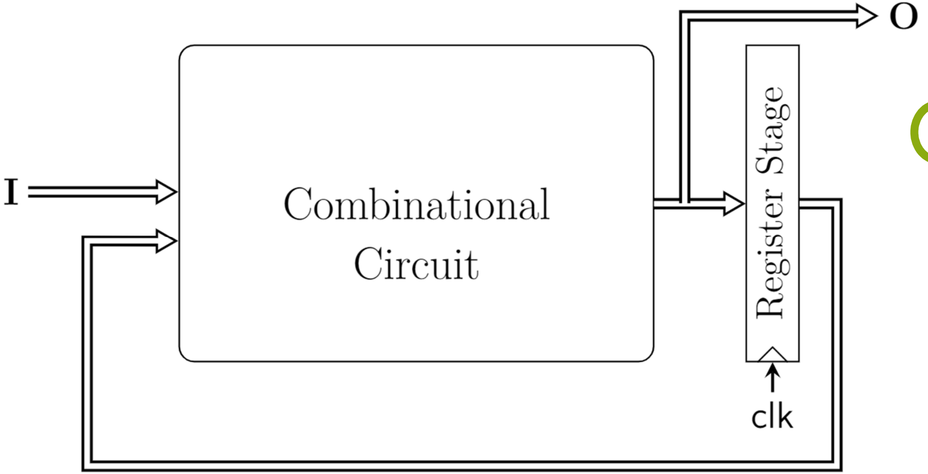
Circuit Model

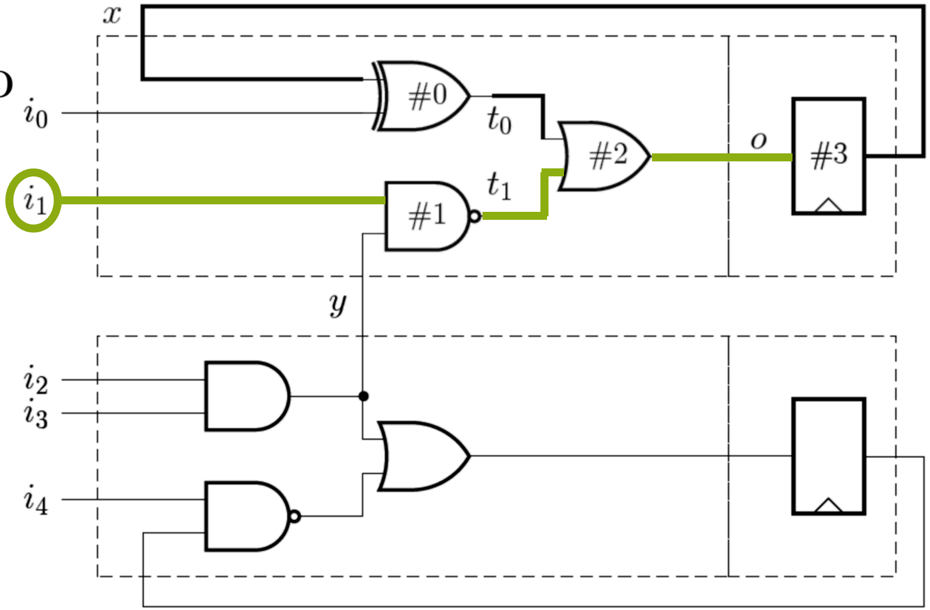


Circuit Model

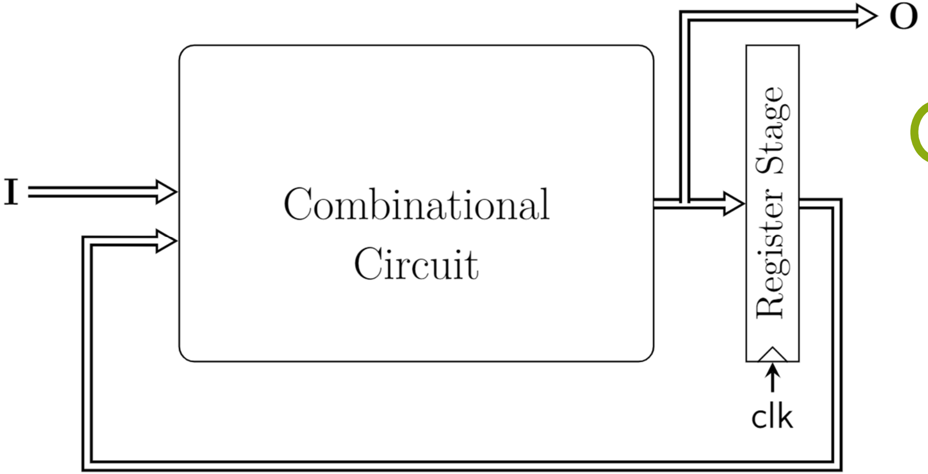


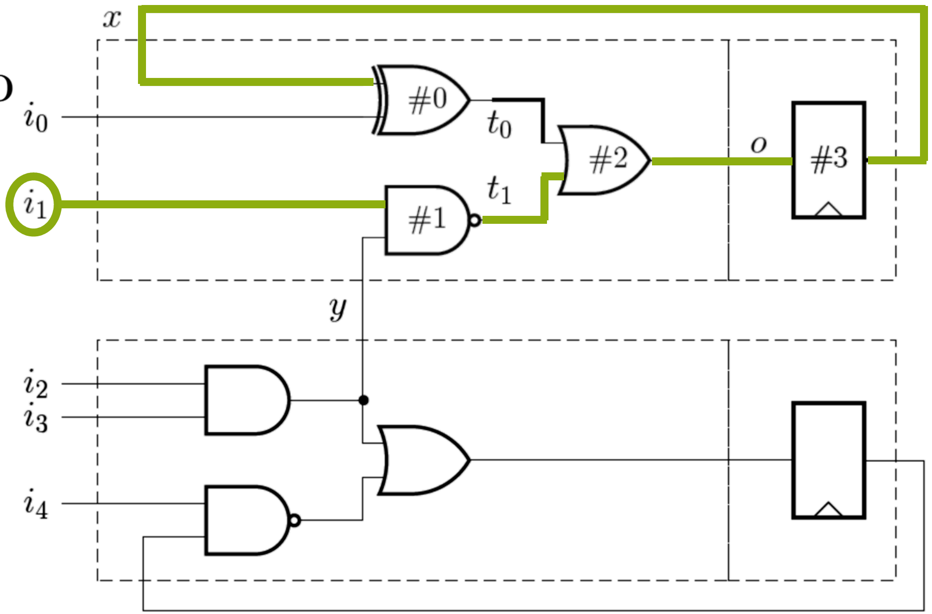
Circuit Model



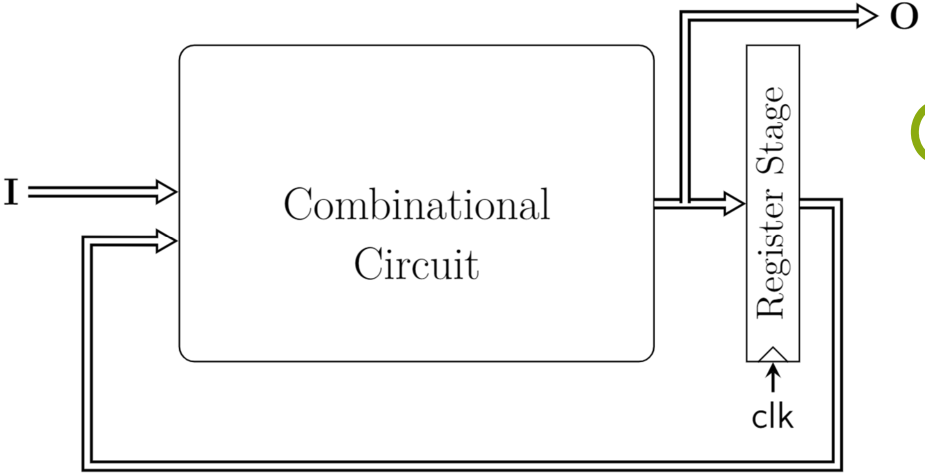


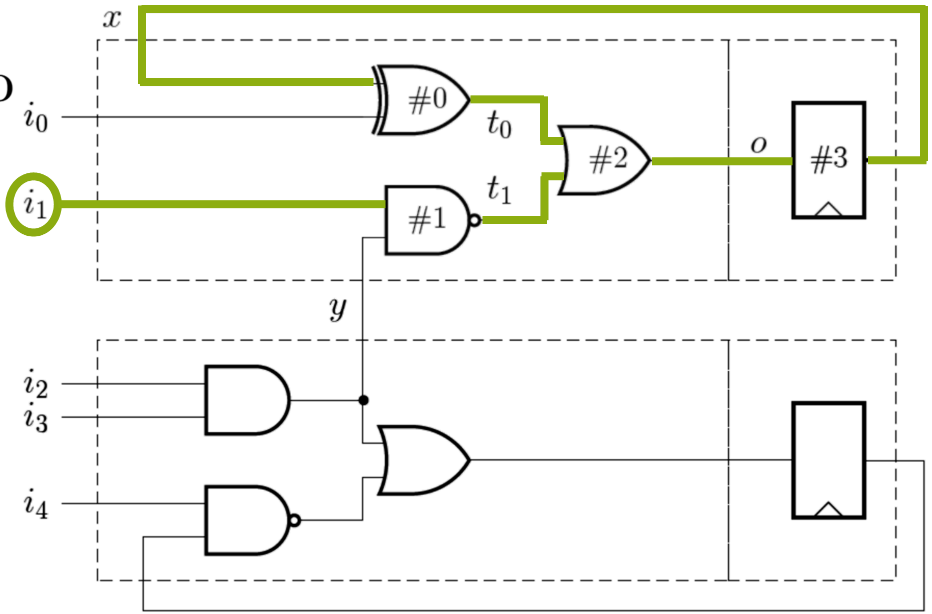
Circuit Model



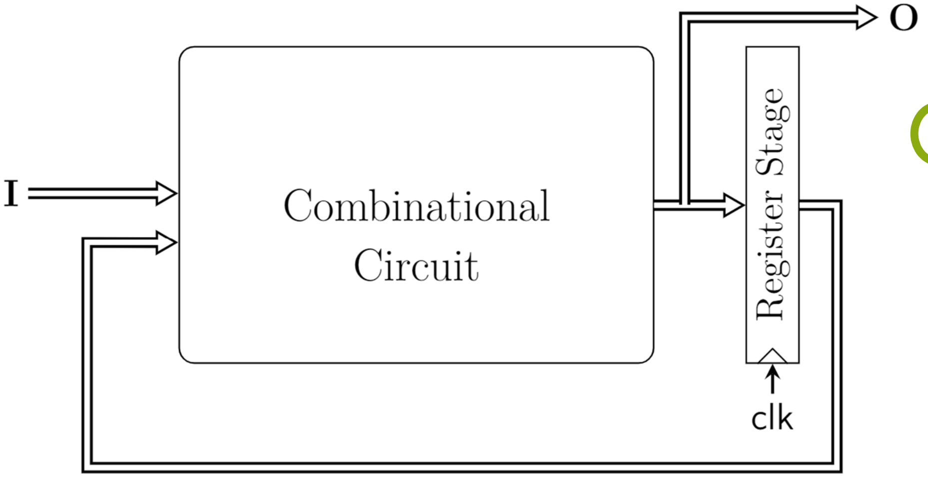


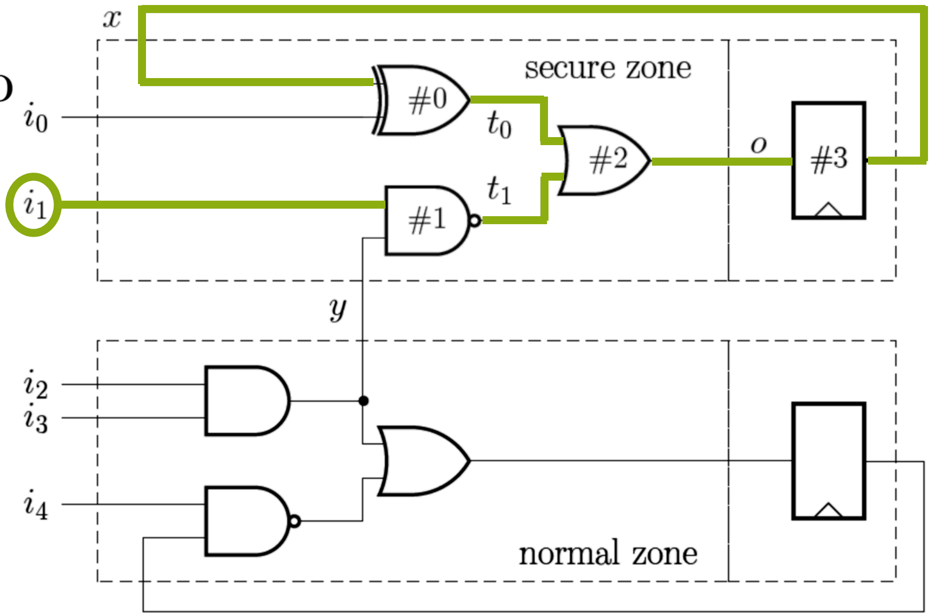
Circuit Model



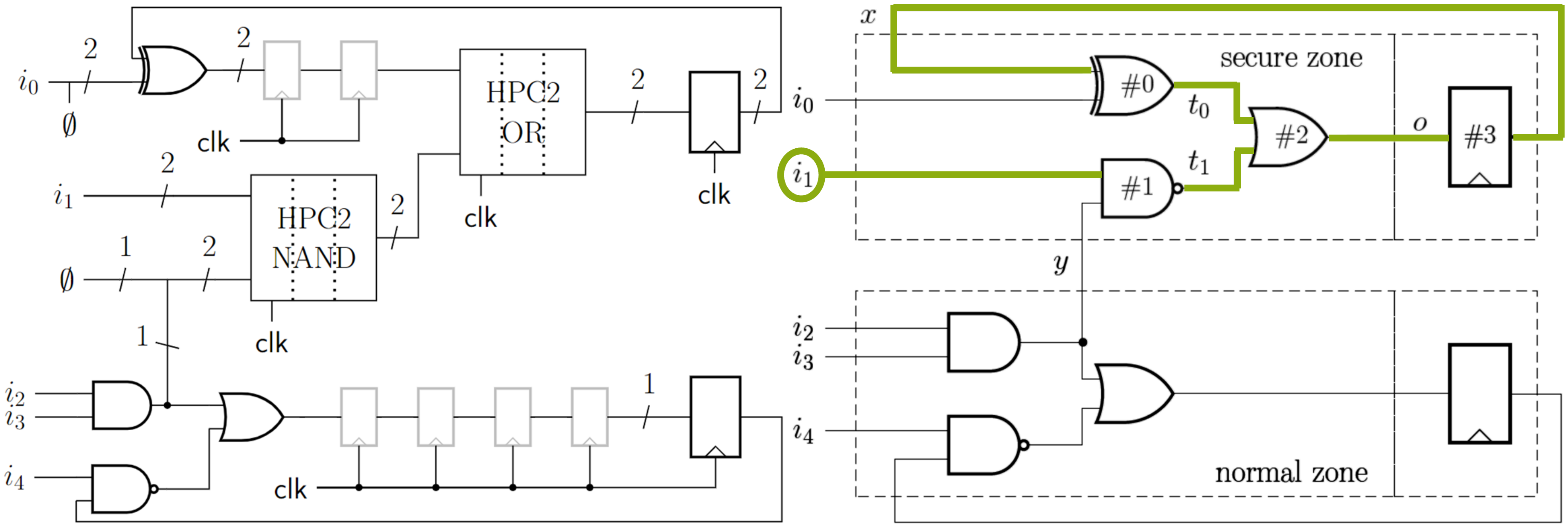


Circuit Model

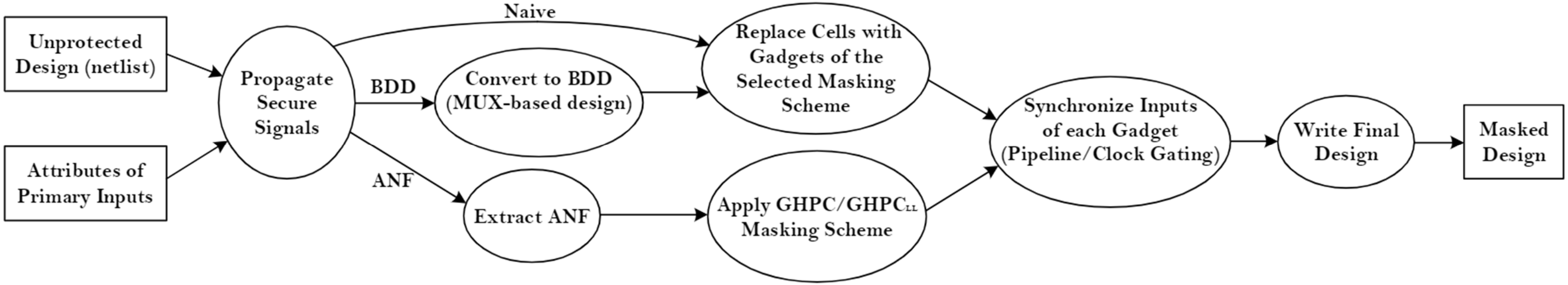




Example


2
i_0
$\emptyset$
2
clk
HPC2
OR
2
2
2
i_1
HPC2
NAND
2
clk
clk
1
2
$\emptyset$
1
clk
i_2
i_3
i_4
1
clk
x
secure zone
i_0
#0
t_0
#2
o
#3
i_1
#1
t_1
y
i_2
i_3
i_4
normal zone

General Procedure



AGEMA

- **Requirements**
  - Composable security
    - A secure circuit is not necessarily secure when composed
    - PINI (Probe-Isolating Non-Interference)
  - PINI gadgets of essential gates
    - AND/NAND/OR/NOR/...

AGEMA

- **Requirements**
  - Composable security
    - A secure circuit is not necessarily secure when composed
    - PINI (Probe-Isolating Non-Interference)
  - PINI gadgets of essential gates
    - AND/NAND/OR/NOR/…

- **Efficiency**
  - Provable security
    - As long as the gadgets are PINI
  - Extendable to any arbitrary order
  - Not as efficient as manually-crafted designs
    - Larger, higher latency, higher demand for fresh masks
  - Any engineer can make secure designs
    - https://github.com/Chair-for-Security-Engineering/AGEMA

Thanks!

Any Questions?

amir.moradi@rub.de

Standardization Process

https://csrc.nist.gov/Projects/masked-circuits