Side Channel Secure Software:
A Hardware Question

Roderick Bloem

Vedad Hadzic *Barbara Gigerl* *Marc Gourjon* *Hannes Gross*
Johannes Haring *Rinat Iusupov* *Bettina Koenighofer* *Stefan Mangard*
Robert Primas *Johannes Winter*

Eurocrypt'18 *FMCAD'21* *Usenix Security'21* *CCS'22* *CHES'24*

Graz University of Technology, NXP, TU Hamburg

Treasure

Exploiting Side Channels

Side Channels

World news

Pentagon pizza monitor predicted 'busy night' ahead of Israel's attack on Iran

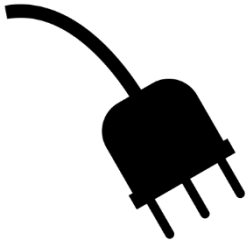
Low gay bar traffic and spike in pizza deliveries before strikes boost theory on Pentagon activity during invasions

Agence France-Press in Washington

Fri 13 Jun 2025 18.46 CEST

Side Channels

Execution time

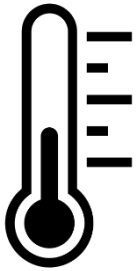


Power consumption

Electromagnetic radiation

Sound

Photon emission



Temperature

Side Channels

Execution time

Power consumption

Electromagnetic radiation

Sound

Photon emission

Temperature

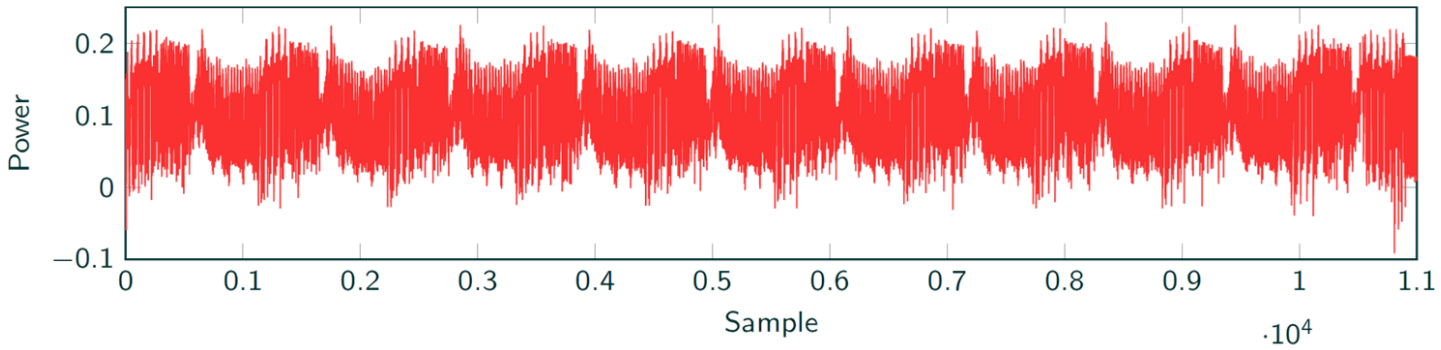
EM side channel attack
(aka power side channel):

- Assumes physical access
- Passive: observe, don't influence

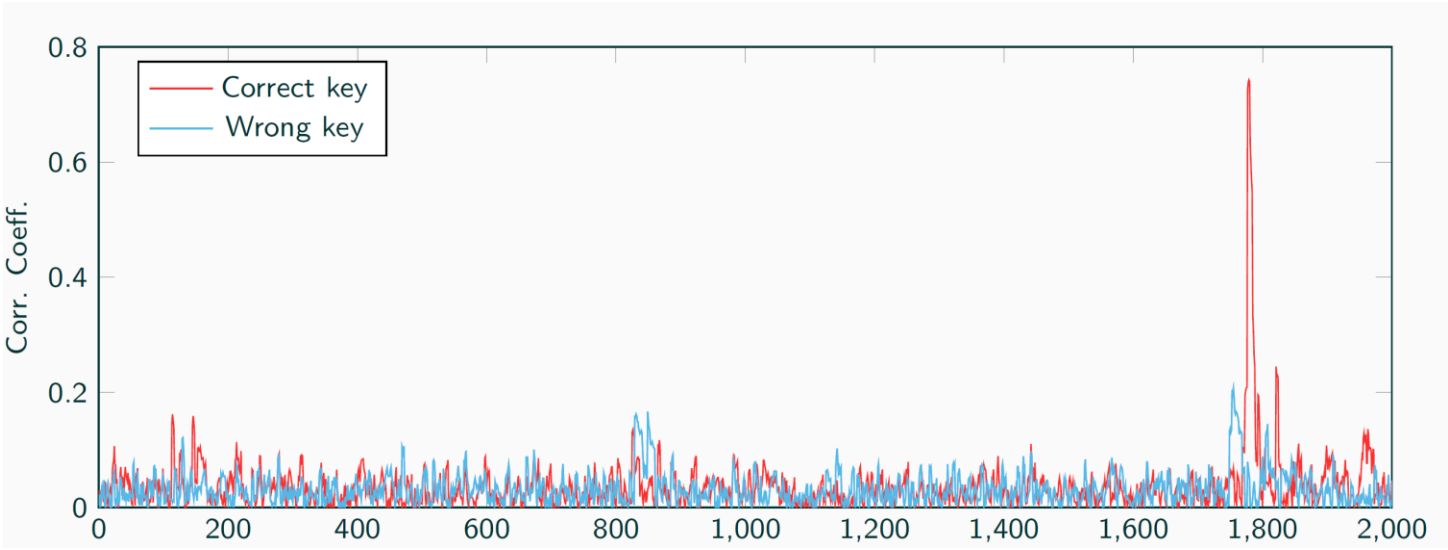
Predict, Measure & Correlate

Predict power usage for given key, one byte at a time

Measure true power usage



Correlate



YubiKey 5 Attack (Thomas Roche, NinjaLab, ’24)

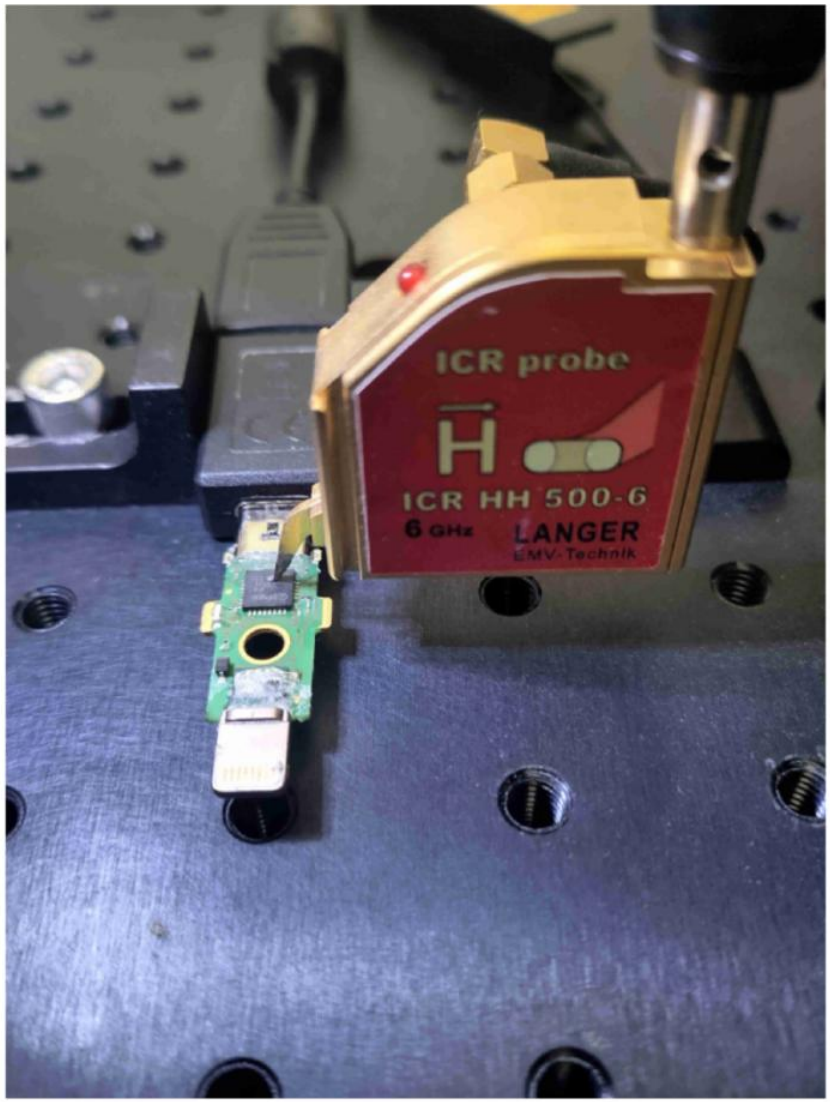


Figure 1.4: YubiKey 5Ci – EM Acquisition Setup

https://ninjalab.io/wp-content/uploads/2024/10/20241022_eucleak.pdf

Security Model

“Static” Model:

- Attacker measures **value** of d wires arbitrarily often
- If measurement and secret correlate, secret is discovered

For this talk, $d = 1$

- Well-accepted, but a **model:**
  - *Overestimates* attacker: no noise, ∞ measurements
  - *Underestimates* attacker: Measures multiple signals

Attacker’s axiom:
Correlation Equals Knowledge

Statistical Dependence

x_1	x_2	m	$y = (x_1 \wedge m) \oplus x_2$
F	F	F	F
F	F	T	F
F	T	F	T
F	T	T	T
T	F	F	F
T	F	T	T
T	T	F	T
T	T	T	F

Suppose m, x_2 uniformly random.

Can attacker find x_1 by observing y repeatedly?

No. y statistically **independent** of x_1: $\mathrm{P}(\mathrm{x}_1 \mid y) = P(x_1 \mid \neg y)$

Note: y is functionally dependent on x_1: $f(F,T,T) \neq f(T,T,T)$

Statistical Dependence

x_1	x_2	m	$y = (x_1 \wedge m) \oplus x_2$
F	**F**	F	F
F	**F**	T	F
F	**T**	F	T
F	**T**	T	T
T	**F**	F	F
T	**F**	T	T
T	**T**	F	T
T	**T**	T	F

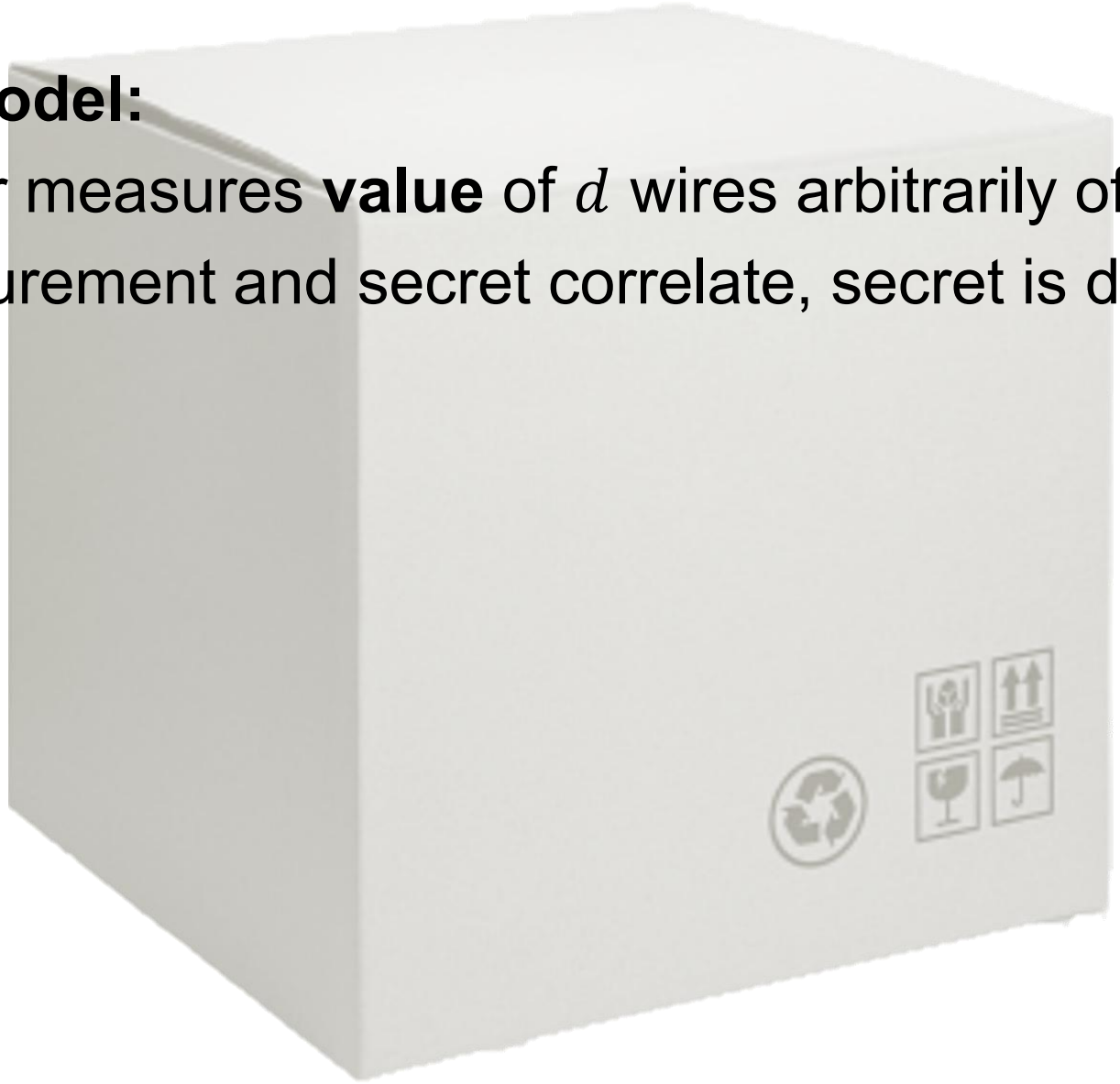
Suppose m, x_1 uniformly random.
Can attacker find $\boldsymbol{x_2}$ by observing y repeatedly?

Yes. y **correlates** positively with x_2

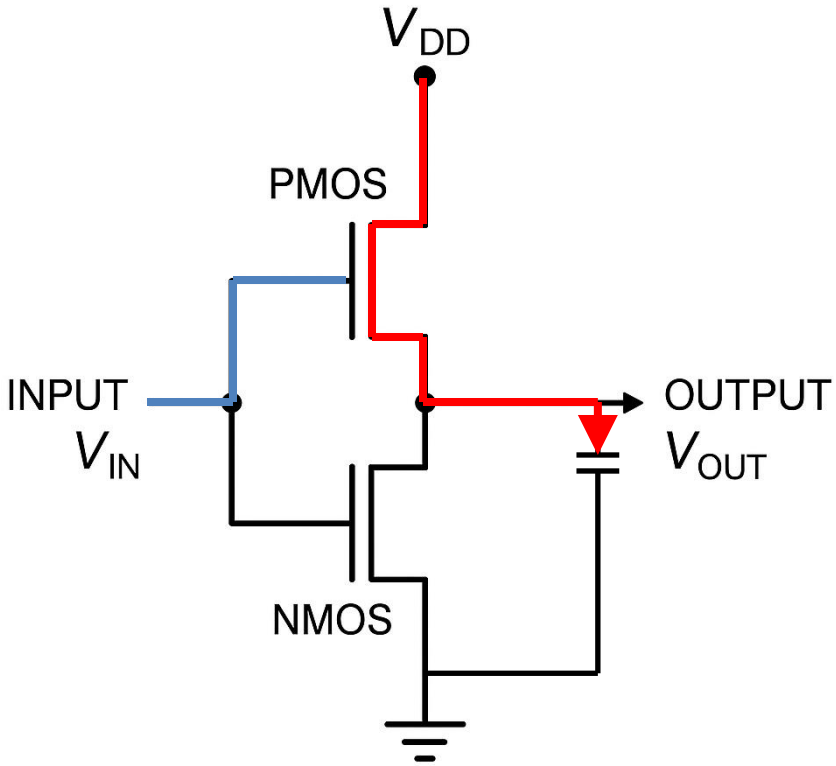
Let's Revisit Model

"Static" Model:

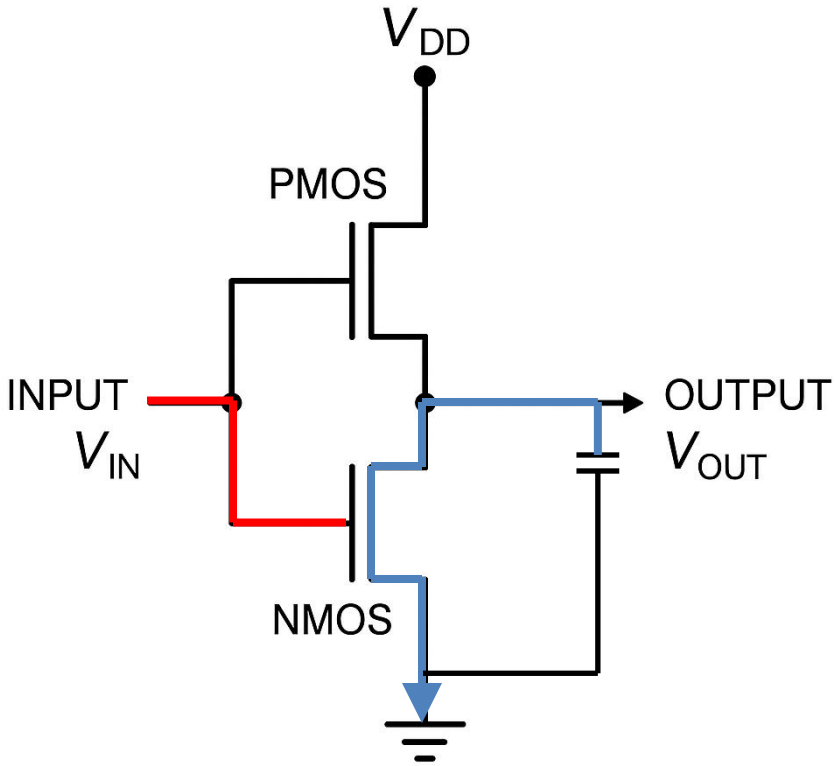
- Attacker measures **value** of d wires arbitrarily often
- If measurement and secret correlate, secret is discovered



CMOS Power Usage

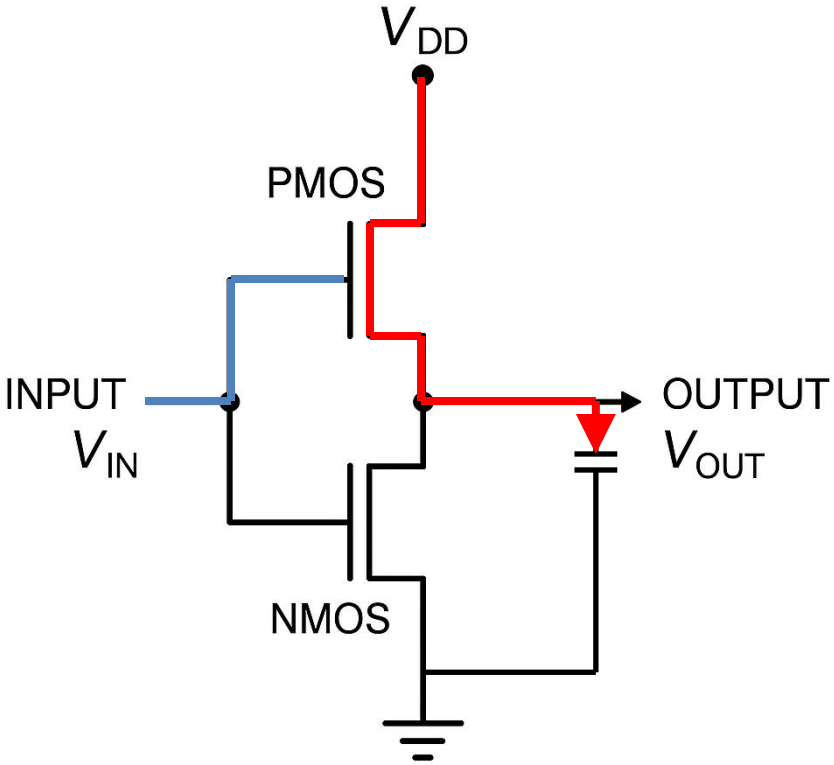


CMOS Power Usage



CMOS Power Usage

- Dynamic
  - Capacitive: (dis-)charge output wire
  - Short circuit: Both transistors conduct during switch
- Static: imperfect isolation



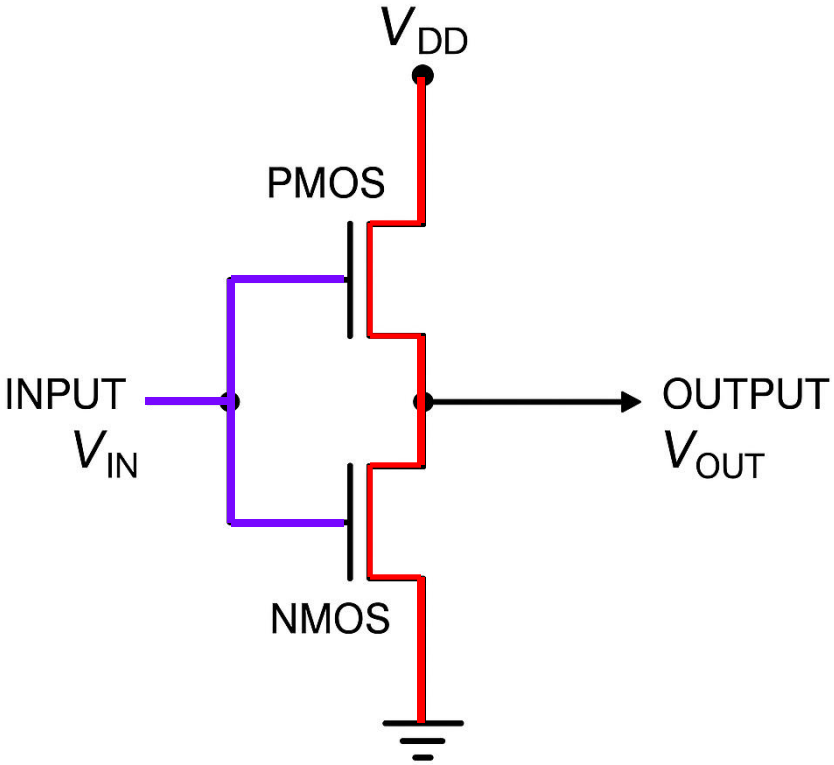

```
w = x0;
w = x1;
```

$W \sim (x_0 \oplus x_1)$.

CMOS Power Usage

- Static: imperfect isolation

- Dynamic
  - Capacitive: (dis-)charge output wire
  - Short circuit: Both transistors partially open during switch



Security Model

Static Model: Attacker measures **value** of d

Dynamic Model: Attacker measures **transitions** of d wires

Another problem:

*Glitching: v*alues depend on propagation speed

Masking Protects(?)

Countermeasure: Masking

- Split secret $\boldsymbol{s}$ into $d + 1$ *shares*
- Add randomness $\boldsymbol{m}$ to shares

Requirements:

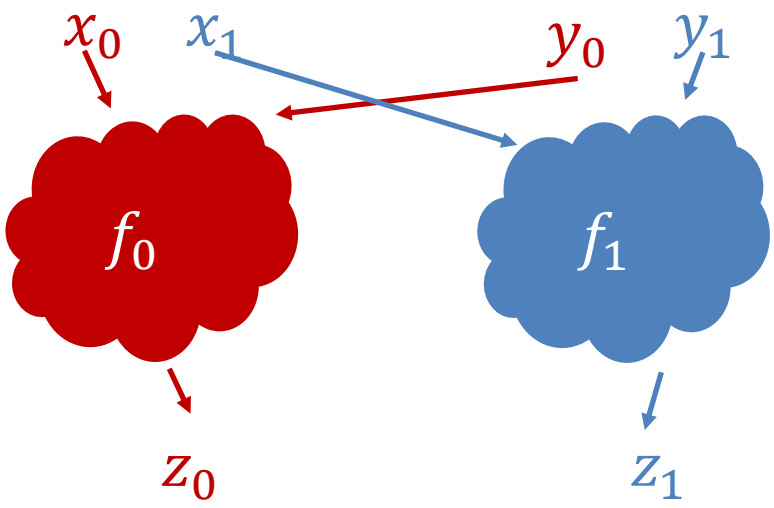
1. I can reconstruct s from s_0 **and** s_1
2. I **cannot** reconstruct s from **only** s_0 **or only** s_1

Main Principle of Masking:
XOR with mask hides secret

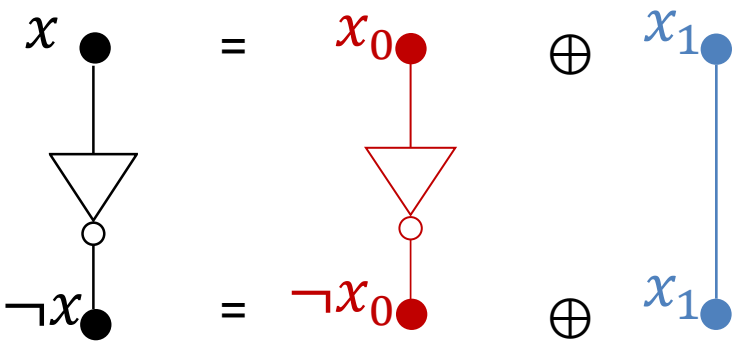
Masking: Computations

Golden Rule of Masking:
Don't combine shares

$$z = f(x, y)$$



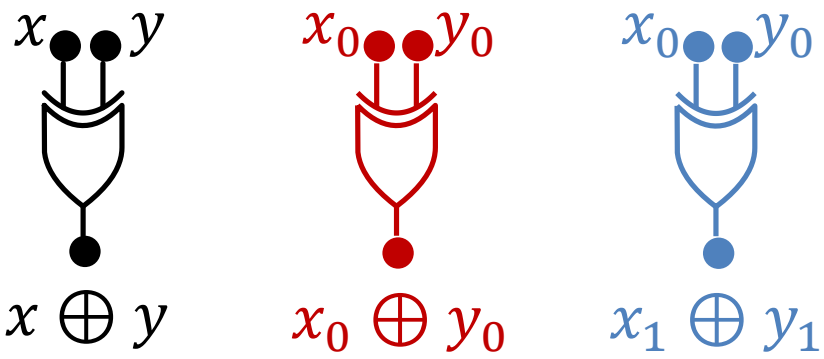
Masking Computations: NOT



Correct: $\neg x_0 \oplus x_1 = \neg x$

Secure: $\neg x_0, x_1$ statistically independent of x

Masking Computations: XOR



Correct: $(x_0 \oplus y_0) \oplus (x_1 \oplus y_1) = (x_0 \oplus x_1) \oplus (y_0 \oplus y_1) = x \oplus y$

Secure: $(x_0 \oplus y_0)$ statistically independent of x and y

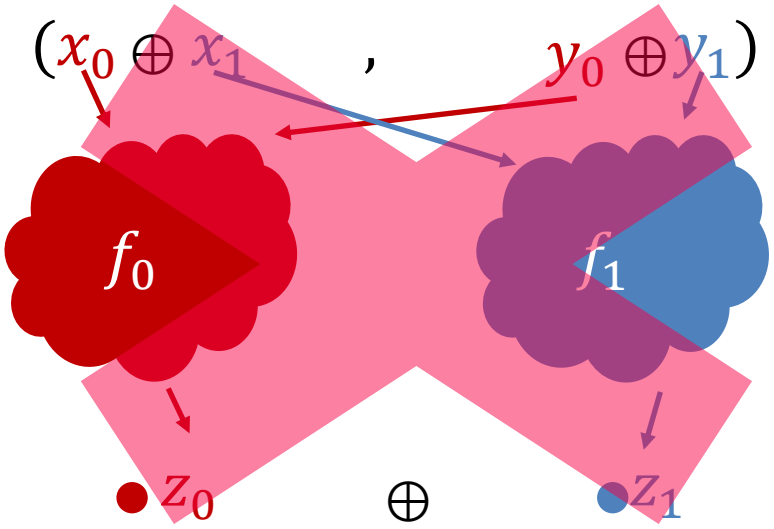
$(x_1 \oplus y_1)$statistically independent of x and y

Masking Computation: AND is not Easy

(x, y)

$z = x \wedge y$

=



A new construction is worth a publication!

Masking Computations: DOM AND

$$z = x \wedge y \rightarrow \begin{cases} z_0 = m' \oplus (x_0 \wedge y_0) \oplus (x_0 \wedge y_1) \\ z_1 = m' \oplus (x_1 \wedge y_0) \oplus (x_1 \wedge y_1) \end{cases}.$$

m' is fresh random

Correct: $z_0 \oplus z_1 = (x_0 \oplus x_1) \wedge (y_0 \oplus y_1)$

Secure: The following are statistically independent of x and y:

- m'
- $m' \oplus (x_0 \wedge y_0)$
- $(x_0 \wedge y_1)$
- $m' \oplus (x_0 \wedge y_0) \oplus (x_0 \wedge y_1)$
- Same for z_1

Easy to Get Wrong!

$$z = x \wedge y \rightarrow \begin{cases} z_0 = [m' \oplus (x_0 \wedge y_0)] \oplus (x_0 \wedge y_1) \\ z_1 = m' \oplus (x_1 \wedge y_0) \oplus (x_1 \wedge y_1) \end{cases}$$

Take instead

$$z_0 = m' \oplus [(x_1 \wedge y_0) \oplus (x_1 \wedge y_1)]$$

$(x_0 \wedge y_0) \oplus (x_0 \wedge y_1)$ correlates with $y_0 \oplus y_1 = y$!

Typical compiler optimization


Does any signal correlate with a secret?

Fourier (Walsh-Haldemard) Transform

Identify **False** $= 1$, **True** $= -1$

x	y	$x \oplus y$
1	1	1
1	-1	-1
-1	1	-1
-1	-1	1

$x \oplus y = x \cdot y$

$x^2 = 1$

Every function can be written as multilinear polynomial.

E.g.,

$$f(x,y) = c_{\emptyset} + c_{\{x\}}x + c_{\{y\}}y + c_{\{x,y\}}xy$$

$$c_S \in \mathbb{Q}$$

Fourier coefficients

Fourier (Walsh) Transform

Identify $\mathbf{False} = \mathbf{1}$, $\mathbf{True} = -\mathbf{1}$

Fourier Expansion of $x \wedge y$:

$$f(x,y) = \quad x \quad y \quad xy.$$

x	y	$x \wedge y$
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

"XOR normal form"

Magic of Fourier:
Coefficient is non-zero iff there is correlation

Statistical Independence

x_1	x_2	m	$y = (x_1 \wedge m) \oplus x_2$
F	F	F	F
F	F	T	F
F	T	F	T
F	T	T	T
T	F	F	F
T	F	T	T
T	T	F	T
T	T	T	F

y is statistically independent of x_1
y correlates positively with x_2

$$y = 0 + \mathbf{0} \cdot \boldsymbol{x_1} + \frac{\mathbf{1}}{\mathbf{2}} \cdot \boldsymbol{x_2} + 0 \cdot m + \frac{1}{2} x_1 x_2 + 0 \cdot m x_1 + \frac{1}{2} m x_2 - \frac{1}{2} m x_1 x_2$$

Verification Using Fourier Coefficients

Fact: f correlates with s iff $c_s \neq 0$

Corollary: Signal OK iff all nonzero Fourier coefficients either

1. Contain a random mask ($m \oplus s$), or
2. Contain no secrets

"Naked" secret s with nonzero coefficient leaks!

Approximating Fourier Coefficients

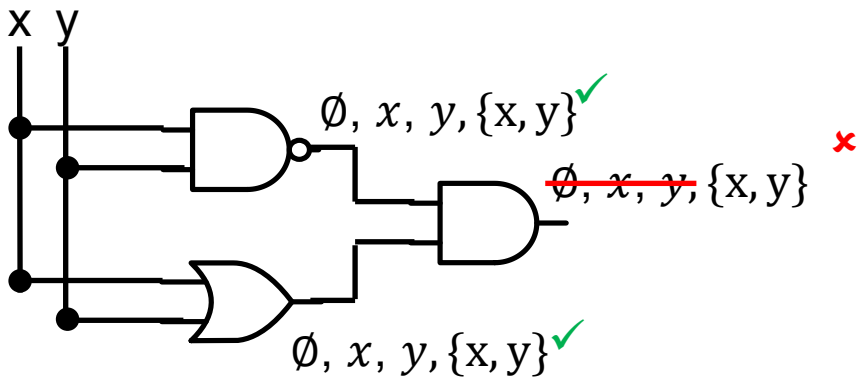
Problem: Fourier representations are exponential

Solution: Overapproximate as zero/non-zero

$x \wedge y = \frac{1}{2} + \frac{1}{2}x + \frac{1}{2}y - \frac{1}{2}xy$ becomes

"$x \wedge y$ correlates with x, y, $x \oplus y$"

This is an over-approximation:



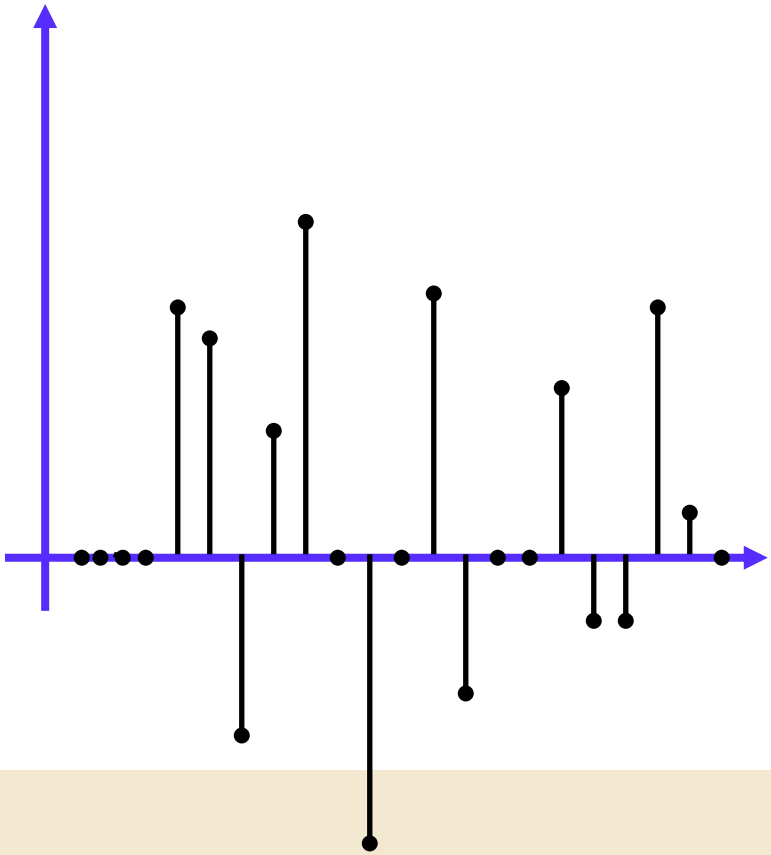
SAT Encoding

One variable per set of input variables and per wire in circuit.

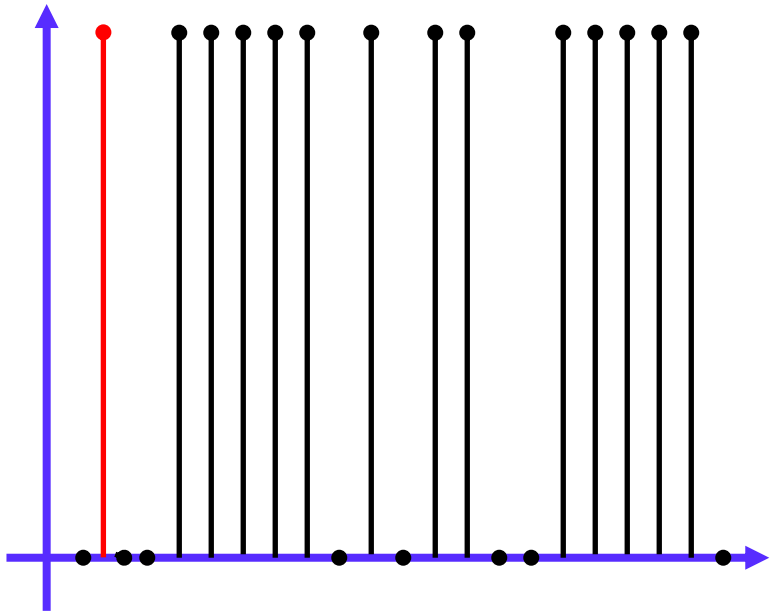
Lemma: variable $v_{w,S}$ = false $\rightarrow$ no correlation with $\bigoplus S$

Corollary: we don't miss leaks

Exact Fourier spectrum



Approximation



Hardware Verification

- We can verify correctness of
  - AES DOM implementation
  - Prince TI implementation
  - …
- Used in industry to verify security of crypto-implementations

Software

Software

$$z = x \wedge y \rightarrow \begin{cases} \boldsymbol{z_0} = \boldsymbol{m'} \oplus (\boldsymbol{x_0} \wedge \boldsymbol{y_0}) \oplus (\boldsymbol{x_0} \wedge \boldsymbol{y_1}) \\ z_1 = m' \oplus (x_1 \wedge y_0) \oplus (x_1 \wedge y_1) \end{cases}.$$

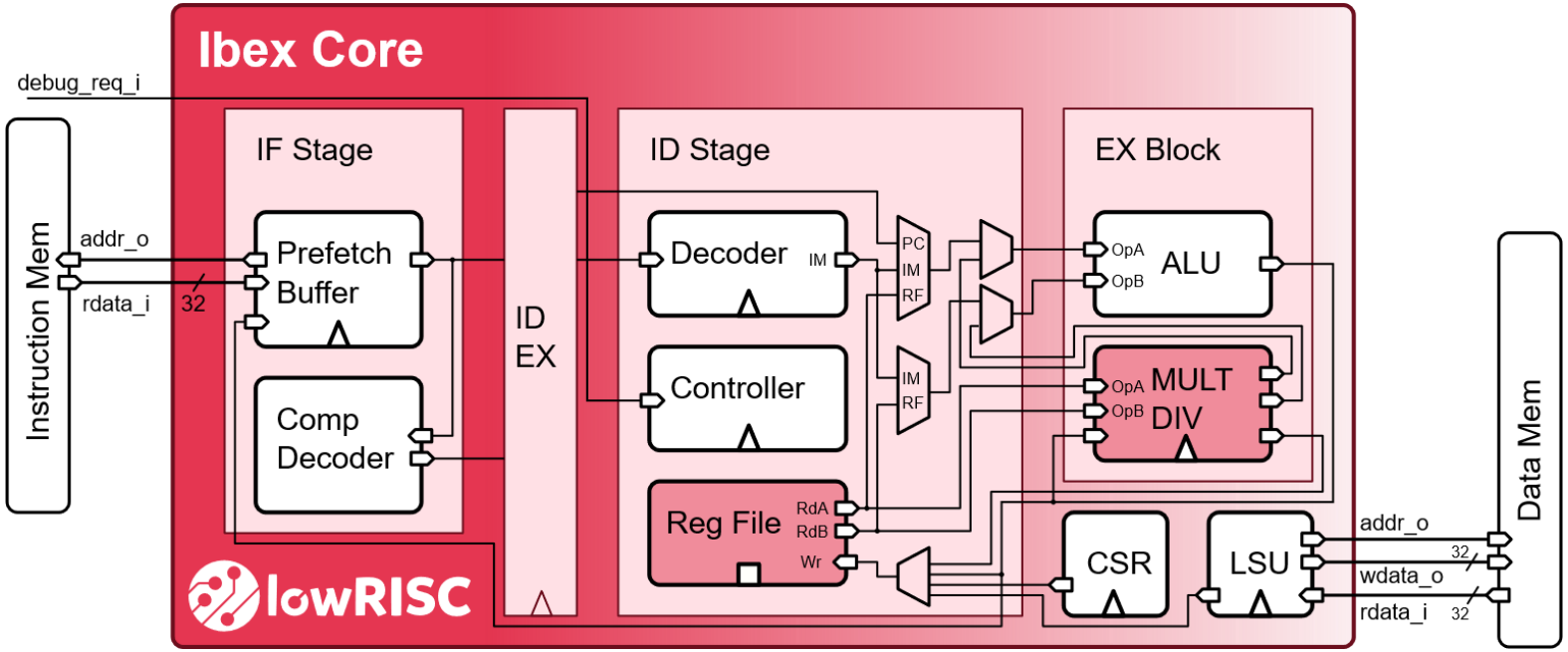
```
1: AND0(x0, y0, y1){
2:    u1 = x0 ∧ y0
3:    u3 = x0 ∧ y1
4:    u2 = m' ⊕ u1
5:    z0 = u2 ⊕ u3
6: }
```

- Use same verification tool…
- Do intermediate variable correlates with x or y?
- Does that mean there are no side channels?

CPUs

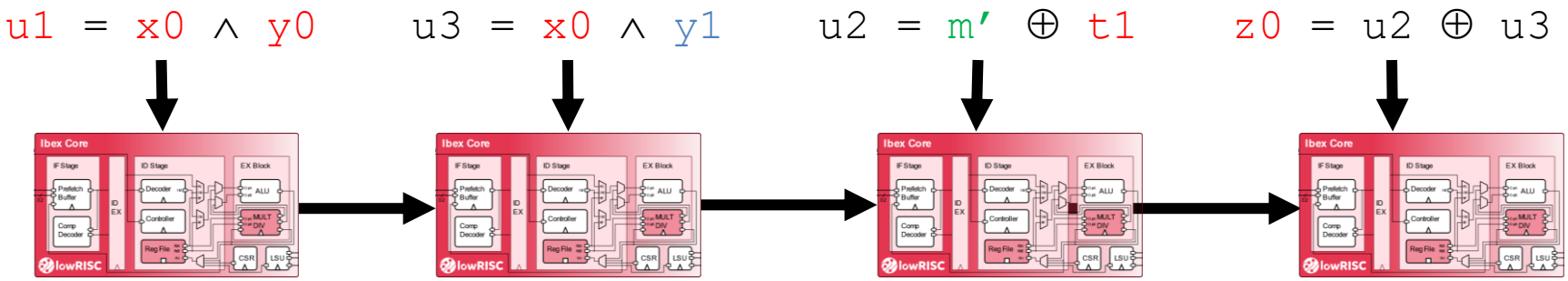
- RISC-V IBEX core
  - 32-bit CPU, two-stage in-order single-issue pipeline
  - Simple but contains most important components
  - Part of PULP and OpenTitan (in production)

opentitan



Microprocessor Verification

Run specific software on a processor, note leaks



Input to verification:

4 copies of CPU netlist, inputs set to program

Unexpected problems:

1. Microarchitectural registers
2. Register files
3. Computational Units

1. Microarchitectural Registers

Assembly Program:

```
1: AND0(x0, y0, y1){
2:    u1 = x0 ∧ y0

3:    u3 = x0 ∧ y1

4:    u2 = m' ⊕ u1
5:    z0 = u2 ⊕ u3
6: }
```

Vulnerable CPU implementation:

```
2.1: t1 = x0
2.2: t2 = y0
2.3: res = t1 ∧ t2
2.4: u1 = res

3.1: t1 = x0
3.2: t2 = y1
3.3: res = t1 ∧ t2
3.4: u2 = res
```

$r2 \oplus r2_{old} = y$

2. Register File = Tree of Multiplexers

x0 r000

r001

r010

x1 r011

r100

r101

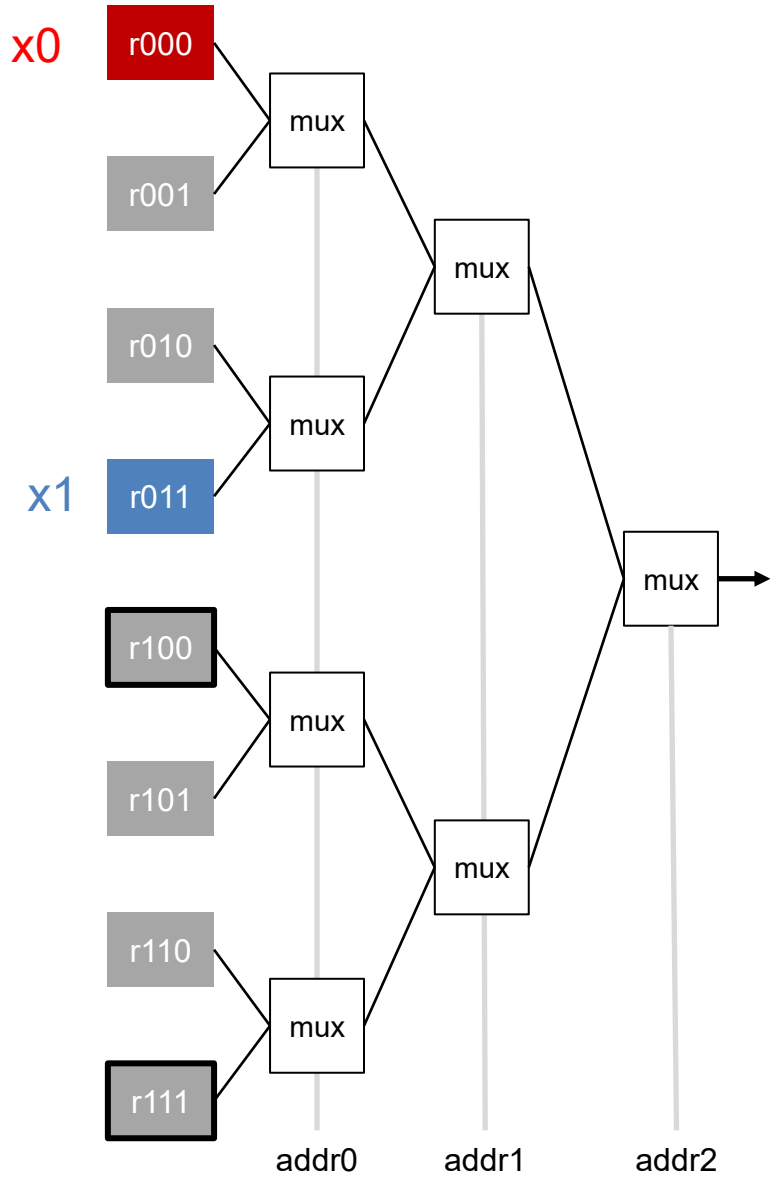
r110

r111

load r100

load r111

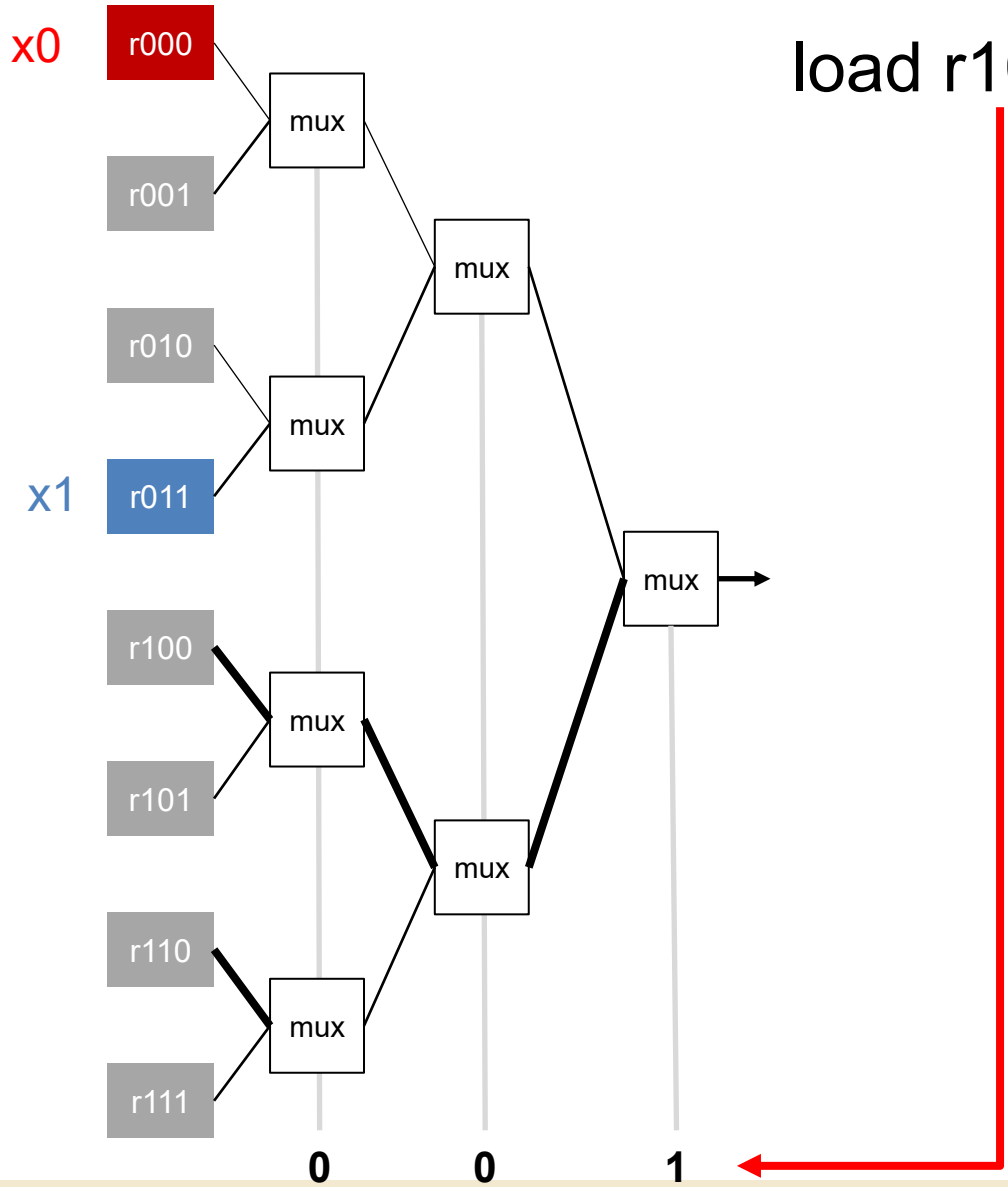
2. Register File = Tree of Multiplexers



load r100

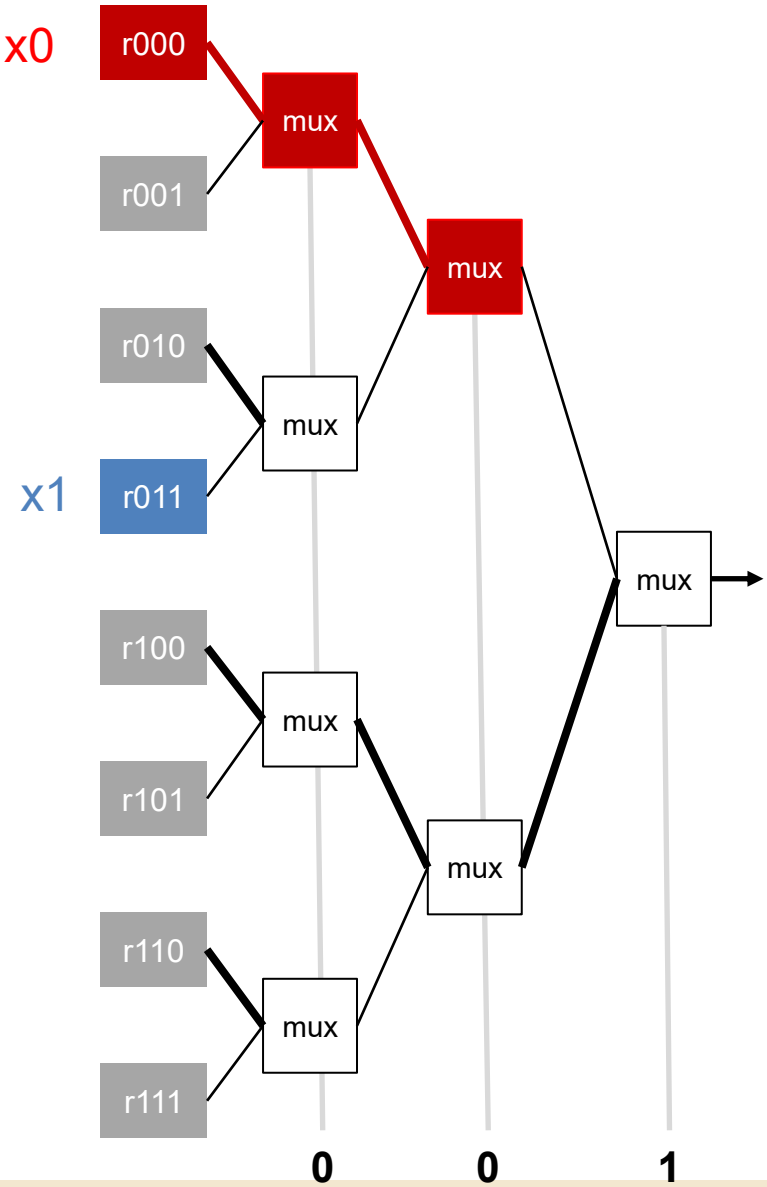
load r111

2. Register File

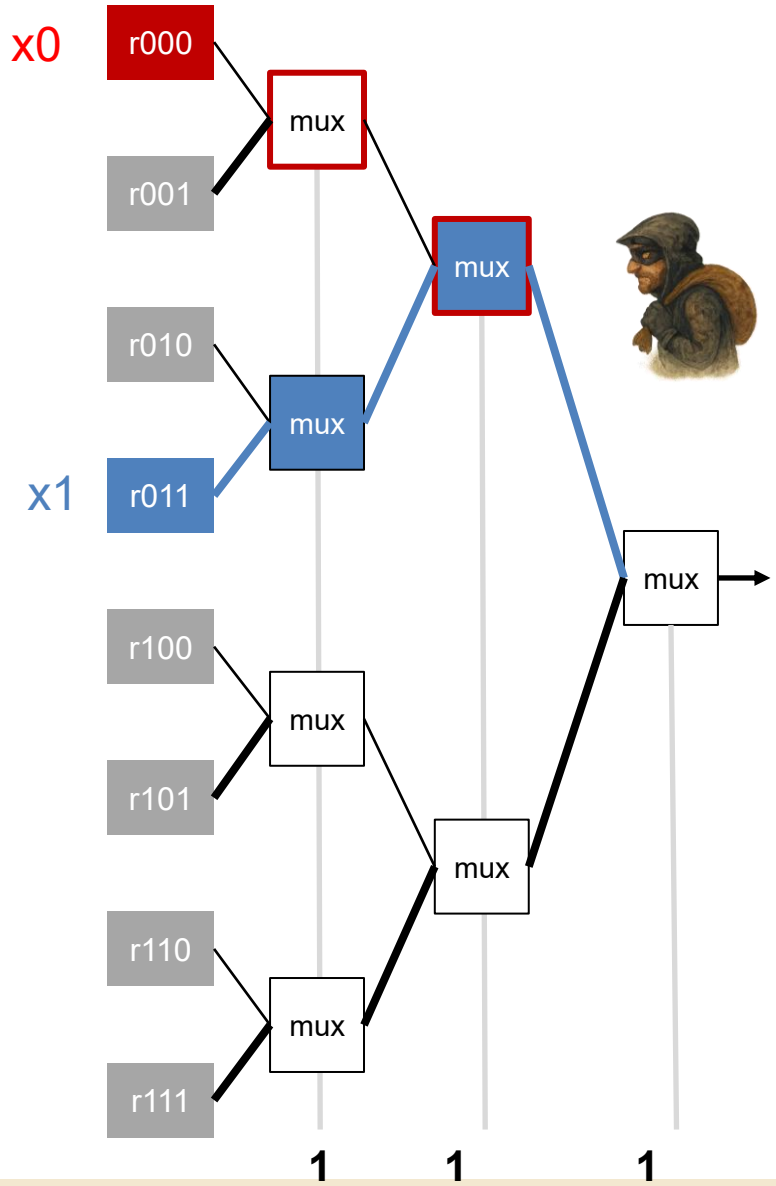


2. Register File

load r100



2. Register File

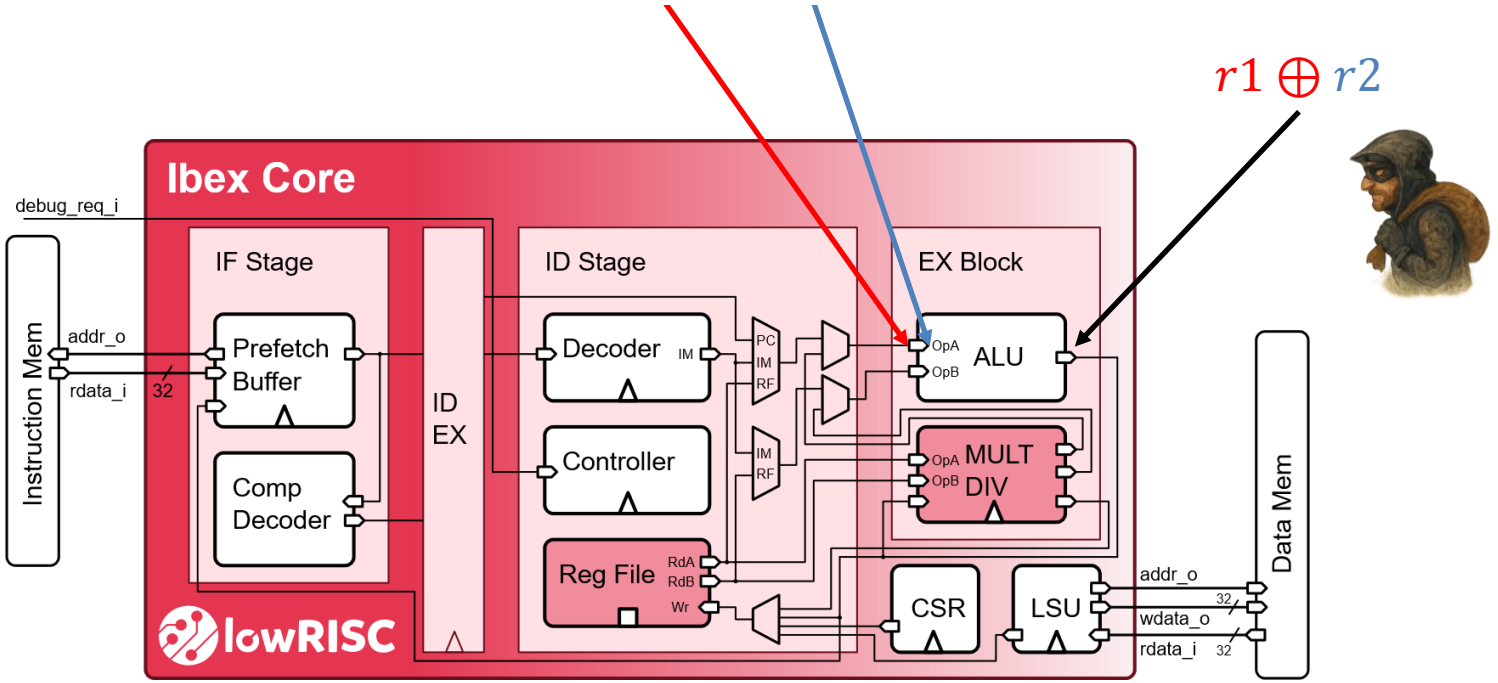


load r100

load r111

3. Computation Units

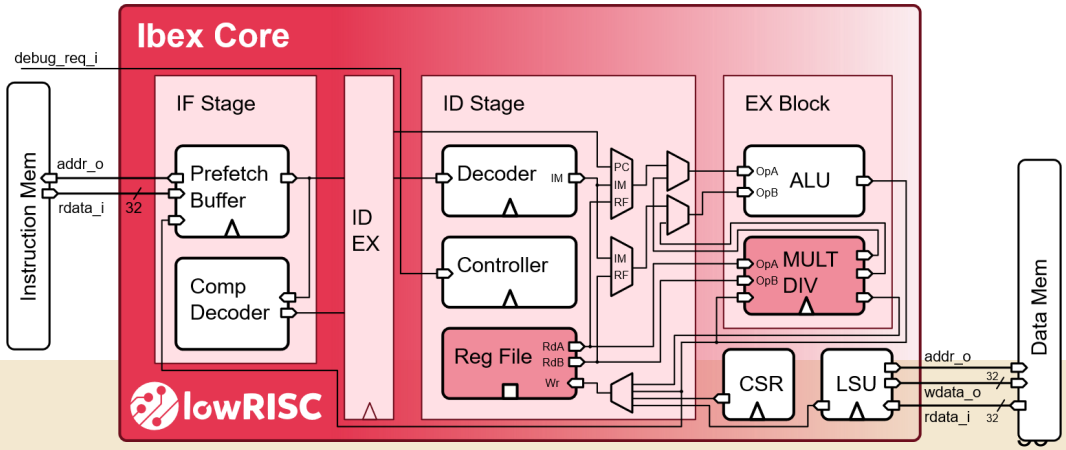
	31						24							16						8						0
opcode	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	1	1	0	1	1	0	1	1	1



Fixing Vulnerabilities

Fix involves combination of SW and HW

1. Microarchitectural registers
   - Software fix: No shares in subsequent instructions
2. Register File
   - Hardware fix: Gate registers
3. Computation Units
   - Hardware Fix: Gate Units

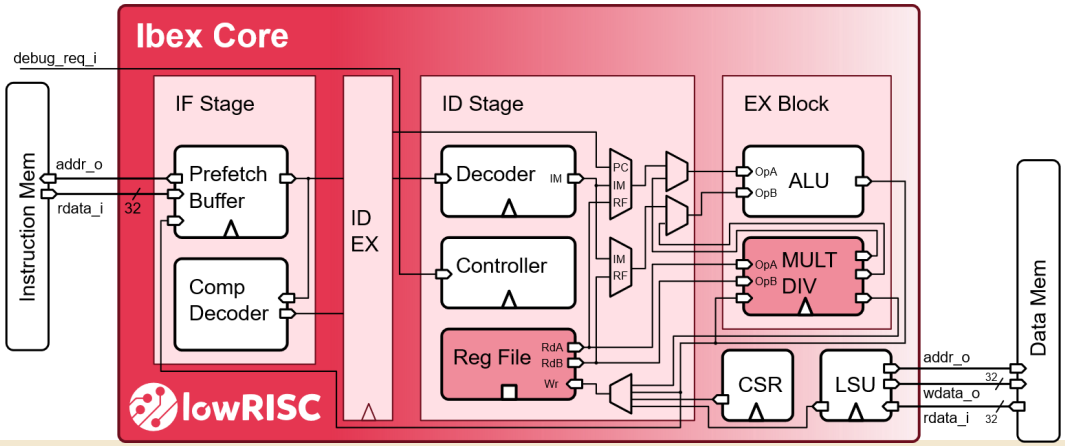


Fixing Vulnerabilities

Securing Ibex with Coco:

1. Construct set of masked programs
2. Execute on IBEX to get execution trace
3. Verify run
4. Fix problems, go to 1

Result: IBEX processor, **proven secure** *for some programs*

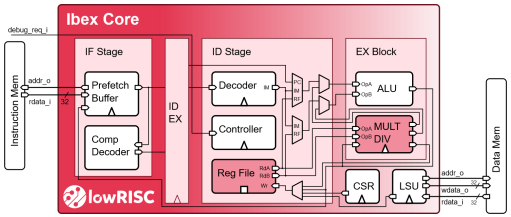


Leakage Contracts

Contract extends Instruction Set Architecture (ISA) with leakage

```
1: AND0(x0, y0, y1){
2:    t1 = x0 ∧ y0
3:    t3 = x0 ∧ y1
4:    t2 = m' ⊕ t1
5:    z0 = t2 ⊕ t3
6: }
```

Software Verifier



Hardware Verifier
(functional simulation)

Contract Example

Microarchitectural Registers: old_src1 = 0, old_src2 = 0

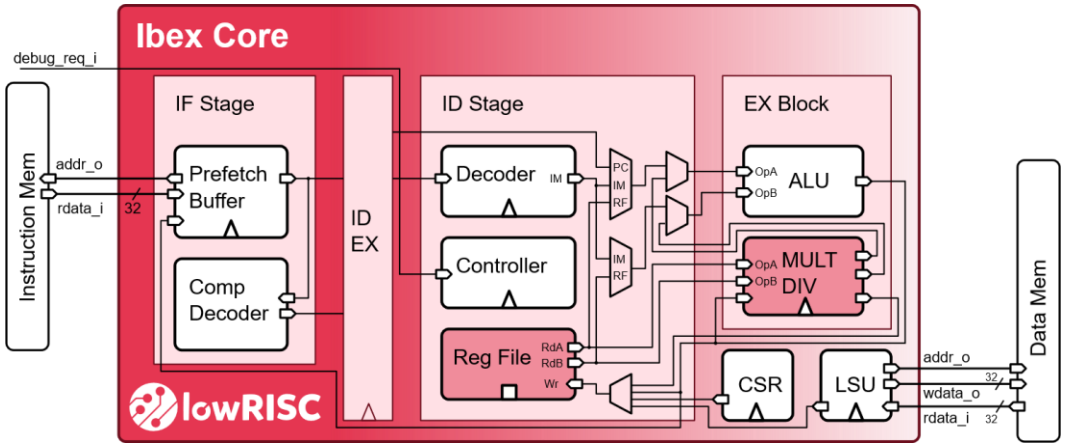
```
ADD(src1, src2, dest) = {
  t1 := Regfile[src1]
  t2 := Regfile[src2]
  leak(t1, old_src1)
  leak(t2, old_src2)
  old_src1 = t1
  old_src2 = t2;
  Regfile[dest] := t1 + t2;
}
```

Verified IBEX Contract

Contract

- Allows for simpler verification
- Exposes only necessary part of microarchitecture
- Reusable across processor implementations
  - Verify once per implementation

We wrote & verified contract for IBEX processor



Conclusions

- Software side channel security: a hardware problem
- We provide
  - Formal model & methodology
  - Power side-channel verifier for hardware (industrial use)
  - Power side-channel verifier for software
  - Hardened IBEX processor with contract
- Limitations
  - Small processor (but real: PULP, OpenTitan)
- TODO
  - Full ecosystem (compilers, verifiers, CPUs)
  - Other side channels (time,…)

Where do masked Inputs Come From?

Two examples.

AES: use a random key k

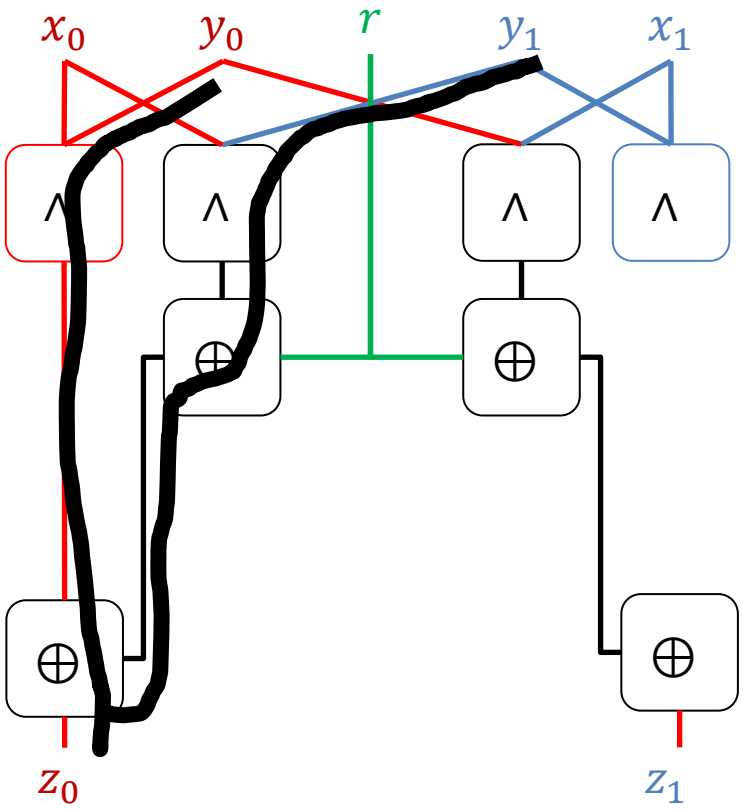
Masked: Create random numbers k_0 and k_1, define $k = k_0 \oplus k_1$. (But don't compute k!)

RSA: Choose two large random numbers p and q. Keep doing that until they are prime.

Masked: Create random numbers p_0, p_1, q_0, q_1, perform $maskedPrimalityCheck(p_0, p_1, q_0, q_1)$.

Glitches

$$z = x \wedge y \rightarrow \begin{cases} z_0 = r \oplus (x_0 \wedge y_0) \oplus (x_0 \wedge y_1) \\ z_1 = r \oplus (x_1 \wedge y_0) \oplus (x_1 \wedge y_1) \end{cases}$$



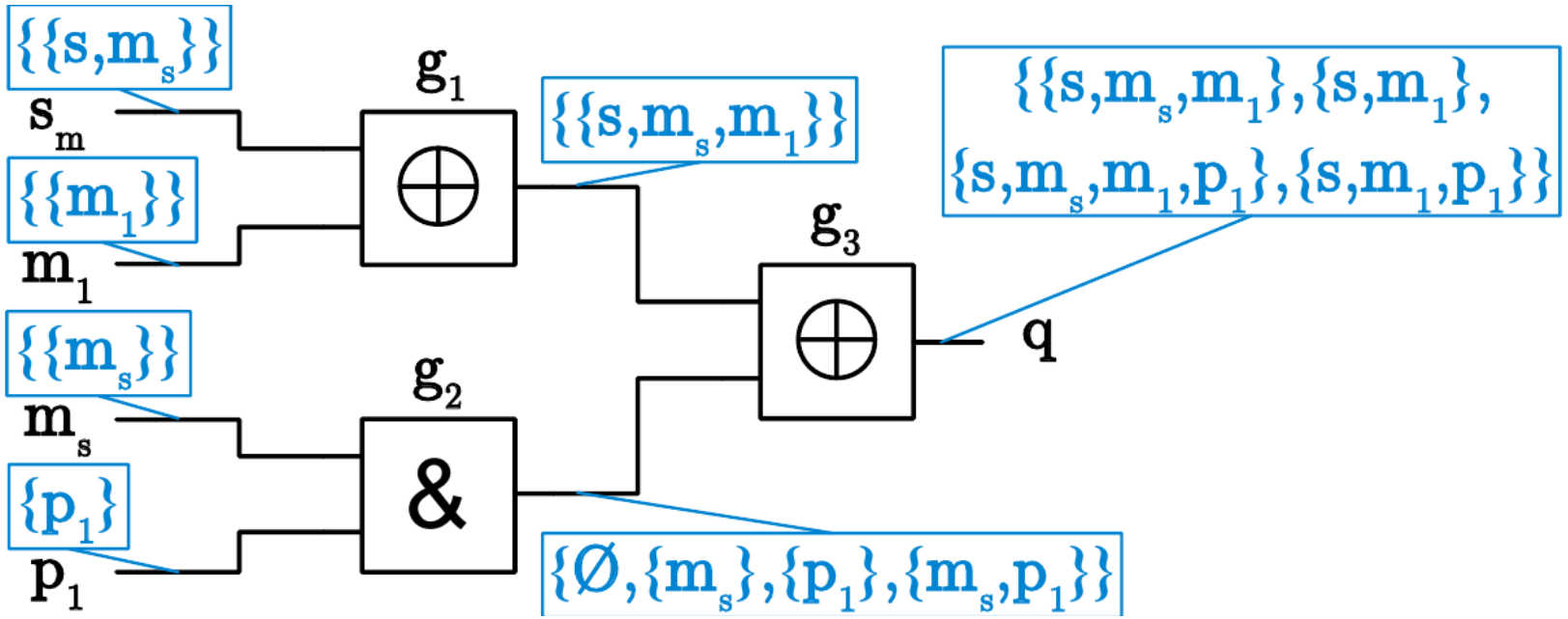
Heuristic

Let $L \star M = \{S \,\Delta\, T \mid S \in L, T \in M\}$

Gate	Rule
$h = s$ for $s \in \mathcal{S}$	$L(h) = \{s\}$
$h = f \oplus g.$	$L(h) = L(f) \star \mathrm{L}(g)$
$h = f \wedge g.$ (nonlinear gates)	$L(h) = \{\emptyset\} \cup L(f) \cup L(g) \cup L(f) \star L(g)\}$

- Vulnerability corresponds to “naked x”: $L(g) \cap X \neq \emptyset$ and $L(g) \cap M = \emptyset$
- Rules can be adapted for glitching

Example


{{s,m_s}}
s_m
{{m_1}}
m_1
{{m_s}}
m_s
{p_1}
p_1
g_1
g_2
&
{{s,m_s,m_1}}
{Ø,{m_s},{p_1},{m_s,p_1}}
g_3
{{s,m_s,m_1},{s,m_1},
{s,m_s,m_1,p_1},{s,m_1,p_1}}
q