Power Contracts:

Provably Complete Leakage Models for Processors

Roderick Bloem, Barbara Gigerl, Marc Gourjon, Vedad Hadžić, Stefan Mangard, Robert Primas

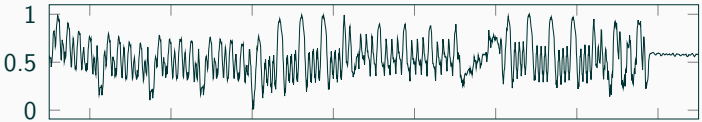
TU Graz

TUHH
Hamburg University of Technology

NXP

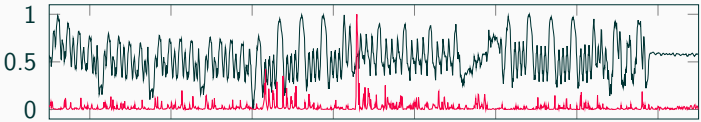
Motivation: Power Analysis Attacks

- Processors leak information about processed data through power side channels:



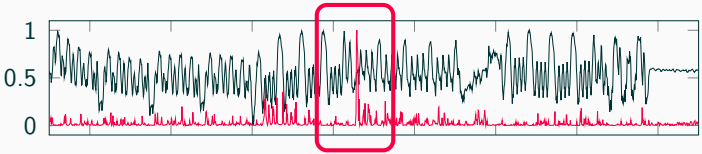
Motivation: Power Analysis Attacks

- Processors leak information about processed data through power side channels:



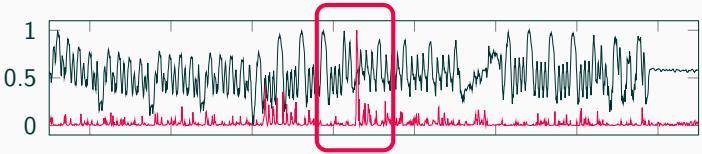
Motivation: Power Analysis Attacks

- Processors leak information about processed data through power side channels:



Motivation: Power Analysis Attacks

- Processors leak information about processed data through power side channels:



- Masking is a side-channel countermeasure
  - Splits secrets into uniformly random shares, e.g.,

$$s = s_1 \oplus s_2$$

Motivation: Masking Verification (Models)

(Option 1) Verify using some leakage model:

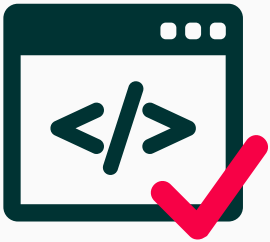
```
xor  x3, x1, x2 ; leaks x1 ⊕ x2
and  x7, x4, x3 ; leaks x4 & x3
```

Motivation: Masking Verification (Models)

(Option 1) Verify using some leakage model:

```
xor  x3, x1, x2 ; leaks x1 ⊕ x2
and  x7, x4, x3 ; leaks x4 & x3
```

👍 Very simple to verify



Motivation: Masking Verification (Models)

(Option 1) Verify using some leakage model:

```
xor  x3, x1, x2 ; leaks x1 ⊕ x2
and  x7, x4, x3 ; leaks x4 & x3
```

- Very simple to verify
- Most likely incomplete

Motivation: Masking Verification (Models)

(Option 1) Verify using some leakage model:

```
xor  x3, x1, x2 ; leaks x1 ⊕ x2
   ; ALU: x1 & x2, x1 + x2, ...
and  x7, x4, x3 ; leaks x4 & x3
```

- Very simple to verify
- Most likely incomplete ...

Motivation: Masking Verification (Models)

(Option 1) Verify using some leakage model:

```
xor  x3, x1, x2 ; leaks x1 ⊕ x2
   ; ALU: x1 & x2, x1 + x2, ...
   ; RF: x1, x2, WB: x3 ⊕ (x1 ⊕ x2)
and  x7, x4, x3 ; leaks x4 & x3
```

- Very simple to verify
- Most likely incomplete ...

Motivation: Masking Verification (Models)

(Option 1) Verify using some leakage model:

```
xor  x3, x1, x2 ; leaks x1 ⊕ x2
   ; ALU: x1 & x2, x1 + x2, ...
   ; RF: x1, x2, WB: x3 ⊕ (x1 ⊕ x2)
and  x7, x4, x3 ; leaks x4 & x3
   ; RF transition:  x4 ⊕ x1, x3 ⊕ x2
```

- Very simple to verify
- Most likely incomplete ...

Motivation: Masking Verification (Models)

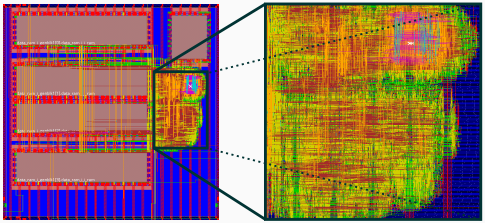
(Option 1) Verify using some leakage model:

```
xor  x3, x1, x2 ; leaks x1 ⊕ x2
   ; ALU: x1 & x2, x1 + x2, ...
   ; RF: x1, x2, WB: x3 ⊕ (x1 ⊕ x2)
and  x7, x4, x3 ; leaks x4 & x3
   ; RF transition:  x4 ⊕ x1, x3 ⊕ x2
   ; WB transition:  (x4 & x3) ⊕ (x1 ⊕ x2)
```

- 👍 Very simple to verify
- 👎 Most likely incomplete ...

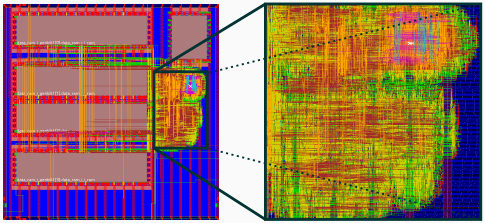
Motivation: Masking Verification (Netlist)

(Option 2) Verify considering CPU design:



Motivation: Masking Verification (Netlist)

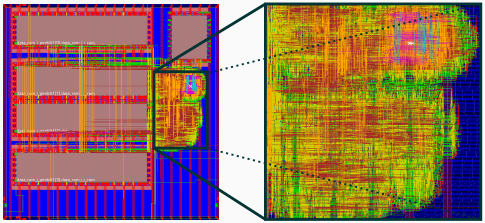
(Option 2) Verify considering CPU design:



Complete by definition

Motivation: Masking Verification (Netlist)

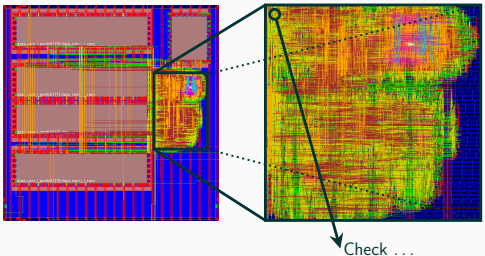
(Option 2) Verify considering CPU design:

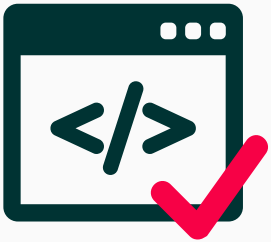


- Complete by definition
- Extremely cumbersome to verify

Motivation: Masking Verification (Netlist)

(Option 2) Verify considering CPU design:

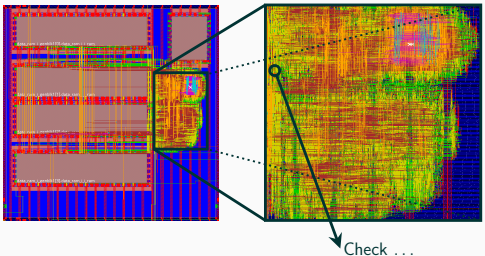


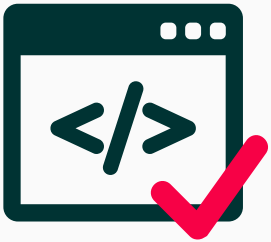


- Complete by definition
- Extremely cumbersome to verify ...

Motivation: Masking Verification (Netlist)

(Option 2) Verify considering CPU design:

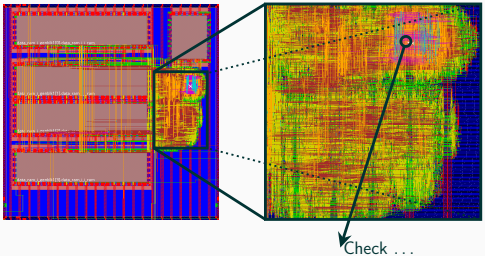


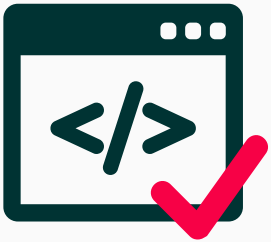


- Complete by definition
- Extremely cumbersome to verify ...

Motivation: Masking Verification (Netlist)

(Option 2) Verify considering CPU design:

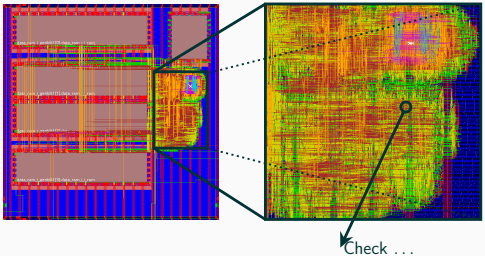


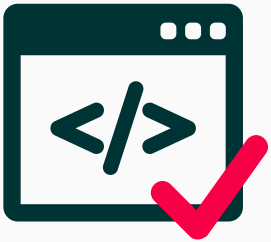


- Complete by definition
- Extremely cumbersome to verify ...

Motivation: Masking Verification (Netlist)

(Option 2) Verify considering CPU design:

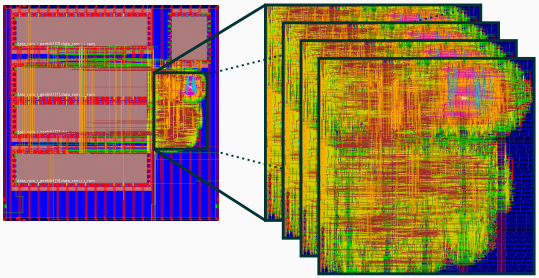


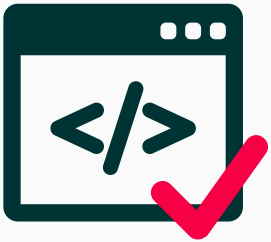


- Complete by definition
- Extremely cumbersome to verify ...

Motivation: Masking Verification (Netlist)

(Option 2) Verify considering CPU design:

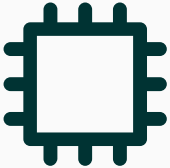




- Complete by definition
- Extremely cumbersome to verify

Solution: Power Contracts

program

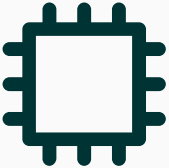


processor

Solution: Power Contracts

program

contract

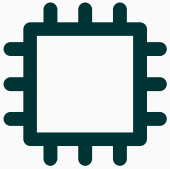


processor

Solution: Power Contracts

program

contract

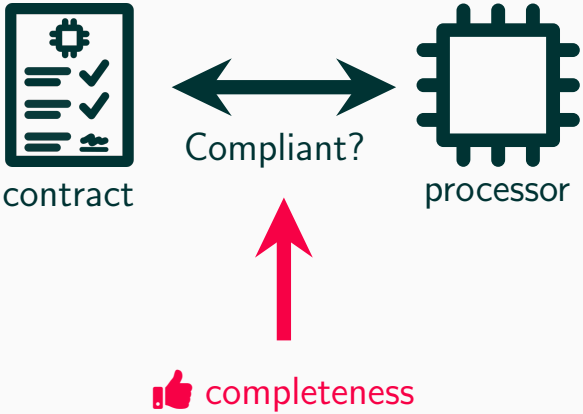


processor

abstraction layer

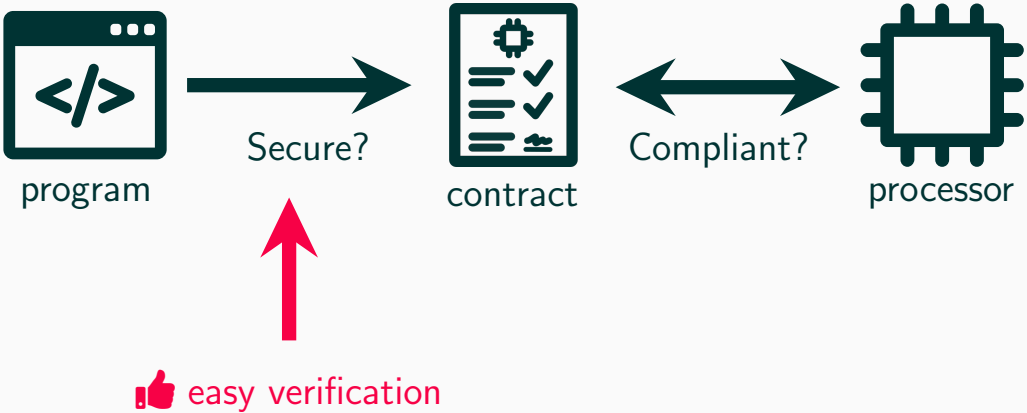
Solution: Power Contracts

Solution: Power Contracts



Solution: Power Contracts

Solution: Power Contracts

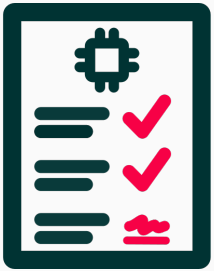


Solution: Power Contracts

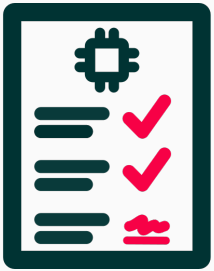
Solution: Power Contracts

What are Power Contracts?

Contracts formally describe a processor.



What are Power Contracts?

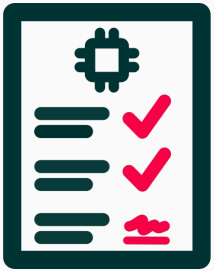


Contracts formally describe a processor.

- Functional (ISA) specification :
  - architectural registers, e.g., `pc`, `x1`, `x2`, ...
  - what instructions compute, e.g.,

`xor x3, x1, x2` $\Rightarrow$ `x3` $\leftarrow$ `x1` $\oplus$ `x2`; `pc` $\leftarrow$ `pc` $+$ `4`

What are Power Contracts?



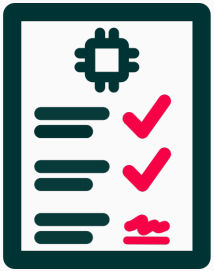
Contracts formally describe a processor.

- Functional (ISA) specification :
  - architectural registers, e.g., `pc`, `x1`, `x2`, ...
  - what instructions compute, e.g.,

$$\texttt{xor x3, x1, x2} \Rightarrow \begin{array}{l} \texttt{x3} \leftarrow \texttt{x1} \oplus \texttt{x2} \\ \texttt{pc} \leftarrow \texttt{pc} + 4 \end{array}$$

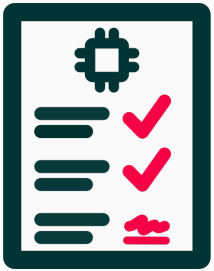
- Leakage specification:
  - microarchitectural elements, e.g., `rf_pA`
  - what is leaked, e.g., `leak(x1, x2, rf_pA)`

Power Contract Example



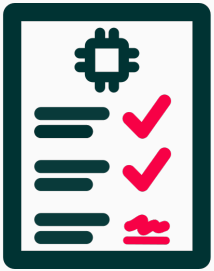
```
1  function clause execute (AND(rs2, rs1, rd)) = {
2    let rs1_val = X(rs1);              // load rs1
3    let rs2_val = X(rs2);              // load rs2
4    let result = rs1_val & rs2_val; // compute
5
6
7
8
9    X(rd) = result;                 // write result
10   pc = pc + 4;                    // update pc
11
12
13   return RETIRE_SUCCESS }
```

Power Contract Example



```
1  function clause execute (AND(rs2, rs1, rd)) = {
2    let rs1_val = X(rs1);                  // load rs1
3    let rs2_val = X(rs2);                  // load rs2
4    let result = rs1_val & rs2_val; // compute
5    leak(rs1_val, rs2_val);   // ALU leakage
6    leak(rs1_val, rf_pA);     // RF transition rs1
7    leak(rs2_val, rf_pB);     // RF transition rs2
8    leak(X(rd), result);      // WB transition
9    X(rd) = result;           // write result
10   pc = pc + 4;              // update pc
11
12
13   return RETIRE_SUCCESS }
```

Power Contract Example



```
1  function clause execute (AND(rs2, rs1, rd)) = {
2    let rs1_val = X(rs1);                  // load rs1
3    let rs2_val = X(rs2);                  // load rs2
4    let result = rs1_val & rs2_val; // compute
5    leak(rs1_val, rs2_val);  // ALU leakage
6    leak(rs1_val, rf_pA);    // RF transition rs1
7    leak(rs2_val, rf_pB);    // RF transition rs2
8    leak(X(rd), result);     // WB transition
9    X(rd) = result;          // write result
10   pc = pc + 4;             // update pc
11   rf_pA = rs1_val;         // update RF A port
12   rf_pB = rs2_val;         // update RF B port
13   return RETIRE_SUCCESS }
```

Contract Compliance

A processor complies with a contract if:

1. They are functionally equivalent*
2. For every gate-level hardware leak, there is a stronger contract leak

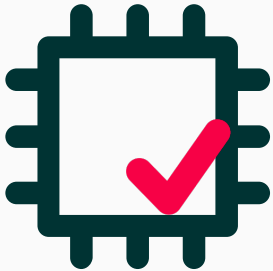
Contract Compliance

A processor complies with a contract if:

1. They are functionally equivalent*
2. For every gate-level hardware leak, there is a stronger contract leak

Example: `leak(x1, x2)` is stronger than

- `leak(x1)` and `leak(x2)`
- `leak(x1 + x2)` and `leak(x1 & x2)`



Contract Compliance

A processor complies with a contract if:

1. They are functionally equivalent*
2. For every gate-level hardware leak, there is a stronger contract leak

Example: `leak(x1, x2)` is stronger than

- `leak(x1)` and `leak(x2)`
- `leak(x1 + x2)` and `leak(x1 & x2)`
- any other function g(`x1, x2`)

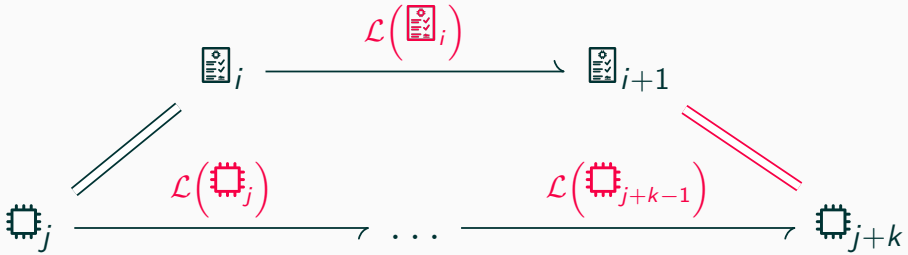
Compliance Verification Tasks

Compliance Verification Tasks

Compliance Verification Tasks

1. $\forall$ j, i : $j \simeq i \Rightarrow j+k \simeq i+1$

Compliance Verification Tasks



1. $\forall\ \square_j, \blacksquare_i : \square_j \simeq \blacksquare_i \Rightarrow \square_{j+k} \simeq \blacksquare_{i+1}$
2. $\exists\ f_\lambda : \forall\ \square_j, \blacksquare_i : \square_j \simeq \blacksquare_i \Rightarrow f_\lambda \circ \lambda\left(\blacksquare_i\right) = \lambda_g\left(\square_{l-1}, \square_l\right)$

Main Verification Trick

We need to show λ is stronger than λ_g:

$$\exists\ f_\lambda : \forall\ \square_j, \square_i :$$
$$\square_j \simeq \square_i \Rightarrow f_\lambda \circ \lambda\left(\square_i\right) = \lambda_g\left(\square_{l-1}, \square_l\right)$$

Main Verification Trick

We need to show λ is stronger than λ_g:

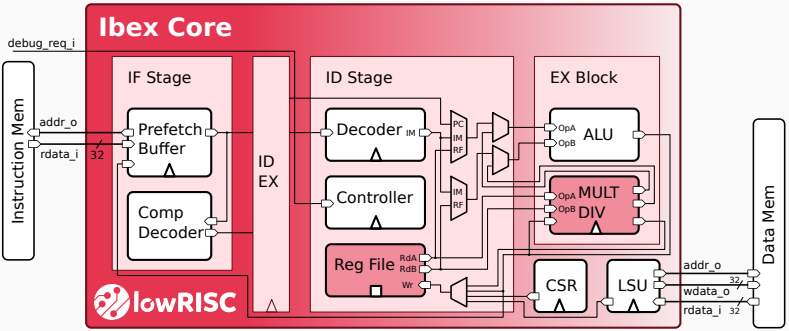
$$\exists\ f_\lambda : \forall\ \square_j, \blacksquare_i :$$
$$\square_j \simeq \blacksquare_i \Rightarrow f_\lambda \circ \lambda\left(\blacksquare_i\right) = \lambda_g\left(\square_{l-1}, \square_l\right)$$

We show it implicitly through congruence:

$$\forall\ \square_j, \square'_j, \blacksquare_i, \blacksquare'_i :$$
$$\square_j \simeq \blacksquare_i \wedge \square'_j \simeq \blacksquare'_i \wedge \lambda\left(\blacksquare_i\right) = \lambda\left(\blacksquare'_i\right) \Rightarrow$$
$$\lambda_g\left(\square_{l-1}, \square_l\right) = \lambda_g\left(\square'_{l-1}, \square'_l\right)$$

Case Study: Target

We have created a contract for IBEX:



Case Study: Leaks

Almost all instructions have the same leakage:

```
leak(rs1_val, rs2_val, rf_pA, rf_pB,
mem_last_addr, mem_last_read);
```

Case Study: Leaks

Almost all instructions have the same leakage:

```
leak(rs1_val, rs2_val, rf_pA, rf_pB,
mem_last_addr, mem_last_read);
```

- Cannot be broken down into multiple leaks
- Even if there is no `rs2` or `rs1`

Case Study: Leaks

Almost all instructions have the same leakage:

```
leak(rs1_val, rs2_val, rf_pA, rf_pB,
mem_last_addr, mem_last_read);
```

- Cannot be broken down into multiple leaks
- Even if there is no `rs2` or `rs1`

Additionally, the writeback stage leaks:

```
leak(x1_val, x1_nval);
leak(x2_val, x2_nval); //...
```

Case Study: Results

- Changed ∼1k lines of RISC-V spec
- Configuration requires ∼400 lines
- Compliance verification takes 30.6 hours

Case Study: Results

- Changed ∼1k lines of RISC-V spec
- Configuration requires ∼400 lines
- Compliance verification takes 30.6 hours

Gadget	# Instr.	# Clear.	Verification time	
			Contract	Netlist
`AND`	62	10	< 1 s ✓	284.63 s ✓
`Refresh`	19	0	< 1 s ✓	32.85 s ✓
`XOR`	16	1	< 1 s ✓	50.79 s ✓
`NOT`	5	0	< 1 s ✓	63.32 s ✓

Power Contracts:

Provably Complete Leakage Models for Processors

Roderick Bloem, Barbara Gigerl, Marc Gourjon, Vedad Hadžić, Stefan Mangard, Robert Primas

TU Graz

TUHH
Hamburg University of Technology

NXP