On the Difficulty of FSM-based Hardware Obfuscation

CHES 2018, September 10, 2018

Marc Fyrbiak[1], Sebastian Wallat[2], Jonathan Déchelotte[3], Nils Albartus[1], Sinan Böcker[1]
Russell Tessier[2], Christof Paar[1,2]

[1]Ruhr-Universität Bochum [2]University of Massachusetts Amherst [3]University of Bordeaux

Motivation

- IP cores transparent to numerous stakeholders
- Problem for IP owner(s): piracy

Motivation

- IP cores transparent to numerous stakeholders
- Problem for IP owner(s): piracy
- Solution: IP theft protection and obfuscation

Motivation

- IP cores transparent to numerous stakeholders
- Problem for IP owner(s): piracy
- Solution: IP theft protection and obfuscation
- State-of-the-Art: FSM obfuscation assumed to provide strong protection
  - *HARPOON*
  - **Dynamic State Deflection**
  - *Active Hardware Metering*
  - *Interlocking Obfuscation*

$$\begin{tikzcd} \arrow[r] & S_0 \arrow[loop above, "0"] \arrow[r, bend left, "1"] & S_1 \arrow[loop above, "0"] \arrow[l, bend left, "1"] \end{tikzcd}$$

Motivation

- IP cores transparent to numerous stakeholders

- Problem for IP owner(s): piracy
- Solution: IP theft protection and obfuscation

- State-of-the-Art: FSM obfuscation assumed to provide strong protection
  - *HARPOON*
  - **Dynamic State Deflection**
  - *Active Hardware Metering*
  - *Interlocking Obfuscation*

$$\begin{tikzcd}
{} \arrow[r] & S_0 \arrow[loop above, "0"] \arrow[rr, bend left, "1"] & & S_1 \arrow[loop above, "0"] \arrow[ll, bend left, "1"]
\end{tikzcd}$$

Our Research Question: How secure are these schemes?

Adversary Model

Assumptions:

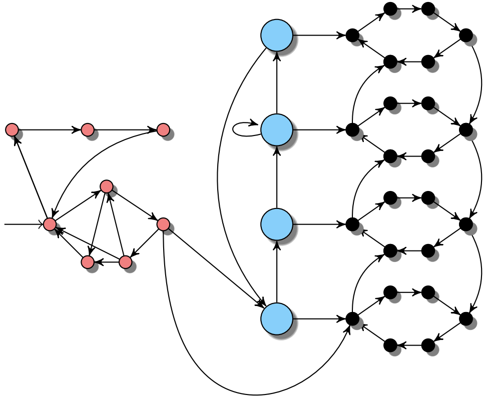
- Access to flattened gate-level netlist equipped with FSM obfuscation
- No information about module hierarchies, synthesis options, and names

Goal:

- Deobfuscate design to commit IP infringement

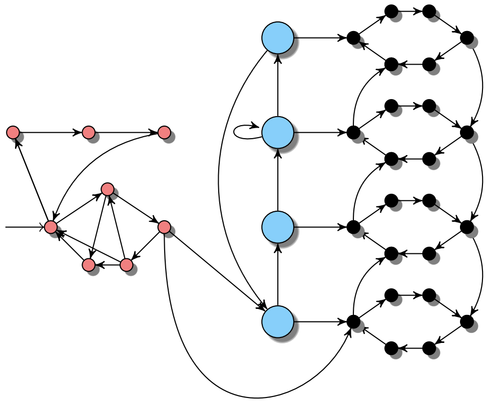
FSM Obfuscation - Dynamic State Deflection

- Obfuscation FSM and **blackhole FSM** are added to original FSM
- Enabling key only known to honest parties



FSM Obfuscation - Dynamic State Deflection

- **Obfuscation FSM** and **blackhole FSM** are added to original FSM
- Enabling key only known to honest parties

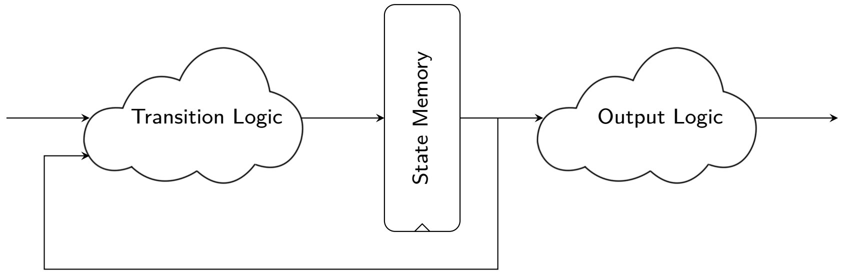


- How challenging is FSM reverse engineering and how secure is this scheme?

Agenda

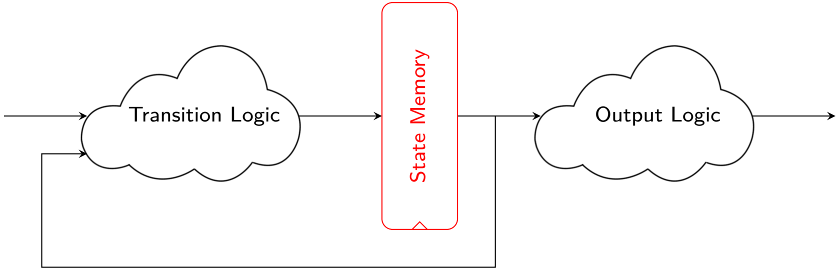
Problem I: Determine FSM Gates in Gate-level Netlist

- Ideas build upon previous work by Shi et al. and Meade et al.



Problem I: Determine FSM Gates in Gate-level Netlist

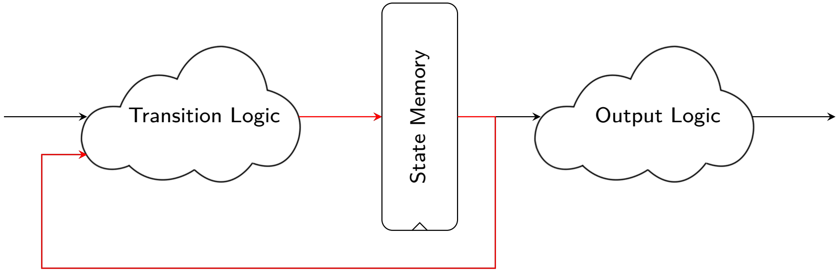
- Ideas build upon previous work by Shi et al. and Meade et al.
- FSM Property I: Register control signals



RUHR UNIVERSITÄT BOCHUM RUB

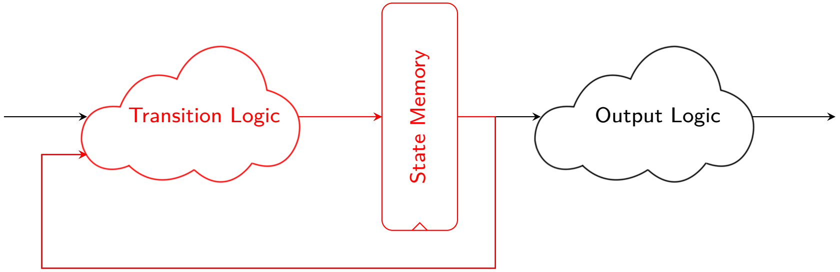
Problem I: Determine FSM Gates in Gate-level Netlist

- Ideas build upon previous work by Shi et al. and Meade et al.
- FSM Property II: Strongly connected component



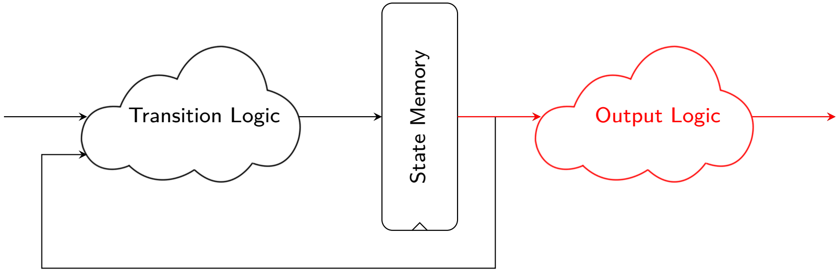
Problem I: Determine FSM Gates in Gate-level Netlist

- Ideas build upon previous work by Shi et al. and Meade et al.
- FSM Property III: Combinational logic feedback path



Problem I: Determine FSM Gates in Gate-level Netlist

- Ideas build upon previous work by Shi et al. and Meade et al.
- FSM Property IV: Control behavior



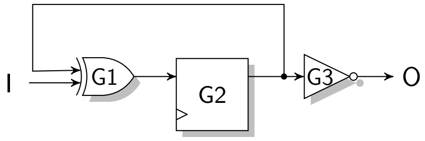
Problem II: Determine State Transition Graph from FSM Gates

- Ideas build upon previous work by Shi et al. and Meade et al.

```
module FSM (CLK, I, O);
    input CLK, I;
    output O;
    wire o_G1, o_G2;
    XOR G1 (
        .IN1(o_G2), .IN2(I), .O(o_G1));
    DFF G2 (
        .CLK(CLK), .D(o_G1), .Q(o_G2));
    INV G3 (
        .IN(o_G2), .O(O));
endmodule
```

Problem II: Determine State Transition Graph from FSM Gates

- Ideas build upon previous work by Shi et al. and Meade et al.

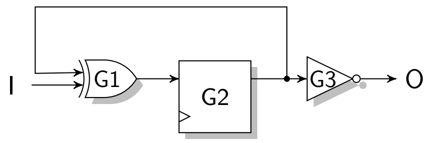


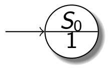
```
module FSM (CLK, I, O);
    input CLK, I;
    output O;
    wire o_G1, o_G2;
    XOR G1 (
        .IN1(o_G2), .IN2(I), .O(o_G1));
    DFF G2 (
        .CLK(CLK), .D(o_G1), .Q(o_G2));
    INV G3 (
        .IN(o_G2), .O(O));
endmodule
```

RUHR UNIVERSITÄT BOCHUM RUB

Problem II: Determine State Transition Graph from FSM Gates

- Ideas build upon previous work by Shi et al. and Meade et al.

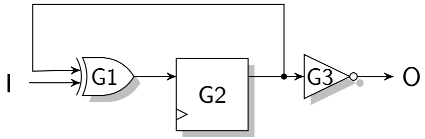


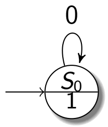


```
module FSM (CLK, I, O);
    input CLK, I;
    output O;
    wire o_G1, o_G2;
    XOR G1 (
        .IN1(o_G2), .IN2(I), .O(o_G1));
    DFF G2 (
        .CLK(CLK), .D(o_G1), .Q(o_G2));
    INV G3 (
        .IN(o_G2), .O(O));
endmodule
```

Problem II: Determine State Transition Graph from FSM Gates

- Ideas build upon previous work by Shi et al. and Meade et al.

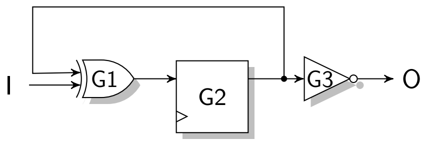


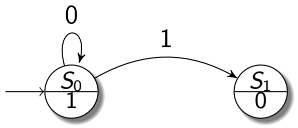


```
module FSM (CLK, I, O);
    input CLK, I;
    output O;
    wire o_G1, o_G2;
    XOR G1 (
        .IN1(o_G2), .IN2(I), .O(o_G1));
    DFF G2 (
        .CLK(CLK), .D(o_G1), .Q(o_G2));
    INV G3 (
        .IN(o_G2), .O(O));
endmodule
```

Problem II: Determine State Transition Graph from FSM Gates

- Ideas build upon previous work by Shi et al. and Meade et al.

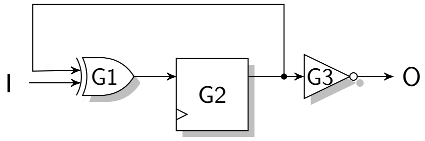


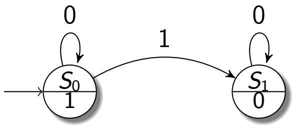


```
module FSM (CLK, I, O);
    input CLK, I;
    output O;
    wire o_G1, o_G2;
    XOR G1 (
        .IN1(o_G2), .IN2(I), .O(o_G1));
    DFF G2 (
        .CLK(CLK), .D(o_G1), .Q(o_G2));
    INV G3 (
        .IN(o_G2), .O(O));
endmodule
```

Problem II: Determine State Transition Graph from FSM Gates

- Ideas build upon previous work by Shi et al. and Meade et al.

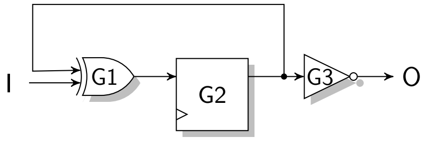


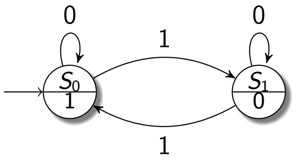


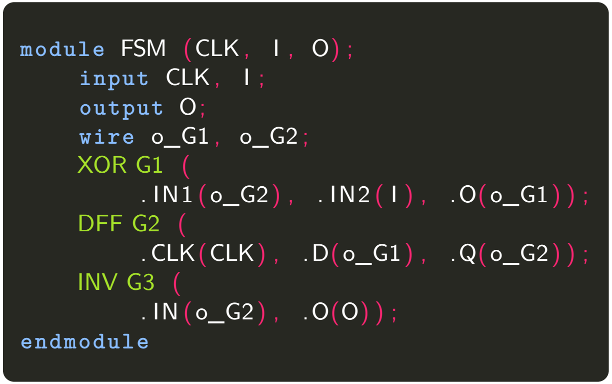
```
module FSM (CLK, I, O);
    input CLK, I;
    output O;
    wire o_G1, o_G2;
    XOR G1 (
        .IN1(o_G2), .IN2(I), .O(o_G1));
    DFF G2 (
        .CLK(CLK), .D(o_G1), .Q(o_G2));
    INV G3 (
        .IN(o_G2), .O(O));
endmodule
```

Problem II: Determine State Transition Graph from FSM Gates

- Ideas build upon previous work by Shi et al. and Meade et al.

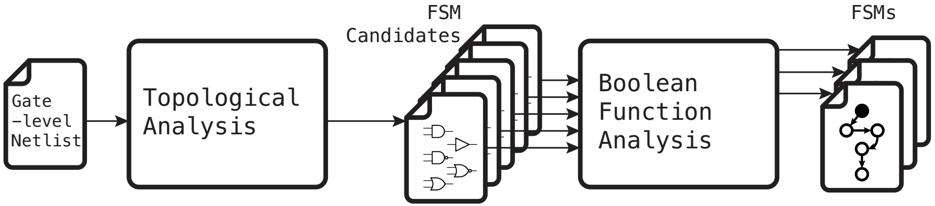




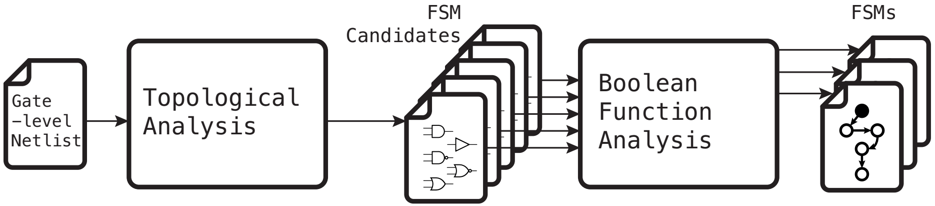


```
module FSM (CLK, I, O);
    input CLK, I;
    output O;
    wire o_G1, o_G2;
    XOR G1 (
        .IN1(o_G2), .IN2(I), .O(o_G1));
    DFF G2 (
        .CLK(CLK), .D(o_G1), .Q(o_G2));
    INV G3 (
        .IN(o_G2), .O(O));
endmodule
```

Automated FSM Reverse Engineering



Automated FSM Reverse Engineering



- Boolean Function Analysis: $\mathcal{O}(|S| \cdot 2^{|I|})$
  - $|S|$ = Number of FSM states $\quad |I|$ = Number of FSM inputs

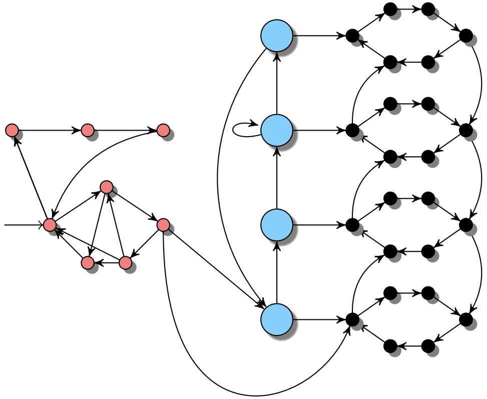
Agenda

Reminder: Dynamic State Deflection

- **Obfuscation FSM** and **blackhole FSM** are added to **original FSM**

- Enabling key only known to honest parties

- Case Study: AES + DSD
  - 12-bit enabling key
  - 14 obfuscation states
  - 5 blackhole states per original one



Case Study: AES + DSD (Topological Analysis)

- Candidate: 8 FFs, 21 inputs, influence/dependence 0.625

```
-------------------------------------------------------------
  [+] FF1 influences and depends on: FF1, FF2, FF3, FF4, FF5, FF6
  [+] FF2 influences and depends on: FF1, FF2, FF3, FF4, FF5, FF6
  [+] FF3 influences and depends on: FF1, FF2, FF3, FF4, FF5, FF6
  [+] FF4 influences and depends on: FF1, FF2, FF3, FF4, FF5, FF6
  [+] FF5 influences and depends on: FF1, FF2, FF3, FF4, FF5, FF6
  [+] FF6 influences and depends on: FF1, FF2, FF3, FF4, FF5, FF6
  [+] FF7 influences and depends on: FF7, FF8
  [+] FF8 influences and depends on: FF7, FF8
-------------------------------------------------------------
```

Case Study: AES + DSD (Topological Analysis)

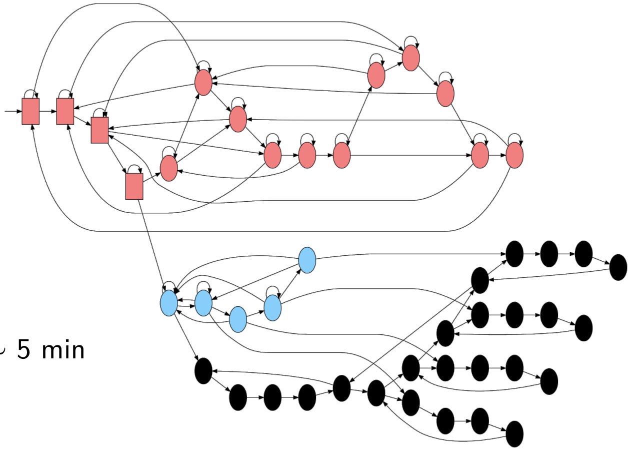
- Candidate: 8 FFs, 21 inputs, influence/dependence 0.625

```
---------------------------------------------------------------------
  [+] FF1 influences and depends on: FF1, FF2, FF3, FF4, FF5, FF6
  [+] FF2 influences and depends on: FF1, FF2, FF3, FF4, FF5, FF6
  [+] FF3 influences and depends on: FF1, FF2, FF3, FF4, FF5, FF6
  [+] FF4 influences and depends on: FF1, FF2, FF3, FF4, FF5, FF6
  [+] FF5 influences and depends on: FF1, FF2, FF3, FF4, FF5, FF6
  [+] FF6 influences and depends on: FF1, FF2, FF3, FF4, FF5, FF6
  [+] FF7 influences and depends on: FF7, FF8
  [+] FF8 influences and depends on: FF7, FF8
---------------------------------------------------------------------
```

Case Study: AES + DSD (Boolean Function Analysis)

- Obfuscation FSM
- Original AES FSM
- Blackhole FSM

Computational Complexity:
2^{22} steps (6 FFs and 16 inputs) $\sim$ 5 min



Summary of Results

- *HARPOON*
  - Disclosure of enabling key
  - Initial state patching
  - Watermark manipulation

- *Active Hardware Metering*
  - Initial state patching
  - Enabling key disclosure

- *Dynamic State Deflection*
  - Disclosure of enabling key
  - State transition function patching

- *Interlocking Obfuscation*
  - Initial state patching
  - Design tampering

Lessons Learned

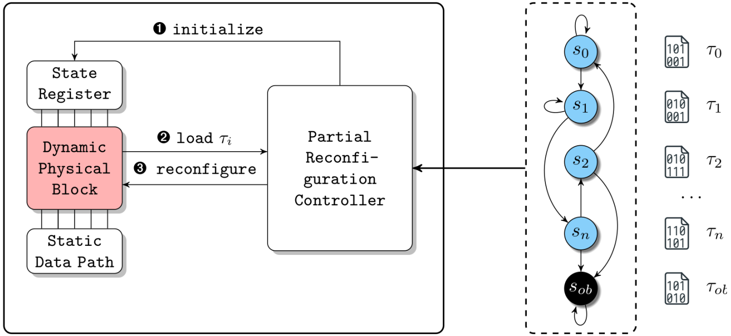
- Topological analysis yields FSM gates
- Separation of obfuscation vs original parts
- Complexity of Boolean function analysis

RUHR UNIVERSITÄT BOCHUM RUB

Agenda

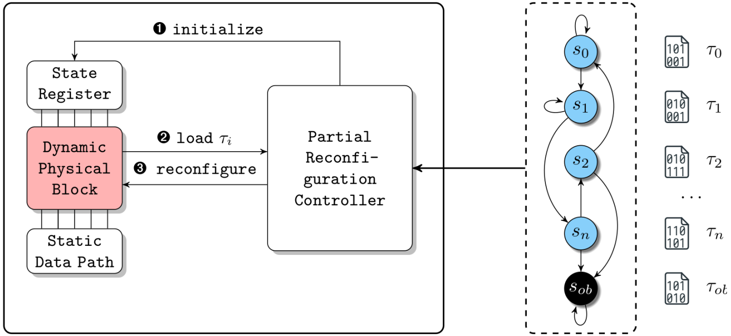
Hardware Nanomites - FSM Obfuscation for FPGAs

- **Idea:** prevent (static) topological analysis via reconfiguration



Hardware Nanomites - FSM Obfuscation for FPGAs

- **Idea:** prevent (static) topological analysis via reconfiguration



- Partial reconfiguration yields: **self-modifying hardware** and **anti-simulation**

Hardware Nanomites - FSM Obfuscation for FPGAs

- **Static Design**

Component	#LUTs (Logic)	#FFs	#LUTs (Memory)
Microblaze	1553 (0.64 %)	1401 (0.29 %)	198 (0.18 %)
DDR Controller	15151 (6.25 %)	17520 (3.61 %)	1379 (1.22 %)
HWICAP	312 (0.13 %)	959 (0.20 %)	1 ($\geq$ 0.01 %)
AXI SmartConnect	5827 (2.40 %)	8977 (1.85 %)	2017 (1.79 %)
Misc. Parts (UART, ...)	1335 (0.55 %)	1752 (0.36 %)	94 (0.08 %)
Complete Static Design	24178 (9.97 %)	30609 (6.31 %)	3689 (3.27 %)

- **Dynamic Physical Block**

#LUTs (Logic)	#FFs	#LUTs (Memory)	Partial Bitstream Size
160 (0.07 %)	320 (0.07 %)	80 (0.07 %)	352 kByte

Conclusion

- We demonstrated several generic, semi-automated strategies on state-of-the-art FSM obfuscation schemes to bypass their protection

- We proposed a novel FSM obfuscation primitive for FPGAs

Thanks for your attention! Any questions?