Stealthy Opaque Predicates in Hardware - Obfuscating Constant Expressions at Negligible Overhead

Max Hoffmann, Christof Paar

Ruhr University Bochum, Horst-Görtz Institute for IT-Security, Germany
Why Obfuscation?

High-level Description

Finished Product

“easy”

“not that easy”
Why Obfuscation?

- aes.c
- aes.vhd

High-level Description

Finished Product

“easy”

“not that easy”
Why Obfuscation?

- aes.c
- aes.vhd

High-level Description

“easy”

Finished Product

“not that easy”

01010100101
01000100101
01110101010
01101010010

01101010010
Why Obfuscation?

Obfuscation

High-level Description

“easy”

Finished Product

“insanely difficult”
Software Obfuscation

- One target in software is control flow obfuscation.
Software Obfuscation

• One target in software is control flow obfuscation.
Software Obfuscation

- Opaque Predicates are used as a basic building block.
Software Obfuscation

• Opaque Predicates are used as a basic building block.

• An opaque predicate:
  – is an expression
  – looks like having a dynamic value
  – evaluates to a constant, known value

Example:
\[(x \times (x + 1)) \mod 2 = 0\]
Software Obfuscation

• Opaque Predicates are used as a basic building block.

• An opaque predicate:
  – is an expression
  – looks like having a dynamic value
  – evaluates to a constant, known value

• Meant to harden against static analysis.

Example:
\[(x * (x + 1)) \% 2 == 0\]

- **Static Analysis**: analysis performed solely on a static data, e.g., a binary.
- **Dynamic Analysis**: analysis performed during operation, e.g., while executing a binary.
Example: Software Opaque Predicates

- Control flow graph of a static analyzer:

```python
if ((x * (x + 1)) % 2 == 0):
    foo()
else
    bar()
```

- “True” control flow graph:
A Software Obfuscation Technique in Hardware?

- How can a software obfuscation technique help in hardware?
- Obfuscation should harden against reverse engineering.
A Software Obfuscation Technique in Hardware?

• How can a software obfuscation technique help in hardware?

• Obfuscation should harden against reverse engineering.

• Reverse engineers rarely analyze an entire design.

• Mostly: small parts of a design.
A Software Obfuscation Technique in Hardware?

- How can a software obfuscation technique help in hardware?

- Obfuscation should harden against reverse engineering.

- Reverse engineers rarely analyze an entire design.

- Mostly: small parts of a design.

- **Goal**: hide as much information as possible.
  - reduces starting points for reverse engineers.
  - makes understanding of any component harder.
Example: Hardware Reversing

if a = "0110" then
  output <= '1';
end if;

V.S.

if a = b then
  output <= '1';
end if;
Example: Hardware Reversing

Use OPs to hide information introduced by constant signals.

if a = "0110" then
  output <= ‘1’;
end if;

vs.

if a = b then
  output <= ‘1’;
end if;

→ Use OPs to hide information introduced by constant signals.
Previous Work
• Only one prior work on opaque predicates.

• Sergeichik et al. presented LFSR-based OPs in 2014 [1].

Stealthiness

• **Problem**: Easy to detect, uncommon structure

• Removal via static analysis demonstrated in [1].

Stealthiness

- **Problem**: Easy to detect, uncommon structure

- Removal via static analysis demonstrated in [1].

- **Desired Metric**: “Stealthiness”
  - Impossible (?) to measure
  - Human factor plays a role
  - Different in hardware and software

---

Opaque Predicates in Hardware
Hardware OPs – Idea

• Stealthiness: use common structures.

• Try to use existing circuitry.
Hardware OPs – Idea

- Stealthiness: use common structures.

- Try to use existing circuitry.

- **Observation:**
  - Signals are changing constantly.
  - A signal’s value is only important while evaluated.
Stealthy Opaque Predicates in Hardware

- Stealthiness: use common structures.
- Try to use existing circuitry.
- **Observation:**
  - Signals are changing constantly.
  - A signal’s value is only important while evaluated.
  
  → Use an existing signal which
    1. has the required state whenever we need it
    2. switches “randomly” when not needed.
Example: Hardware OPs
Example: Hardware OPs

- Constant value required in Work1, Work2, and Work3.

- Multiple options to use the state of an FSM as an OP.
• Constant value required in Work1, Work2, and Work3.

• Multiple options to use the state of an FSM as an OP.
Example: Hardware OPs

- Constant value required in Work1, Work2, and Work3.

- Multiple options to use the state of an FSM as an OP.
• Example:
  – Constant $1101000_2$ to be obfuscated.
  – 5-bit FSM passes 3 states during the processing period.
Hardware OPs

- 1\textsuperscript{st} State:
Hardware OPs

• 2\textsuperscript{nd} State:
Hardware OPs

- 3rd State:
Hardware OPs

- 4th State:

![Transition Logic Diagram]
Hardware OPs

- Very stealthy: existing FSMs are used.
- Zero additional gates (in theory...)

Stealthy Opaque Predicates in Hardware | CHES 2018 | 10.09.2018
Hardware OPs

• Very stealthy: existing FSMs are used.

• Zero additional gates (in theory...)

• Applicable to nearly all designs.

• Considerably increases reversing effort: Reversing of control- and data-path required for identification of constants.
Hardware OPs

• Very stealthy: existing FSMs are used.

• Zero additional gates (in theory...)

• Applicable to nearly all designs.

• Considerably increases reversing effort: Reversing of control- and data-path required for identification of constants.

• Applicable to ASICs and FPGAs.

• Forces a reverse engineer to apply dynamic analysis.
Hardware OPs

- If no suitable FSM available, add a new FSM-like module.
  - Make it reset outside of the processing period.
  - Make it stabilize in a known state after some cycles.
  - Generate OP value from stable state.

- Still stealthy (FSMs are common).

- Stabilizing FSMs are also common (DONE state).
CASE STUDIES
Scenario
Algorithm 1 Subverted RSA KeyGen

**Input:** $1^\lambda$

**Output:** $pk = (n, e)$, $sk = (d)$

1: Choose $p, q$ as random $\lambda/2$-bit primes
2: $n \leftarrow pq$
3: $e \leftarrow p^{E_{adv}} \mod N_{adv}$
4: while $\gcd(e, \Phi(n)) \neq 1$ do
5: \hspace{1cm} $e \leftarrow e + 1$
6: $d \leftarrow e^{-1} \mod \Phi(n)$
7: \textbf{return} $pk = (n, e)$, $sk = (d)$
Results

<table>
<thead>
<tr>
<th>Design</th>
<th>LUTs</th>
<th>FFs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unobfuscated</td>
<td>304</td>
<td>347</td>
</tr>
<tr>
<td>PRESENT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strategy 1</td>
<td>307</td>
<td>347</td>
</tr>
<tr>
<td>Strategy 2</td>
<td>304</td>
<td>350</td>
</tr>
<tr>
<td>RSA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strategy 1</td>
<td>10811</td>
<td>5314</td>
</tr>
<tr>
<td>Strategy 2</td>
<td>10692</td>
<td>5323</td>
</tr>
</tbody>
</table>

**Platform:** XILINX Artix-7 35T FPGA

**Legend:**
- Unobfuscated: no opaque predicates were used
- Strategy 1: opaque predicate from existing circuitry
- Strategy 2: new circuitry for the opaque predicate
APPLICATION: WATERMARKING
Watermarking

• A watermark enables identification of IP-theft.

• A vendor can inspect products for presence of his watermark.

• Schmid et al. proposed a watermarking scheme for FPGAs which implements a watermark into LUT configurations [1].

FPGA LUT Configuration

• A LUT is configured by defining it's output values.

• Example:

| $I_3$ | 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 |
| $I_2$ | 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 |
| $I_1$ | 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 |
| $I_0$ | 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 |

| output/config | 1 0 1 1 1 0 1 0 0 0 0 1 1 0 1 0 |

• These configurations can be read from the bitstream of an FPGA.
**Watermarking by Schmid et al.**

- **Idea:** fix some inputs to GND.

<table>
<thead>
<tr>
<th>GND $\rightarrow$ $I_3$</th>
<th>0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND $\rightarrow$ $I_2$</td>
<td>0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>$I_1$</td>
<td>0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>$I_0$</td>
<td>0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1</td>
</tr>
</tbody>
</table>
Watermarking by Schmid et al.

- **Idea:** fix some inputs to GND.

- Configuration bits for other cases become effectively unused.

\[
\begin{array}{l|cccccccccccccccc}
\text{GND} & \rightarrow & I_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\text{GND} & \rightarrow & I_2 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
I_1 & & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
I_0 & & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{array}
\]
Watermarking by Schmid et al.

- **Idea**: fix some inputs to GND.
- Configuration bits for other cases become effectively unused.
- Embed watermark there.

| GND → $I_3$ | 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 |
| GND → $I_2$ | 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 |
| $I_1$      | 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 |
| $I_0$      | 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 |
| **output/config** | C C C C W W W W W W W W W W W W |
Applying OPs

- Netlist-level attacker was included in attacker model.

- **Problem**: Tracing GND to LUTs $\rightarrow$ detected $\rightarrow$ easy to remove the watermark.
**Applying OPs**

- Netlist-level attacker was included in attacker model.

- **Problem:** Tracing GND to LUTs $\rightarrow$ detected $\rightarrow$ easy to remove the watermark.

- **Solution:** Use our OPs instead of GND.

<table>
<thead>
<tr>
<th>OP $\rightarrow I_3$</th>
<th>0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP $\rightarrow I_2$</td>
<td>0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1</td>
</tr>
<tr>
<td>$I_1$</td>
<td>0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>$I_0$</td>
<td>0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1</td>
</tr>
<tr>
<td>output/config</td>
<td>C C C C W W W W W W W W W W W W</td>
</tr>
</tbody>
</table>
CONCLUSION
Conclusion

• Novel technique for opaque predicates in hardware (ASICs + FPGAs).

• Strong technique (discussion in the paper).

• Instantiation strategies:
  – Existing circuitry.
  – Additional circuitry.

• Practical evaluation.

• Demonstrate potential to mitigate existing attacks.
Thank You For Your Attention!
Any Questions?