Runtime Code Polymorphism as a Protection against Physical Attacks
Damien Couroussé, Bruno Robisson, Thierno Barry, P Jaillon, Olivier Potin

To cite this version:
Core Idea: Runtime Code Polymorphism

Definition
Regularly changing the behaviour of a (secured) component, at runtime, while maintaining unchanged its functional properties.

What for?
- Protection against reverse engineering of SW: the secured code is not available before runtime.
- Protection against physical attacks: polymorphism changes the spatial and temporal properties of the secured code: side channel & fault attacks can be combated with usual SW protections against focused attacks.

How?
- deGoal: runtime code generation for embedded systems.
  - fast code generation
  - tiny memory footprint: proof of concept on TI’s MSP430 (512 bytes of RAM)

Example: polymorphic AES

Polymorphic implementation of the SubBytes function:

```c
void gen_subBytes( cdg_insn_t* code, uint8_t* state_addr, uint8_t* sbox_addr)
{
    # Begin code Prelude
    inst uint32 state, sbox, i, x, y
    inst state_addr, sbox_addr
    inst i, #0
    loop:
        lb x, #state[i] // x := state[i]
        lb y, #sbox[x] // y := sbox[x]
        add i, i, #1
        bneq loop, i, #16)
    };
    # End
}
```

Unprotected

Protected with code polymorphism

~100 EM traces of AES SubBytes

Execution times (in cycles), over 1000 runs:

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>max</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
<td>6385</td>
<td>6385</td>
<td>6385</td>
</tr>
<tr>
<td>code generator</td>
<td>5671</td>
<td>12910</td>
<td>9345</td>
</tr>
<tr>
<td>polymorphic instance</td>
<td>7185</td>
<td>9745</td>
<td>8303</td>
</tr>
</tbody>
</table>

Impact of the code generation interval ω:

<table>
<thead>
<tr>
<th></th>
<th>k</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.76</td>
<td>53.0%</td>
</tr>
<tr>
<td>5</td>
<td>1.59</td>
<td>18.4%</td>
</tr>
<tr>
<td>20</td>
<td>1.37</td>
<td>2.1%</td>
</tr>
<tr>
<td>100</td>
<td>1.31</td>
<td>1.1%</td>
</tr>
</tbody>
</table>

k: overhead vs. reference implementation

%: percentage contribution of runtime code generation to the performance overhead

References


Instruction scheduling for VLIW processors: