RISC-V Extension for Lightweight Cryptography, Protection against SCA

E. Tehrani, T. Graba, A. Si Merabet and J.L. Danger
Outline

Context

RISC-V Extension

Protection against SCA

Conclusion
Outline

Context

RISC-V Extension

Protection against SCA

Conclusion
Context

- Requirement of **Agility** for **Lightweight Cryptography** (LWC) to suit any emerging application or standard.

- The **RISC-V ISA** can be extended to support execution of new instructions for LWC.

- The execution of LWC instructions can leak the key by side channels, **protections** are required.
Lightweight Block Ciphers

Considered Lightweight Block Ciphers:

- 128-bit key
- 64-bit blocks

They all share these 3 steps at each round:

- **Key Addition**
  - Adding the secret to the message
- **Confusion**
  - Non linear function provided generally by S-Box
- **Diffusion**
  - linear operation
### Classification of LWC Block Ciphers

<table>
<thead>
<tr>
<th>Cat.</th>
<th>Cipher</th>
<th>Confusion</th>
<th>Diffusion Type</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Simon</td>
<td>AND</td>
<td>Rot.</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>Simeck</td>
<td>AND</td>
<td>Rot.</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>Speck</td>
<td>ADD</td>
<td>Rot.</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>RC5</td>
<td>ADD</td>
<td>Rot.</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>XTea</td>
<td>ADD</td>
<td>Rot.</td>
<td>Bit</td>
</tr>
<tr>
<td>II</td>
<td>GOST</td>
<td>S-Box</td>
<td>Rot.</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>Rectangle</td>
<td>S-Box</td>
<td>Rot.</td>
<td>Bit</td>
</tr>
<tr>
<td>III a</td>
<td>PRESENT</td>
<td>S-Box</td>
<td>Perm.</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>GIFT</td>
<td>S-Box</td>
<td>Perm.</td>
<td>Bit</td>
</tr>
<tr>
<td>III b</td>
<td>PRINCE</td>
<td>S-Box x2</td>
<td>MatMult x2</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>Twine</td>
<td>S-Box</td>
<td>MatMult</td>
<td>Nib.</td>
</tr>
<tr>
<td></td>
<td>Skinny</td>
<td>S-Box</td>
<td>MatMult</td>
<td>Nib.</td>
</tr>
<tr>
<td></td>
<td>Midori</td>
<td>S-Box</td>
<td>MatMult</td>
<td>Nib.</td>
</tr>
<tr>
<td></td>
<td>MANTIS</td>
<td>S-Box x2</td>
<td>MatMult x2</td>
<td>Nib.</td>
</tr>
</tbody>
</table>

**Table:** Categorisation of Lightweight Block Ciphers
## Classification of LWC Block Ciphers

<table>
<thead>
<tr>
<th>Cat.</th>
<th>Cipher</th>
<th>Confusion</th>
<th>Diffusion Type</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Simon</td>
<td>AND</td>
<td>Rot.</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>Simeck</td>
<td>AND</td>
<td>Rot.</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>Speck</td>
<td>ADD</td>
<td>Rot.</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>RC5</td>
<td>ADD</td>
<td>Rot.</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>XTea</td>
<td>ADD</td>
<td>Rot.</td>
<td>Bit</td>
</tr>
<tr>
<td>II</td>
<td>GOST</td>
<td>S-Box</td>
<td>Rot.</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>Rectangle</td>
<td>S-Box</td>
<td>Rot.</td>
<td>Bit</td>
</tr>
<tr>
<td>III a</td>
<td>PRESENT</td>
<td>S-Box</td>
<td>Perm.</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>GIFT</td>
<td>S-Box</td>
<td>Perm.</td>
<td>Bit</td>
</tr>
<tr>
<td>III b</td>
<td>PRINCE</td>
<td>S-Box x2</td>
<td>MatMult x2</td>
<td>Bit</td>
</tr>
<tr>
<td>IV</td>
<td>Twine</td>
<td>S-Box</td>
<td>MatMult</td>
<td>Nib.</td>
</tr>
<tr>
<td></td>
<td>Skinny</td>
<td>S-Box</td>
<td>MatMult</td>
<td>Nib.</td>
</tr>
<tr>
<td></td>
<td>Midori</td>
<td>S-Box</td>
<td>MatMult</td>
<td>Nib.</td>
</tr>
<tr>
<td></td>
<td>MANTIS</td>
<td>S-Box x2</td>
<td>MatMult x2</td>
<td>Nib.</td>
</tr>
</tbody>
</table>

**Table:** Categorisation of Lightweight Block Ciphers
Additional Hardware Instructions

Specific instructions (64-bit data) which could accelerate LWC execution:

- **SBOX**
  - Non-linear function (Look-Up Table)
  - 16 identical 4x4 S-Boxes, requires configuration

- **PRESENT_D**
  - Cipher specific bit-level permutation
  - Costless in terms of hardware

- **GIFT_D**
  - Cipher specific bit-level permutation
  - Costless in terms of hardware

- **PRINCE_D**
  - Cipher specific bit-level matrix multiplication
  - A set of 2 inverse functions

- **NMAT_D**
  - Generic Nibble-level matrix multiplication
  - Requires configuration
Outline

Context

RISC-V Extension

Protection against SCA

Conclusion
The VexRisc Core

- 32-bit RISC-V ISA (RV32I)
  - Written in Spinal HDL
  - Powerful and flexible: https://riscv.org/blog/2018/12/risc-v-softcpu-contest-highlights/

- Plug-in based
  - New instructions are easy to add
  - A 64-bit computation can be decomposed into two 32-bit instructions

VexRisc Test Platform

VexRisc Plugin for a new instruction

The Plug-in Insertion in the VexRiscv Pipeline
VexRisc Plugin for a new instruction

Software Compilation Flow of a Plug-in Added Instruction

---Present_cipher.c---

// C code with inline ASM

...  
// s1, s2 --> inputs  
// d --> output  
uint32_t s1, s2;  
uint32_t d;  
// inline asm passing variables from the C code
asm volatile ("PRESENT_D [%rd],%[rs1],%[rs2]":\  
[rd]="r" (d) : [rs1]"r" (s1) , [rs2]"r" (s2))  
...  

Jean-Luc Danger  
May 2022
64-bit Output Using Two 32-bit RISC-V Instruction
# Resource usage

## Table: Resource Usage Overhead for the Agile Protected Implementation

<table>
<thead>
<tr>
<th>ISA</th>
<th>Configuration</th>
<th>Additional Instructions</th>
<th>LUTs</th>
<th>%</th>
<th>FFs</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference Basic RISC-V ISA</td>
<td>None</td>
<td>-</td>
<td>973</td>
<td>-</td>
<td>765</td>
<td>-</td>
</tr>
<tr>
<td>PRESENT</td>
<td></td>
<td>S-Box + PRESENT_D</td>
<td>1173</td>
<td>+21</td>
<td>767</td>
<td>+0.1</td>
</tr>
<tr>
<td>GIFT</td>
<td></td>
<td>S-Box + GIFT_D</td>
<td>1103</td>
<td>+13</td>
<td>767</td>
<td>+0.1</td>
</tr>
<tr>
<td>PRINCE</td>
<td></td>
<td>S-Box + PRINCE_DF + PRINCE_DM + PRINCE_DL</td>
<td>1438</td>
<td>+48</td>
<td>769</td>
<td>+0.5</td>
</tr>
<tr>
<td>NMAT</td>
<td></td>
<td>S-Box + NMAT_D</td>
<td>1778</td>
<td>+83</td>
<td>1023</td>
<td>+34</td>
</tr>
<tr>
<td>ALL</td>
<td></td>
<td>S-Box + PRESENT_D + GIFT_D + PRINCE_DF + PRINCE_DM + PRINCE_DL + NMAT_D</td>
<td>2113</td>
<td>+117</td>
<td>1028</td>
<td>+34</td>
</tr>
</tbody>
</table>

Complexity x 1.2 to 2.2 depending on the Agility requirement
## Execution Latency Gain

<table>
<thead>
<tr>
<th>Block Cipher</th>
<th>Base ISA</th>
<th>LWC ISA</th>
<th>Gain factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRESENT</td>
<td>12544</td>
<td>358</td>
<td>35</td>
</tr>
<tr>
<td>GIFT</td>
<td>10661</td>
<td>319</td>
<td>33</td>
</tr>
<tr>
<td>PRINCE</td>
<td>17357</td>
<td>126</td>
<td>138</td>
</tr>
<tr>
<td>Midori</td>
<td>18944</td>
<td>232</td>
<td>81</td>
</tr>
<tr>
<td>Twine</td>
<td>41279</td>
<td>622</td>
<td>66</td>
</tr>
<tr>
<td>Skinny</td>
<td>40887</td>
<td>409</td>
<td>100</td>
</tr>
</tbody>
</table>

**Table:** Instruction Count for Different Algorithms
Outline

Context

RISC-V Extension

Protection against SCA

Conclusion
Guessing Entropy of a Non-protected Execution of PRESENT, up to 30000 power traces
The RSM Protection

- The plaintext is masked with a mask rotated randomly at nibble level.
- Unmasking is done before the S-Box and remasking after.
- The operation $Y = M_{i+1} \oplus S(X \oplus (M_i))$ is implemented in a table ⇒ Hence the Side-channel leakage is not visible inside the table.
- The mask changes by rotation by 1 nibble at each round.

The RSM Confusion step

$s_0 \oplus m_1 \rightarrow \text{Barrel Shifter} \rightarrow \text{S-Box} (m_0-m_1) \rightarrow \text{Barrel Shifter} \rightarrow \text{SB}(s_0) \oplus m_2$

$s_1 \oplus m_2 \rightarrow \text{Barrel Shifter} \rightarrow \text{S-Box} (m_1-m_2) \rightarrow \text{Barrel Shifter} \rightarrow \text{SB}(s_1) \oplus m_3$

$s_2 \oplus m_3 \rightarrow \text{Barrel Shifter} \rightarrow \text{S-Box} (m_2-m_3) \rightarrow \text{Barrel Shifter} \rightarrow \text{SB}(s_2) \oplus m_4$

$s_{15} \oplus m_0 \rightarrow \text{Barrel Shifter} \rightarrow \text{S-Box} (m_{15}-m_0) \rightarrow \text{Barrel Shifter} \rightarrow \text{SB}(s_{15}) \oplus m_1$
Leakage due to Instruction Ordering

The nibble rotation is performed using a hardware Nibble-level Barrel Shifter and two additional instructions: **NSHIFT** and **invNSHIFT**

Algorithm Protected Block Cipher Algorithm

Mask_init()

\[ \text{roundKey}[32] \leftarrow \text{GenerateRoundKeys(Key)} \]

\[ \text{curr\_mask} \leftarrow \text{invNShift(Mask)} \]

\[ \text{state} \leftarrow \text{Plaintext} \oplus \text{curr\_mask} \]

\[
\text{for} \quad \text{round} = 0 \text{ to } \#\text{ofRounds do }
\]

\[ \text{state} \leftarrow \text{state} \oplus \text{roundKey[round]} \]

\[ \text{state} \leftarrow \text{NShift(state)} \]

\[ \text{state} \leftarrow \text{SBox(state)} \]

\[ \text{state} \leftarrow \text{invNShift(state)} \]

\[ \text{state} \leftarrow \text{Diffusion(state)} \]

\[ \text{Mask\_next()} \]

\[ \text{curr\_mask} \leftarrow \text{invNShift(Mask)} \]

\[ \text{comp\_mask} \leftarrow \text{Diffusion(curr\_mask) } \oplus \text{curr\_mask} \]

\[ \text{state} \leftarrow \text{state} \oplus \text{comp\_mask} \]

\text{end for }

\[ \text{state} \leftarrow \text{state} \oplus \text{curr\_mask} \]

\text{return state}
Leakage due to Instruction Ordering

Algorithm Instruction Ordering with Leakage

Mask_next()
curr_mask ← invNShift(Mask)
state ← Diffusion(state)
comp_mask ← Diffusion(curr_mask) ⊕ curr_mask

Algorithm Instruction Ordering without Leakage

state ← Diffusion(state)
Mask_next()
curr_mask ← invNShift(Mask)
comp_mask ← Diffusion(curr_mask) ⊕ curr_mask
## Resource Usage with RSM protections in Artix7 FPGA

### Table: Resource Usage Overhead for the Agile Protected Implementation

<table>
<thead>
<tr>
<th>ISA</th>
<th>Configuration</th>
<th>Additional Instructions</th>
<th>LUTs</th>
<th>%</th>
<th>FFs</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference Basic RISC-V ISA</td>
<td>None</td>
<td>-</td>
<td>973</td>
<td>-</td>
<td>765</td>
<td>-</td>
</tr>
<tr>
<td>PRESENT</td>
<td>S-Box + PRESENT_D + Nibble-Level Barrel Shifter</td>
<td>1971</td>
<td>+103</td>
<td>804</td>
<td>+0.5</td>
<td></td>
</tr>
<tr>
<td>GIFT</td>
<td>S-Box + GIFT_D + Nibble-Level Barrel Shifter</td>
<td>1968</td>
<td>+102</td>
<td>804</td>
<td>+0.5</td>
<td></td>
</tr>
<tr>
<td>PRINCE SBOX_C x1</td>
<td>S-Box + PRINCE_DF + PRINCE_DM + PRINCE_DL + Nibble-Level Barrel Shifter</td>
<td>2027</td>
<td>+108</td>
<td>802</td>
<td>+0.5</td>
<td></td>
</tr>
<tr>
<td>Protected RISC-V Extension ProtLBC-ISA</td>
<td>S-Box + PRINCE_DF + PRINCE_DM + PRINCE_DL + NMAT_D + Nibble-Level Barrel Shifter</td>
<td>2227</td>
<td>+129</td>
<td>803</td>
<td>+0.5</td>
<td></td>
</tr>
<tr>
<td>NMAT</td>
<td>S-Box + NMAT_D + Nibble-Level Barrel Shifter</td>
<td>2432</td>
<td>+150</td>
<td>1057</td>
<td>+38</td>
<td></td>
</tr>
<tr>
<td>ALL (with Protection)</td>
<td>S-Box + PRESENT_D + GIFT_D + PRINCE_DF + PRINCE_DM + PRINCE_DL + NMAT_D + Nibble-Level Barrel Shifter</td>
<td>2595</td>
<td>+167</td>
<td>1061</td>
<td>+39</td>
<td></td>
</tr>
</tbody>
</table>

Jean-Luc Danger
May 2022
Attack Results with RSM Protection

Guessing Entropy of a Protected Execution of PRESENT, up to 1 million power traces

20/23

Jean-Luc Danger

May 2022
Outline

Context

RISC-V Extension

Protection against SCA

Conclusion
Conclusion

- **LWC block ciphering** can be greatly accelerated by extending the RISC-V ISA of the VexRisc:
  - By adding **5 new instructions** dedicated to 128-bit key, 64-bit message
  - Providing an acceleration of a factor **33-100**
  - With a complexity x 1.3 to 2.5 according to the agility level

- They can also be protected against SCA at 1st order,
  - By adding **2 instructions** to implement **RSM**
  - Protection validated with 1 million traces with all considered LWC
  - With a complexity x 2 to 2.7 according to the agility level
Thank you for your attention and questions