PHISIC

May 2022

RISC-V Extension for Lightweight Cryptography, Protection against SCA

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

Outline

Outline

Context

- Requirement of **Agility** for **Lightweight Cryptography** (LWC) to suit any emerging application or standard.
- The **RISC-V ISA** can be extended tu 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

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

Table: Categorisation of Lightweight Block Ciphers

Classification of LWC Block Ciphers

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

Table: Categorisation of Lightweight Block Ciphers

Additional Hardware Instructions

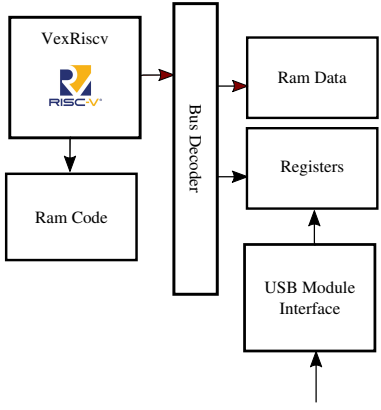
Specific instructions (64-bit data) which could accelerate LWC execution:

- **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 specfic 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

The VexRisc Core

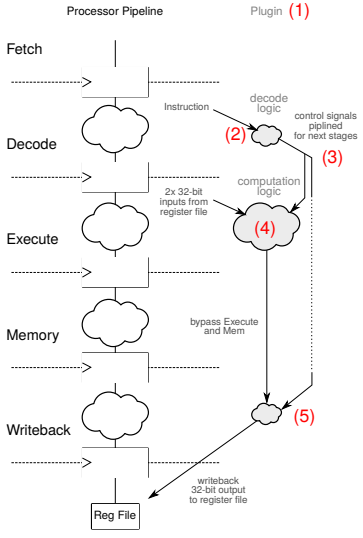
- 32-bit RISC-V ISA (RV32I)
  - Written in Spinal HDL
  - Powerful and flexible: https://riscv.org/blog/2018/12/risc-v-softcpu-contest-highlights/
- Plug-in based
  - New instructions are easy to add
  - A 64-bit computation can be decomposed into two 32-bit instructions



VexRisc Test Platform

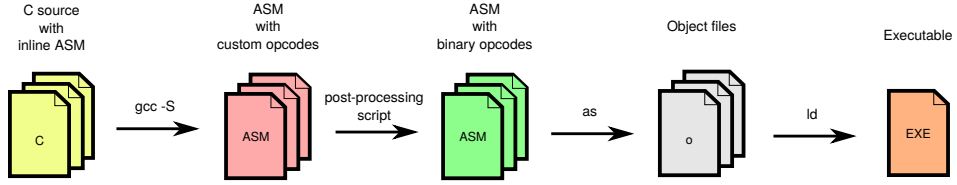
https://spinalhdl.github.io/SpinalDoc-RTD/v1.3.1/SpinalHDL/Libraries/vexriscv.html

VexRisc Plugin for a new instruction



The Plug-in Insertion in the VexRiscv Pipeline

VexRisc Plugin for a new instruction

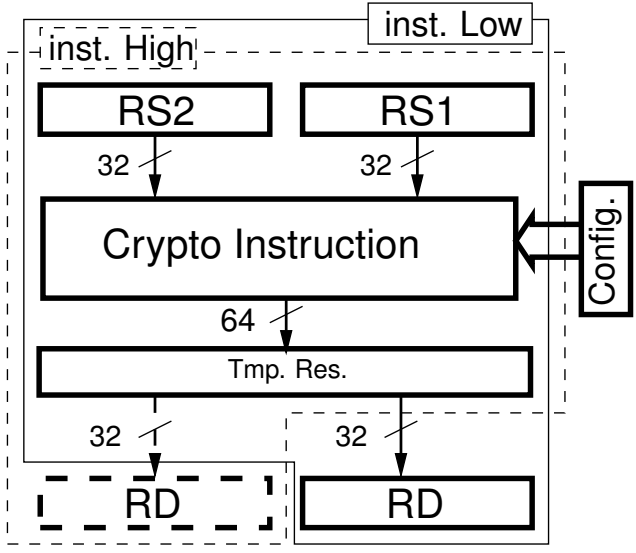


Software Compilation Flow of a Plug-in Added Instruction

```
----------------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))
...
-----------------------------------------------
```

Instructions Implementation



64-bit Output Using Two 32-bit RISC-V Instruction

Resource usage

Table: Resource Usage Overhead for the Agile Protected Implementation

ISA	Configuration	Additional Instructions	LUTs	%	FFs	%
Reference Basic RISC-V ISA	None	-	973	-	765	-
Non-Protected Extended RISC-V LBC-ISA	PRESENT	S-Box + PRESENT_D	1173	+21	767	+0.1
	GIFT	S-Box + GIFT_D	1103	+13	767	+0.1
	PRINCE	S-Box + PRINCE_DF + PRINCE_DM + PRINCE_DL	1438	+48	769	+0.5
	NMAT	S-Box + NMAT_D	1778	+83	1023	+34
	ALL	S-Box + PRESENT_D + GIFT_D + PRINCE_DF + PRINCE_DM + PRINCE_DL + NMAT_D	2113	+117	1028	+34

Complexity x 1.2 to 2.2 depending on the Agility requirement

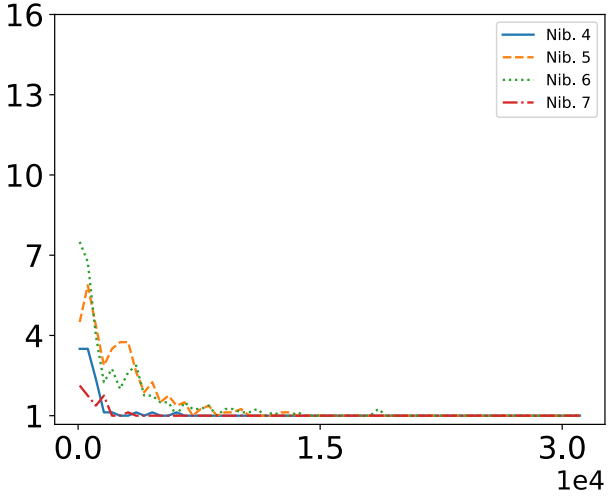
Execution Latency Gain

Block Cipher	Base ISA	LWC ISA	Gain factor
PRESENT	12544	358	35
GIFT	10661	319	33
PRINCE	17357	126	138
Midori	18944	232	81
Twine	41279	622	66
Skinny	40887	409	100

Table: Instruction Count for Different Algorithms

Outline

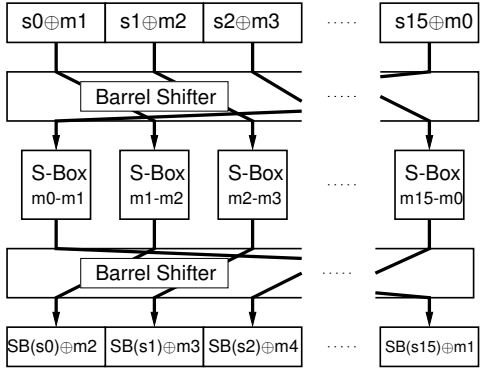
Attack Results without Protection



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** $\Rightarrow$ 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

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()
roundKey[32] ← *GenerateRoundKeys*(*Key*)
curr_mask ← *invNShift*(*Mask*)
state ← *Plaintext* ⊕ *curr_mask*

for *round* = 0 to #*ofRounds* **do**
  state ← *state* ⊕ *roundKey*[*round*]
  state ← *NShift*(*state*)
  state ← *SBox*(*state*)
  state ← *invNShift*(*state*)
  state ← Diffusion(state)
  Mask_next()
  curr_mask ← invNShift(Mask)
  comp_mask ← Diffusion(curr_mask) ⊕ curr_mask
  state ← *state* ⊕ *comp_mask*
end for
state ← *state* ⊕ *curr_mask*
return *state*

Leakage due to Instruction Ordering

Algorithm Instruction Ordering with Leakage

Mask_next()
curr_mask $\leftarrow$ *invNShift*(*Mask*)
state $\leftarrow$ Diffusion(state)
comp_mask $\leftarrow$ *Diffusion*(*curr_mask*) $\oplus$ *curr_mask*

$\Downarrow$

Algorithm Instruction Ordering without Leakage

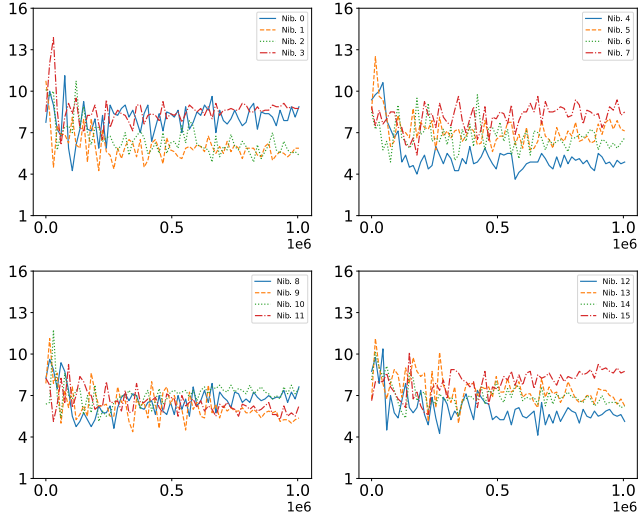
state $\leftarrow$ Diffusion(state)
Mask_next()
curr_mask $\leftarrow$ *invNShift*(*Mask*)
comp_mask $\leftarrow$ *Diffusion*(*curr_mask*) $\oplus$ *curr_mask*

Resource Usage with RSM protections in Artix7 FPGA

Table: Resource Usage Overhead for the Agile Protected Implementation

ISA	Configuration	Additional Instructions	LUTs	%	FFs	%
Reference Basic RISC-V ISA	None	-	973	-	765	-
Protected RISC-V Extension ProtLBC-ISA	PRESENT	S-Box + PRESENT_D + Nibble-Level Barrel Shifter	1971	+103	804	+0.5
	GIFT	S-Box + GIFT_D + Nibble-Level Barrel Shifter	1968	+102	804	+0.5
	PRINCE SBOX_C x1	S-Box + PRINCE_DF + PRINCE_DM + PRINCE_DL + Nibble-Level Barrel Shifter	2027	+108	802	+0.5
	PRINCE SBOX_C x2	S-Box + PRINCE_DF + PRINCE_DM + PRINCE_DL + Nibble-Level Barrel Shifter	2227	+129	803	+0.5
	NMAT	S-Box + NMAT_D + Nibble-Level Barrel Shifter	2432	+150	1057	+38
	ALL (with Protection)	S-Box + PRESENT_D + GIFT_D + PRINCE_DF + PRINCE_DM + PRINCE_DL + NMAT_D + Nibble-Level Barrel Shifter	2595	+167	1061	+39

Attack Results with RSM Protection



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

Outline

TELECOM Paris

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