FROM RESEARCH TO INDUSTRY

cea tech

Compilation of a Countermeasure Against Instruction-Skip Fault Attacks

Thierno Barry[1] Damien Couroussé[1] Bruno Robisson[2]

[1] CEA – LIST / LIALP
Grenoble, France

[2] CEA – Tech / DPACA
Gardanne, France

firstname.lastname@cea.fr

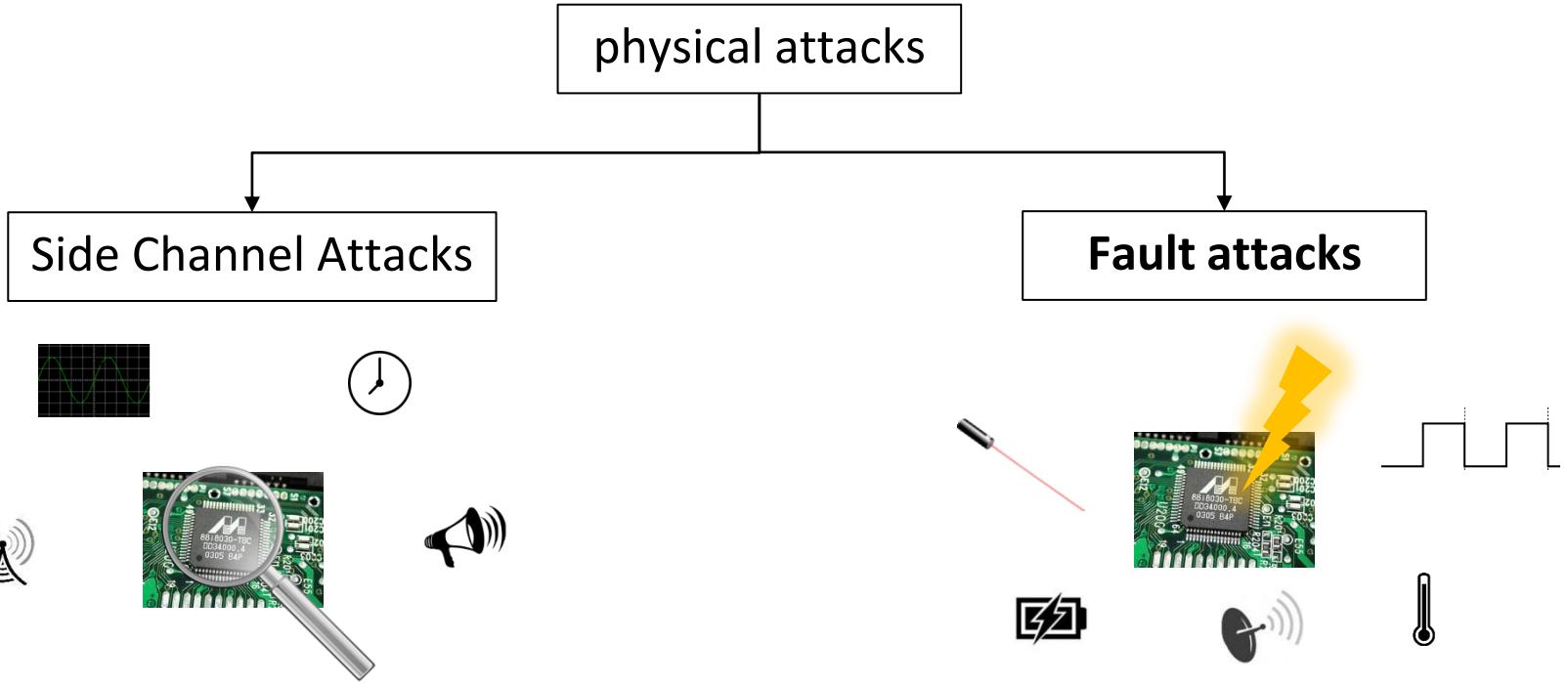
3th Workshop on Cryptography and Security in Computing Systems

Prague Jan. 20, 2016

www.cea.fr

leti & list

Context



Observing physical quantities of the device during its operation

Injecting a fault in order to disrupt the normal functioning of the device

Our objective

- Using compiler techniques to efficiently automate the application of software countermeasures against fault attacks

Context

Fault Model

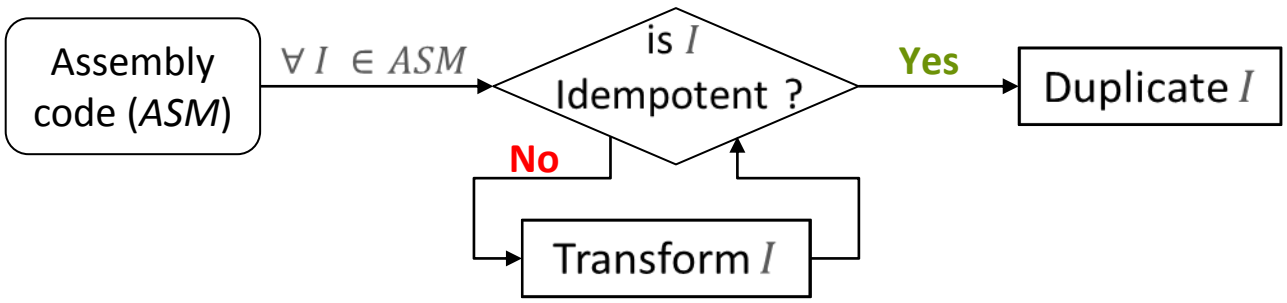
- A fault may occurs at different levels

Fault level	Fault Model	Countermeasure
Algorithmic	➢ Skip an instruction	➢ Instruction redundancy
Instruction	➢ Corrupt a data being transferred from/to memory	
Register		
Transistor		

- We propose an implementation of the instruction duplication technique

- Based on the scheme proposed and formally verified by [Moro et al. 2014] :
  "**Formal verification of a software countermeasure against instruction skip attacks.**"

Instruction Duplication Scheme



*"An instruction is idempotent when it can be **re-executed** several times with always the same result"*

Example:

is *idempotent*

```
add R0, R1, R2
```
Duplication →
```
add R0, R1, R2
add R0, R1, R2
```

Is not *idempotent*

```
add R1, R1, R2
```
→ ✗ →
```
add R1, R1, R2
add R1, R1, R2
```

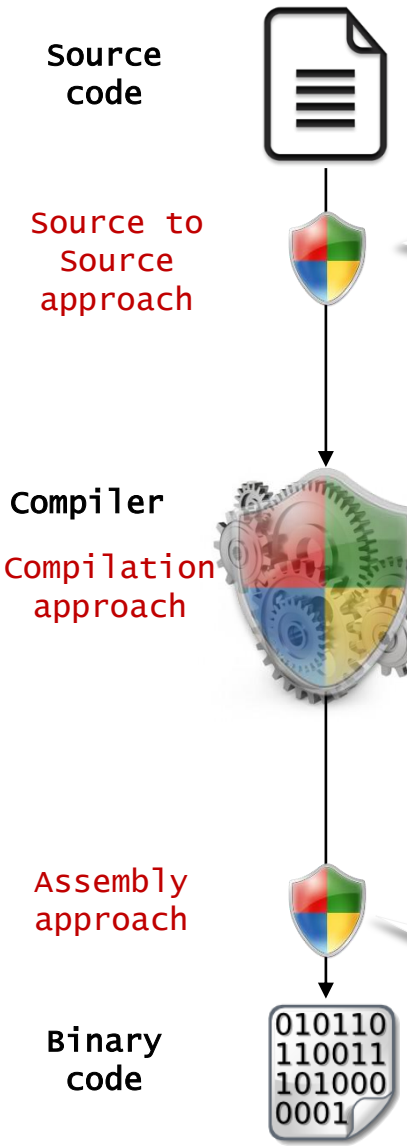
Transformation [Moro et al. 2014]

```
mv RX, R1
add R1, RX, R2
```
Duplication →
```
mv RX, R1
mv RX, R1
add R1, RX, R2
add R1, RX, R2
```

Limitations

- **How to find a free register at the assembly code level ?**
  - For [Barenghi et al. 2010] the number of free registers are known for their implemented AES
  - For [Moro et al. 2014] the use of the ARM scratch register `r12`
- **Overhead:**
  - At least × **4** for each instruction
  - [Moro et al. 2014] Reported × **14** for the ARM instruction: `umlal`

Implementation approaches



- Security properties cannot be guaranteed after the compilation [Balakrishan et al. 2008]
- Except if the compiler code optimizers are disabled as suggested in [Eldib et al. 2014]
  ➔ leads to a very high overheads + 400% in [Lalande et al. 2014]

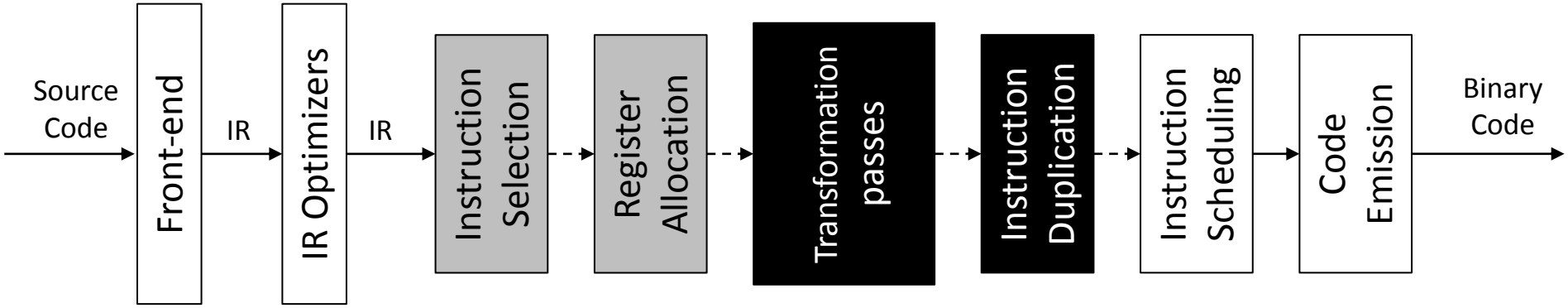
- Unlike the source to source approach we have control over code optimizers
- Unlike assembly approach we have the benefit of code transformation opportunities provided by the compiler

  ➔ Allows to reduce the security overhead

- Several transformations need to be performed
  ➔ leads to significant overheads [Moro et al. 2014]

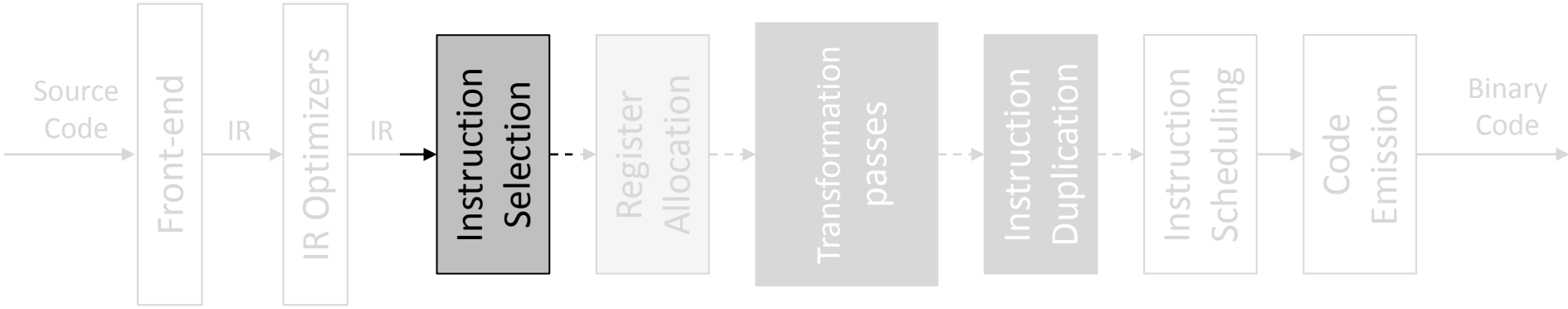
Our approach

- We implemented the instruction duplication inside the LLVM compiler



Our approach

- We implemented the instruction duplication inside the LLVM compiler



This pass is responsible for selecting the appropriate target instructions for each operation described by the program developer

This pass is modified in such a way that idempotent instructions are the ones privileged during the selection

Example:

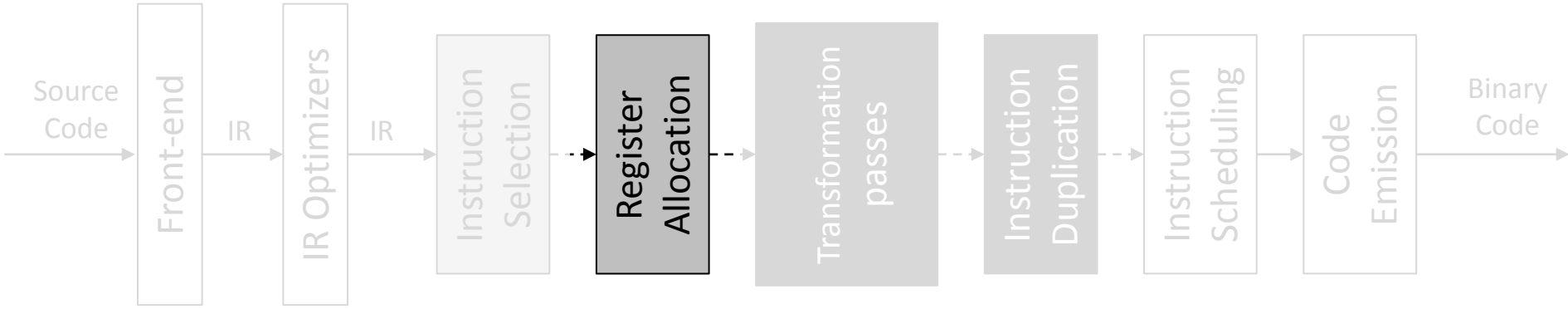
For the operation: $a * b + c$

`mul` and `add` are selected instead of `mla`

`mla` is not idempotent
But `mul` and `add` can be idempotent if the source and destination registers are different

Our approach

- We implemented the instruction duplication inside the LLVM compiler



This pass is responsible for mapping the endless number of program variables to a limited number of physical registers

This pass is modified to introduce a constraint
so that: destinations registers are always different to sources ones

Example:

For the operation: $a = b + c$

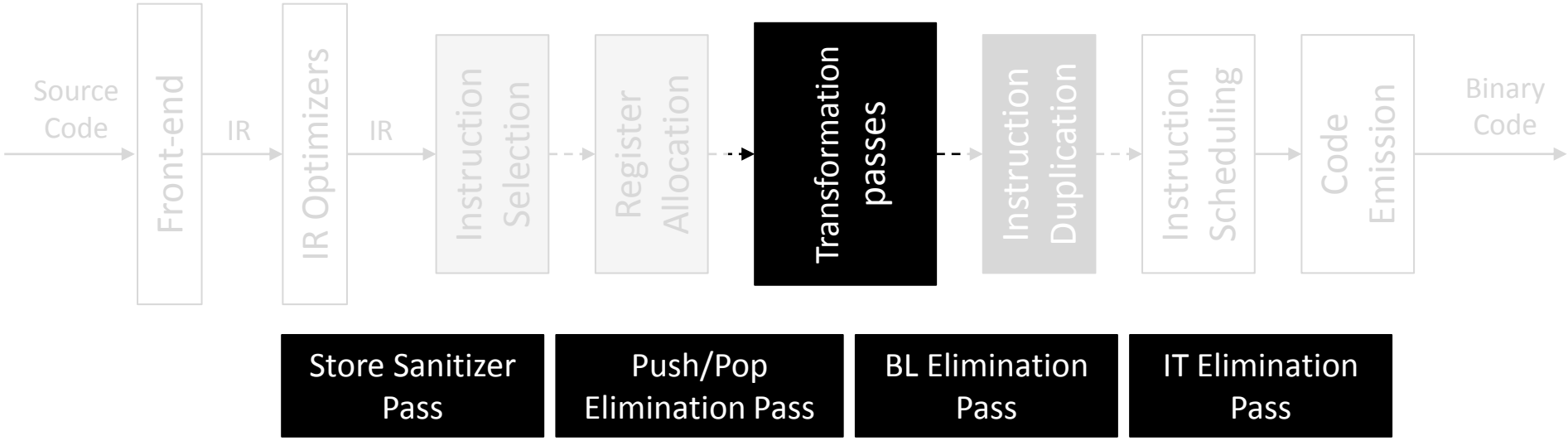
instead of having: `add R0, R0, R1`

we have something like: `add R0, R1, R2` → Duplication →

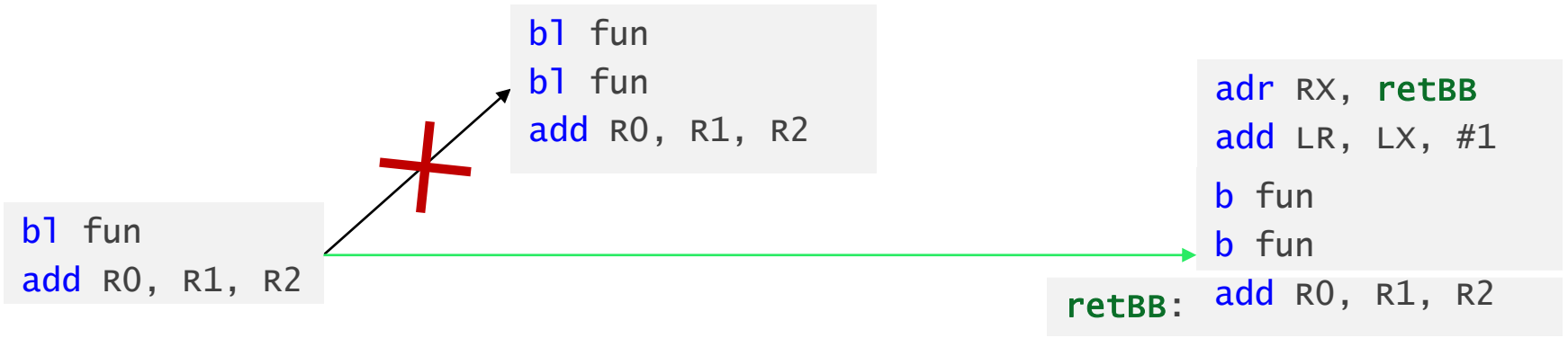
```
add R0, R1, R2
add R0, R1, R2
```

Our approach

- We implemented the instruction duplication inside the LLVM compiler

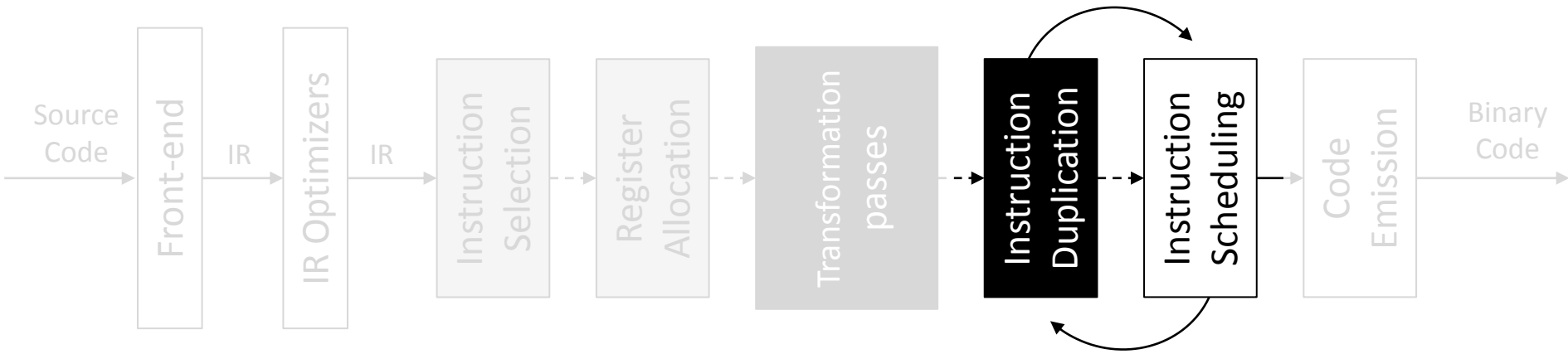


The role of these passes is to handle instructions that need special treatments



Our approach

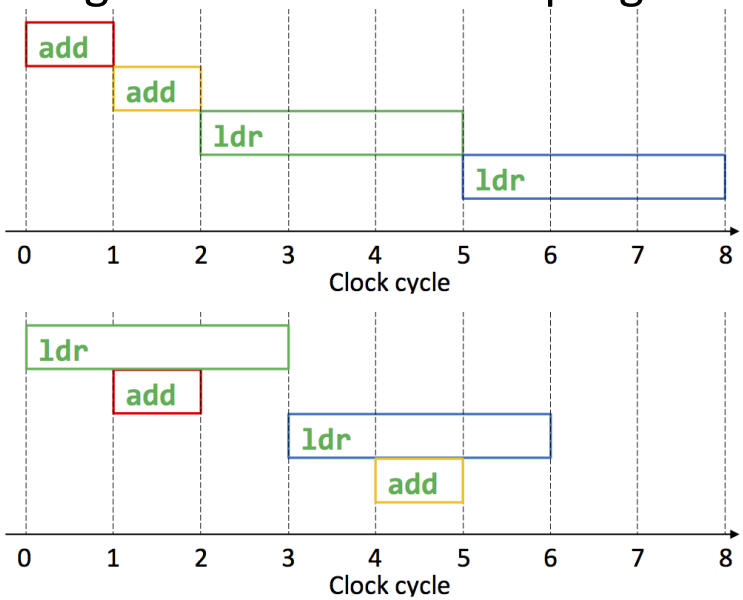
- We implemented the instruction duplication inside the LLVM compiler



The role of the scheduler is to rearrange the execution order of instruction in order to improve the execution time while preserving the original behavior of the program

Example:

```
add R0, R1, R2
add R0, R1, R2
ldr R3, [R1, #4]
ldr R3, [R1, #4]
```



Experimental evaluation

	Unprotected		Overhead		Moro et al. 2014	
	Cycles	Size	Cycles	Size	Cycles	Size
Moro et al.'s AES	14407	11552	× 1.71	× 1.15	× 2.14	× 3.02
MiBench AES	1908	67644	× 1.76	× 1.18	× 2.86	× 2.90

Target Architecture: ARM Cortex-M3

- **Cycles**: clock cycles
- **Size**: Bytes

- More than 95% of instructions we generate are idempotent
  - Only less than 5% need to be transformed
- The impact of the scheduler

- Our ARM-based Microcontroller supports both 32-bit and 16-bit instruction set
  - The compiler selects 16-bit instructions whenever it is possible

Conclusion

- We proposed a modified LLVM compiler to efficiently automate the application of the instruction duplication technique
- We illustrated through experimentations the effectiveness of our approach in terms of overheads compared to existing solutions

Thanks for your attention

Compilation of a Countermeasure Against Instruction-Skip Fault Attacks

Thierno Barry
CEA – LIST / LIALP
Grenoble, France

thierno.barry@cea.fr

http://thiernobarry.fr

References

- Balakrishnan, G., & Reps, T. (2010). Wysinwyx: What you see is not what you execute. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *32*(6), 23..

- Eldib, H., Wang, C., Taha, M., & Schaumont, P. (2014, June). QMS: Evaluating the side-channel resistance of masked software from source code. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE* (pp. 1-6). IEEE.

- Lalande, J. F., Heydemann, K., & Berthomé, P. (2014). Software countermeasures for control flow integrity of smart card C codes. In *Computer Security-ESORICS 2014* (pp. 200-218). Springer International Publishing.

- Moro, N., Heydemann, K., Encrenaz, E., & Robisson, B. (2014). Formal verification of a software countermeasure against instruction skip attacks. *Journal of Cryptographic Engineering*, *4*(3), 145-156.

- Barenghi, A., Breveglieri, L., Koren, I., Pelosi, G., & Regazzoni, F. (2010, October). Countermeasures against fault attacks on software implemented AES: effectiveness and cost. In *Proceedings of the 5th Workshop on Embedded Systems Security* (p. 7). ACM.