Belleville Nicolas [1]
Barry Thierno [1]
Seriai Abderrahmane [1]
Couroussé Damien [1]
Heydemann Karine [2]
Robisson Bruno [3]
Charles Henri-Pierre [1]

[1] Univ Grenoble Alpes, CEA, List, F-38000 Grenoble, France
firstname.lastname@cea.fr
[2] Sorbonne Universités, UPMC, Univ. Paris 06, CNRS,LIP6,UMR 7606 75005 Paris, France
firstname.lastname@lip6.fr
[3] CEA/EMSE, Secure Architectures and Systems Laboratory CMP, 880 Route de Mimet, 13541 Gardanne, France
firstname.lastname@cea.fr

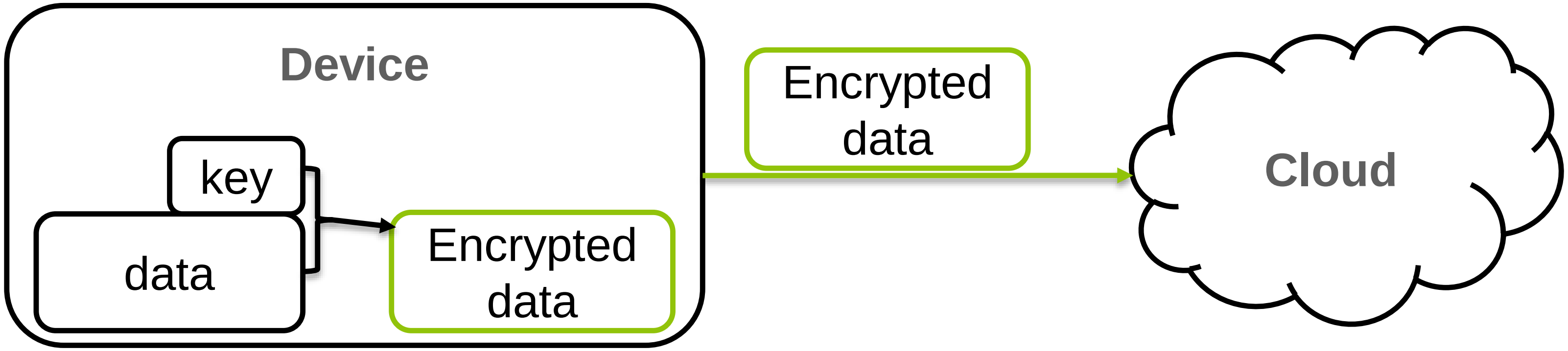
THE MULTIPLE WAYS TO AUTOMATE THE APPLICATION OF SOFTWARE COUNTERMEASURES AGAINST PHYSICAL ATTACKS: PITFALLS AND GUIDELINES

INTRODUCTION

- **In 2008, for an average person: 230 embedded chips used every day !**
- **Number of Cyber-Physical Systems is expected to grow**
- **Lots of them…**
  - Connected watches
  - Connected buildings
  - Smartphones
  - Monitors for human health in hospitals
  - …
- **… manipulate sensitive data**
  - Where you are
  - Messages between you and someone else
  - Pictures / videos of you or your house
  - Health data
  - …

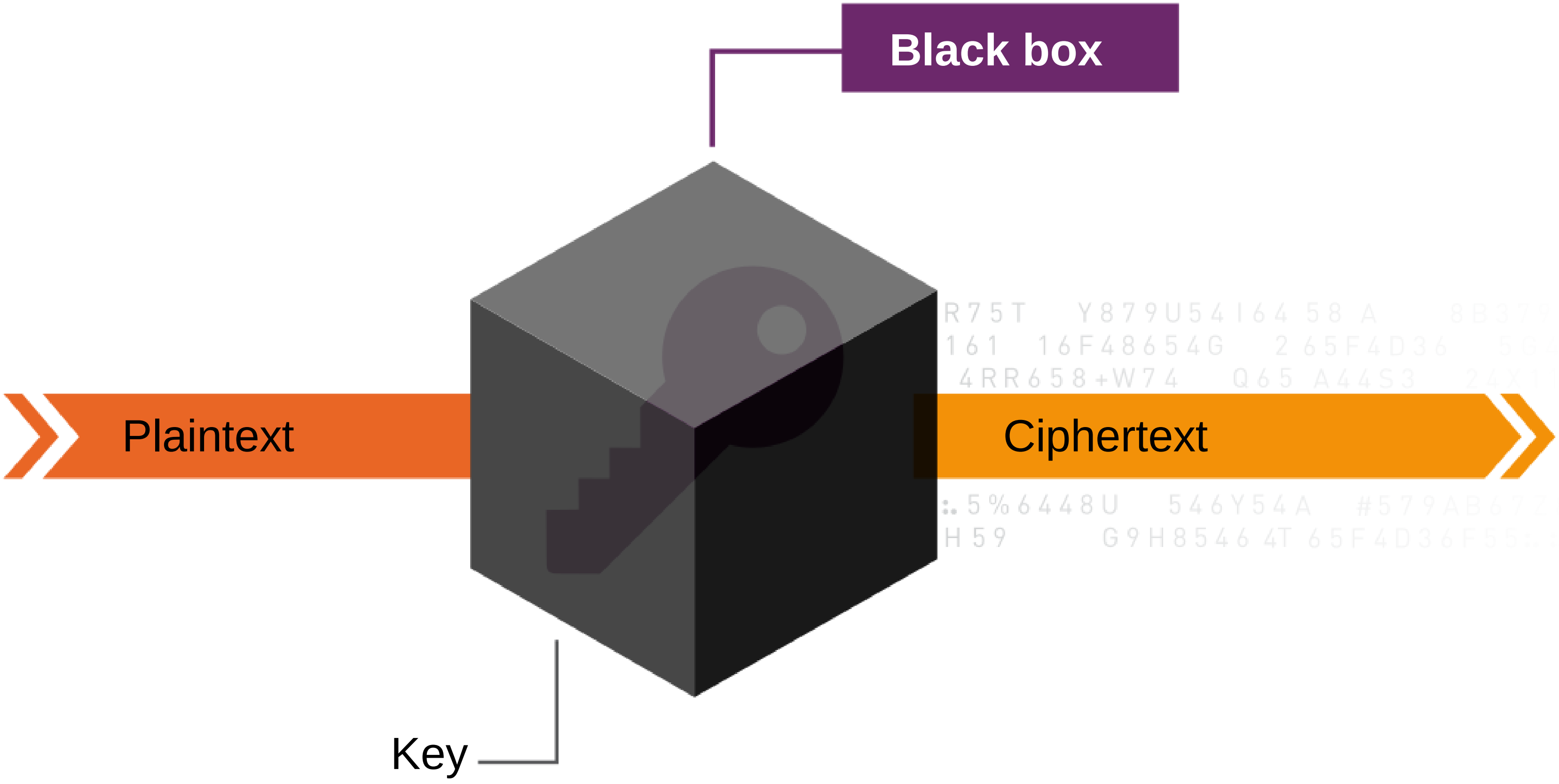
INTRODUCTION

- **Encryption is used to protect this data**
  - Secure transfers of data between connected objects and servers or cloud
  - Once encrypted, data cannot be recovered without the key



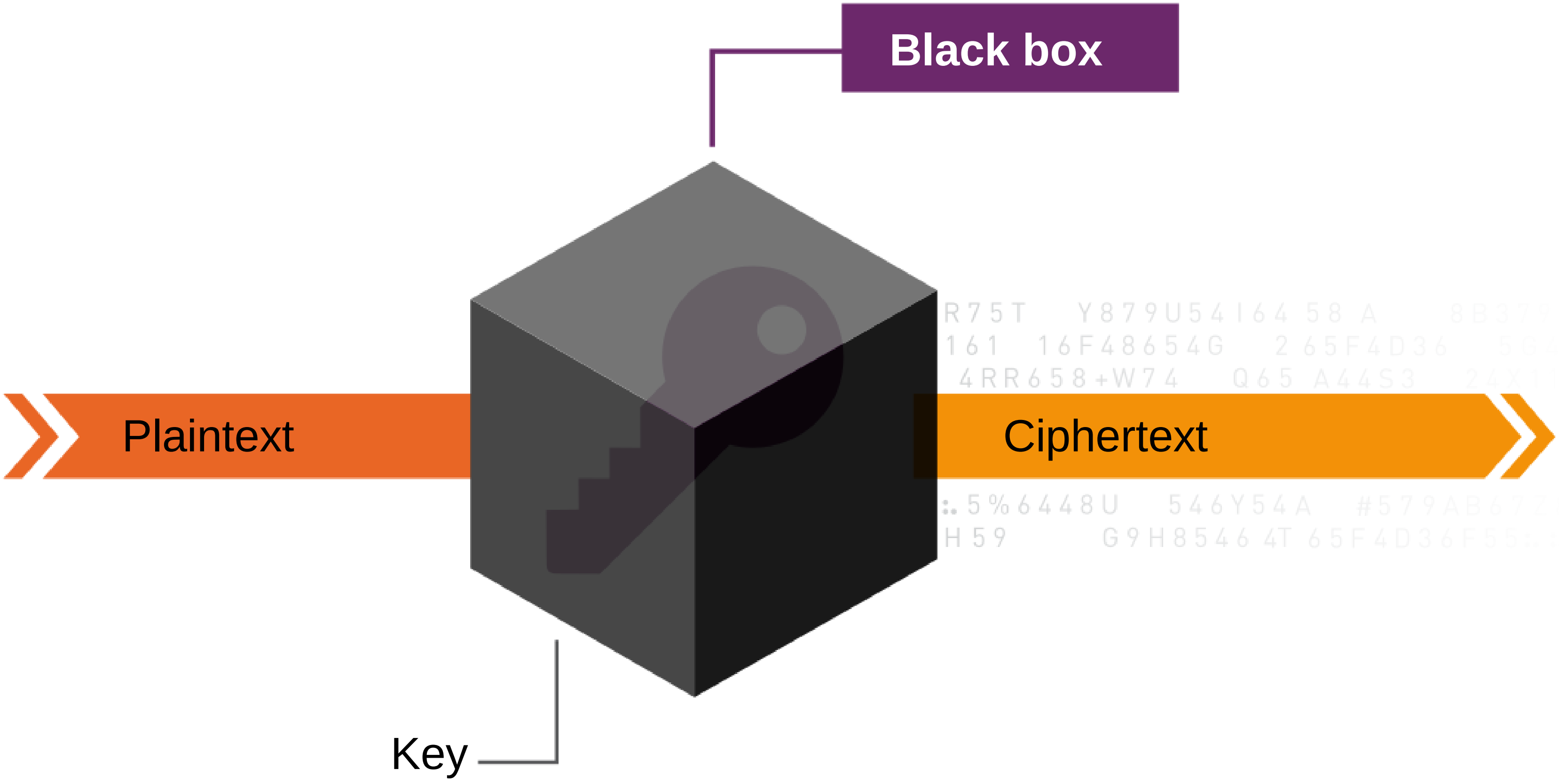
- **Cryptanalysis: The designs of encryption algorithms used are well studied**
  - Security relatively to attacker's means
  - Lot of research teams try to break them
  - Their designs are a lot studied!

- **Black box assumption**
  - the attacker has no physical access to the key, nor to any internal processing, but can only observe external information and behavior



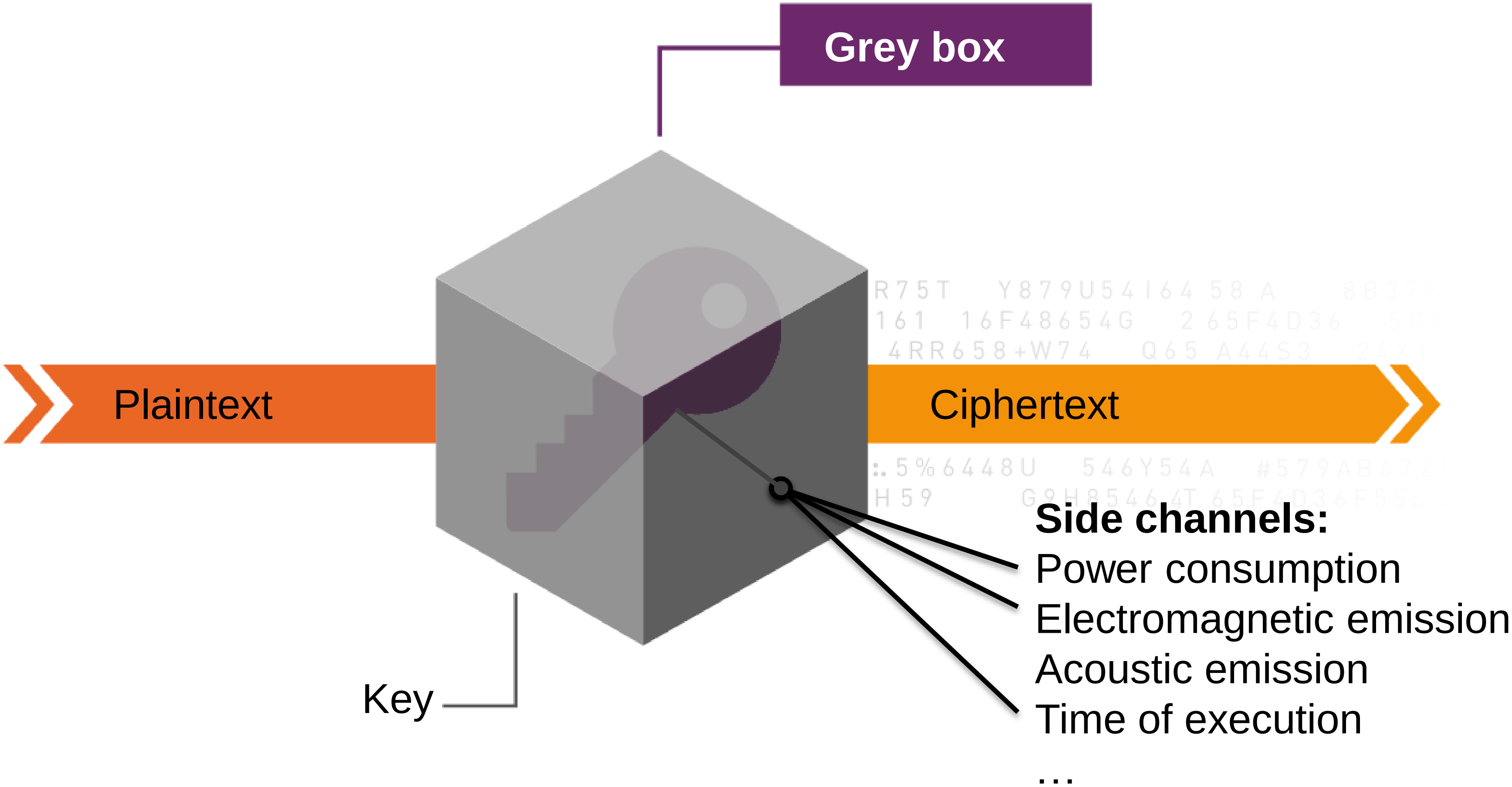
INTRODUCTION: CRYPTOGRAPHY

- **Black box assumption**
  - the attacker has no physical access to the key, nor to any internal processing, but can only observe **external information and behavior**



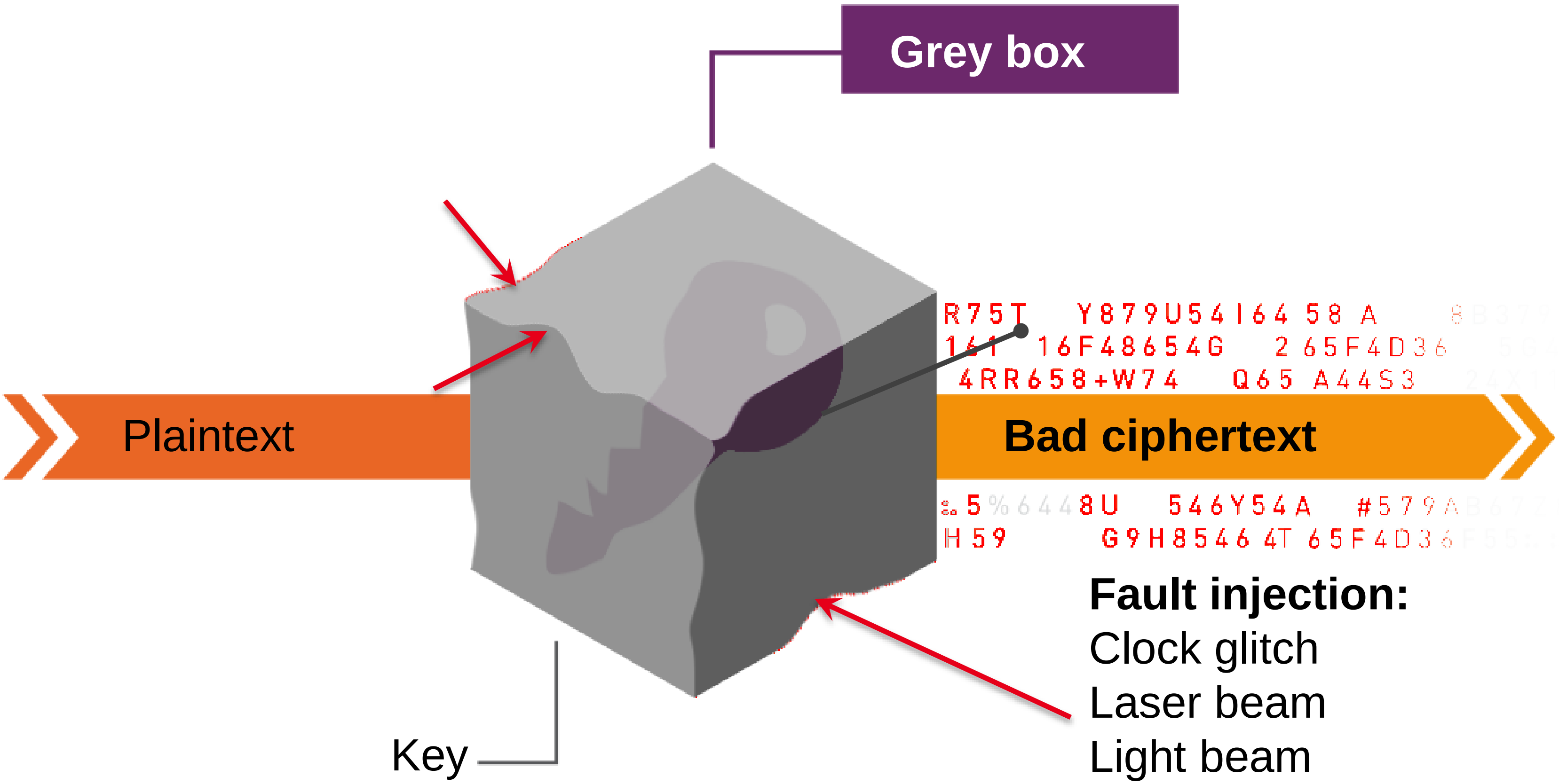
PHYSICAL ATTACKS: SIDE-CHANNEL ATTACKS

- **In reality: grey box**
  - Side channel information leakage:



PHYSICAL ATTACKS: FAULT INJECTION ATTACKS

- **In reality: grey box**
  - Side channel information leakage
  - System vulnerable to faults



Fault injection:
Clock glitch
Laser beam
Light beam
Heating
…

INTRODUCTION

- **Encryption is used to protect this data**
  - Secure transfers of data between connected objects and servers or cloud
  - Once encrypted, data cannot be recovered without the key

- **Cryptanalysis: The designs of encryption algorithms used are well studied**
  - Security relatively to attacker's means
  - Lot of research teams try to break them
  - Their designs are a lot studied!

- **Physical attacks are the only effective way to break cryptanalysis-resistant crypto ciphers**
  - That's why their countermeasures are usually evaluated on crypto blocks
  - But their range of target is BROADER than that

OUTLINE

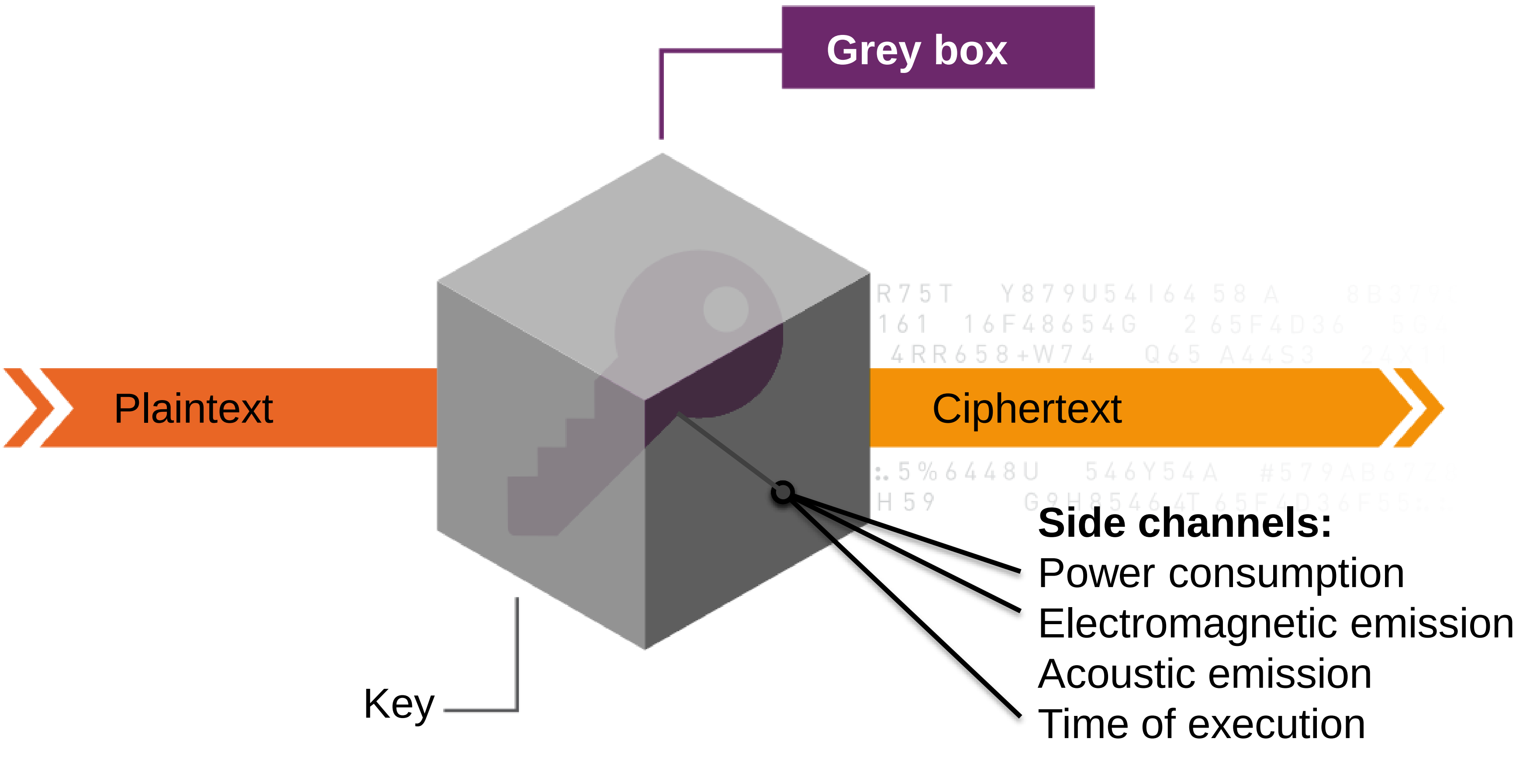
- Introduction

- **Side channel attacks detailed example:**
  how correlation power analysis works
- **Fault injection attacks detailed example:**
  how differential fault attacks works

- **Hardware countermeasures**
- **Software countermeasures**
  Why we want to apply them automatically
  Survey of existing approaches to apply some of them automatically
  Why we should take the compiler into account while applying countermeasure
  Why applying countermeasures within compilation process is valuable

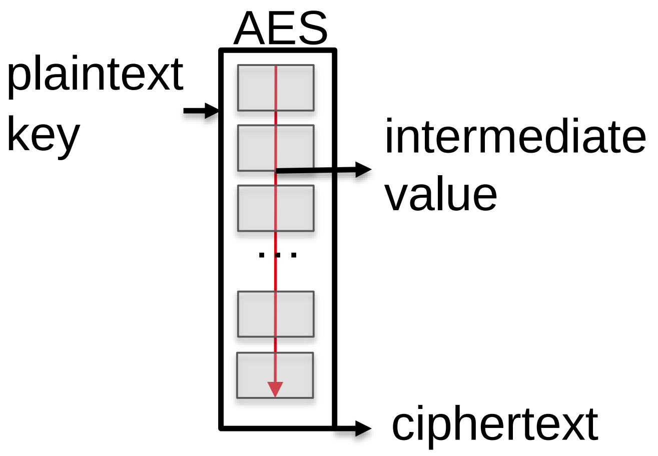
- **Conclusion**

PHYSICAL ATTACKS: SIDE-CHANNEL ATTACKS



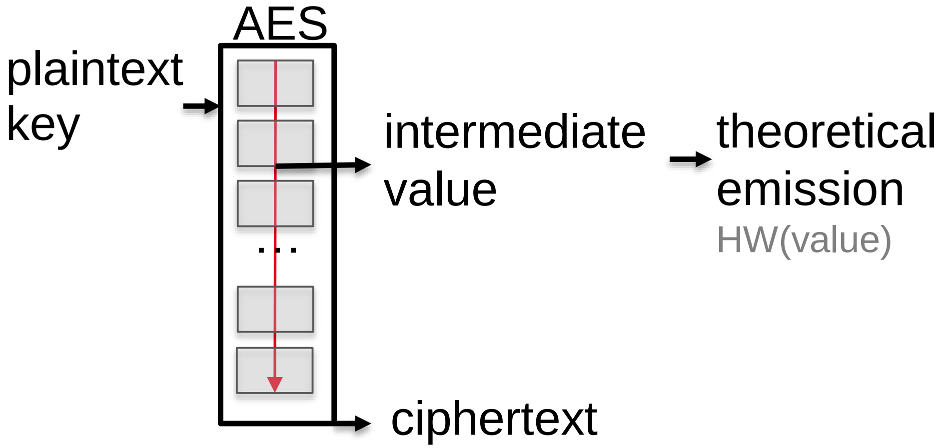
PHYSICAL ATTACKS: SIDE-CHANNEL ATTACKS

- **General approach:**
  - Divide and conquer: the key is recovered bit by bit or byte by byte
  - The attacker has a model of the electrical consumption / electromagnetic emission /…
- **Attack steps:**
  - Choose a target intermediate value
    - That depends only of one byte of the key ideally



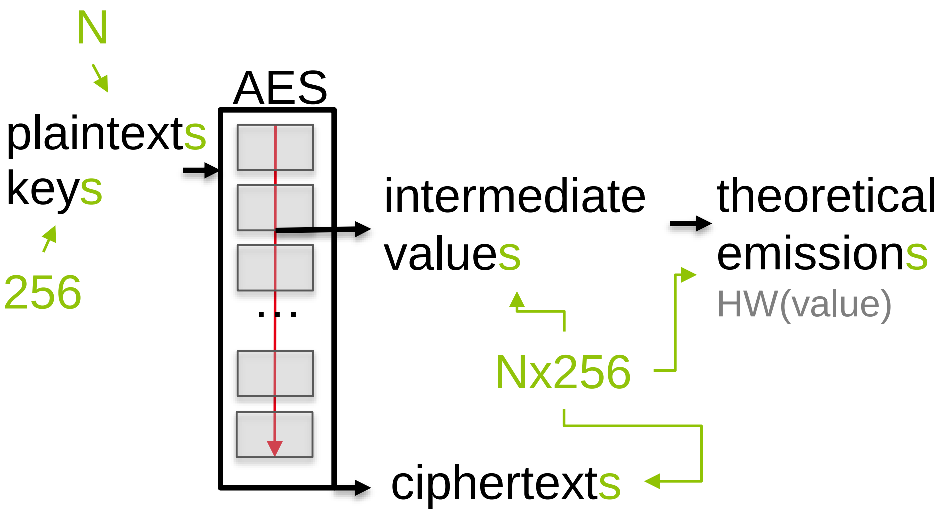
PHYSICAL ATTACKS: SIDE-CHANNEL ATTACKS

- **General approach:**
  - Divide and conquer: the key is recovered bit by bit or byte by byte
  - The attacker has a model of the electrical consumption / electromagnetic emission /…
- **Attack steps:**
  - Choose a target intermediate value
  - Compute a theoretical emission for this value for all key hypothesis
    - With a model of emission (hamming weight or hamming distance usually used)
    - The theoretical emission is computed for all key hypothesis for N plaintexts
    - We get Nx256 theoretical emissions (attack of one byte of the key)



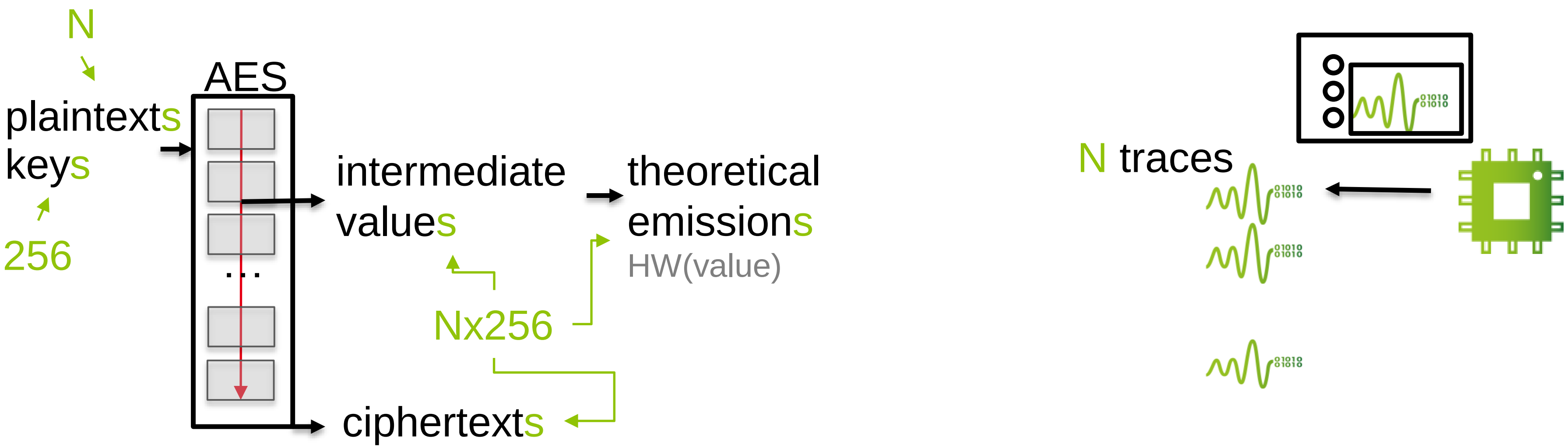
PHYSICAL ATTACKS: SIDE-CHANNEL ATTACKS

- **General approach:**
  - Divide and conquer: the key is recovered bit by bit or byte by byte
  - The attacker has a model of the electrical consumption / electromagnetic emission /…
- **Attack steps:**
  - Choose a target intermediate value
  - Compute a theoretical emission for this value for all key hypothesis
    - With a model of emission (hamming weight or hamming distance usually used)
    - The theoretical emission is computed for all key hypothesis for N plaintexts
    - We get Nx256 theoretical emissions (attack of one byte of the key)



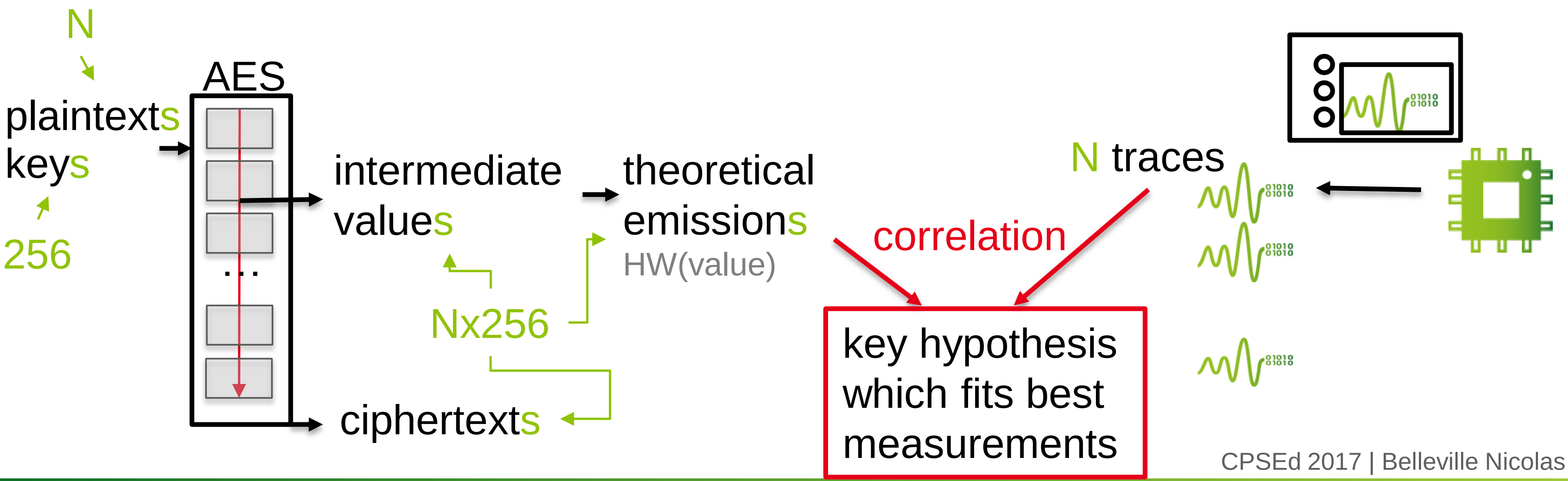
PHYSICAL ATTACKS: SIDE-CHANNEL ATTACKS

- **General approach:**
  - Divide and conquer: the key is recovered bit by bit or byte by byte
  - The attacker has a model of the electrical consumption / electromagnetic emission /…
- **Attack steps:**
  - Choose a target intermediate value
  - Compute a theoretical emission for this value for all key hypothesis
  - Measure emission through several encryptions
    - At least one encryption per plaintext
    - Measurements have to be aligned



PHYSICAL ATTACKS: SIDE-CHANNEL ATTACKS

- **General approach:**
  - Divide and conquer: the key is recovered bit by bit or byte by byte
  - The attacker has a model of the electrical consumption / electromagnetic emission /…
- **Attack steps:**
  - Choose a target intermediate value
  - Compute a theoretical emission for this value for all key hypothesis
  - Measure emission through several encryptions
  - Compare measurements with theoretical values
    - Highest correlation between theory and traces gives a key candidate

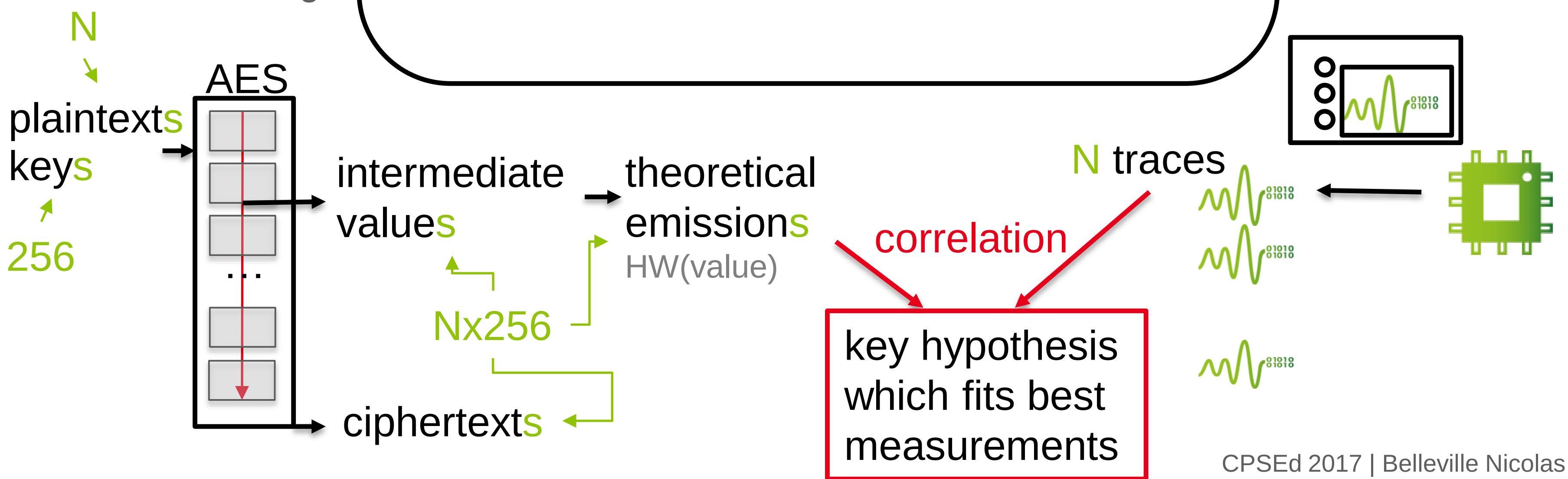


PHYSICAL ATTACKS: SIDE-CHANNEL ATTACKS

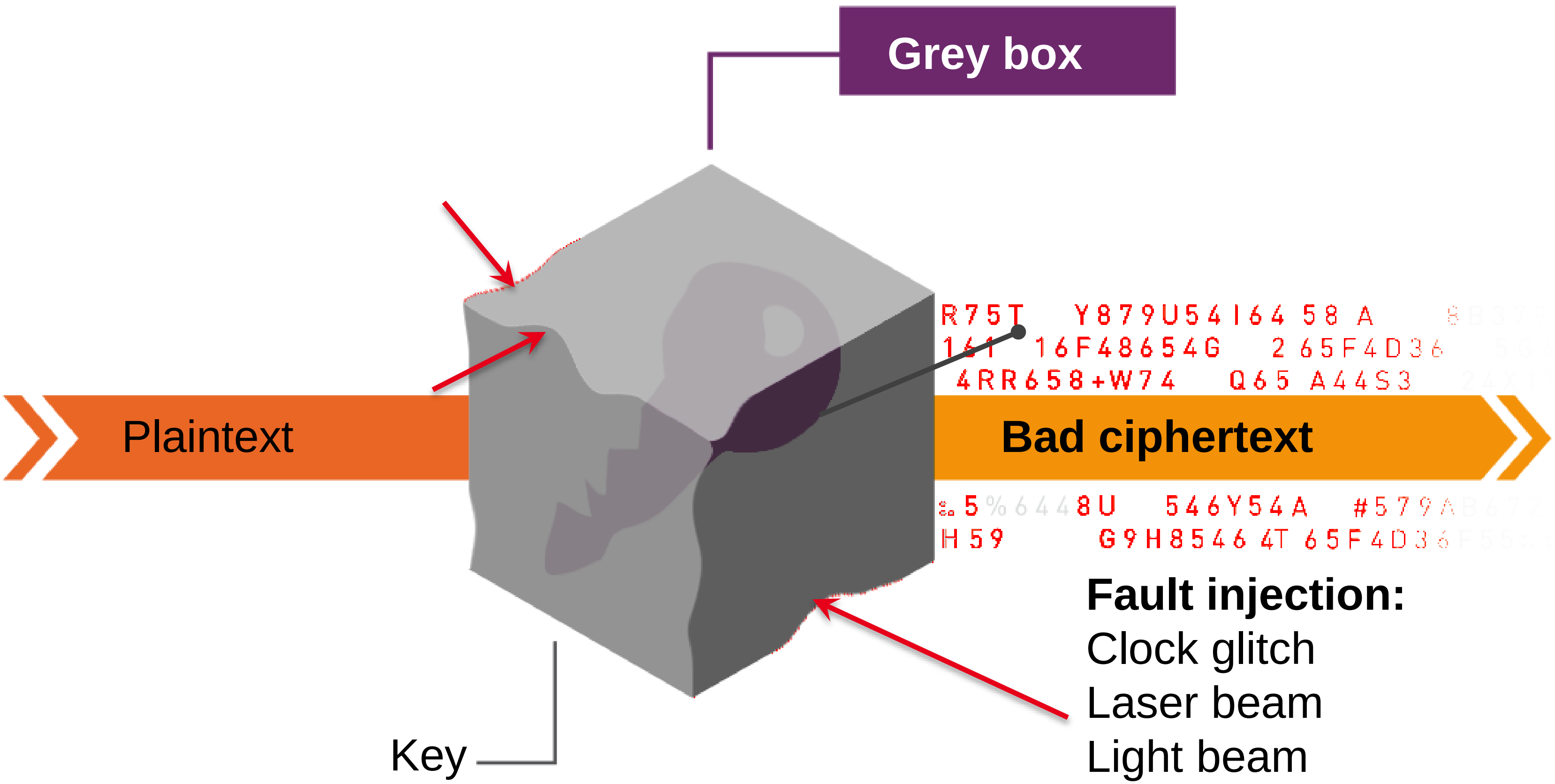
- **General approach:**
  - Divide an … by byte
  - The attac… …ectromagnetic emission…
- **Attack step…**
  - Choose …
  - Compute … …othesis
  - Measure…
  - Compare…
    - Highe… …ndidate

This is **an example** of how side channel attacks can be mounted.

BUT: they can target other kind of applications (web browsers, verifypin, …), and can also be used to help monitoring fault injection attacks



PHYSICAL ATTACKS: FAULT INJECTION ATTACKS



Fault injection:

Clock glitch

Laser beam

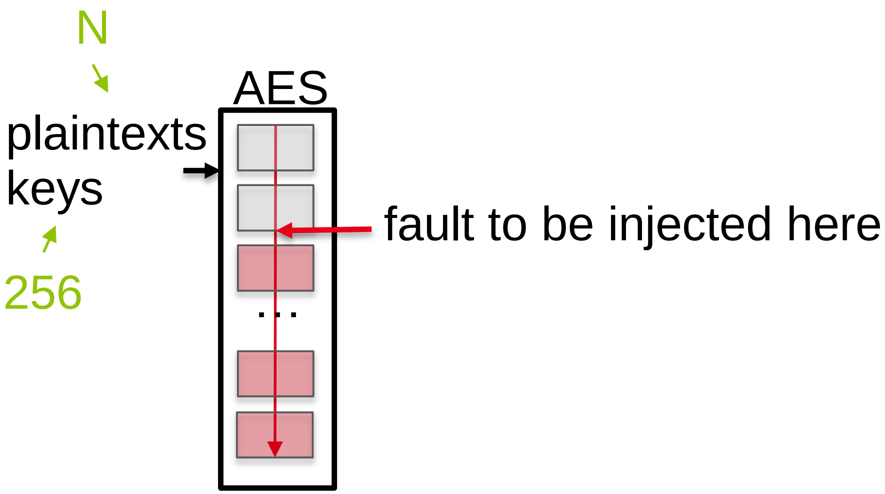
Light beam

Heating

…

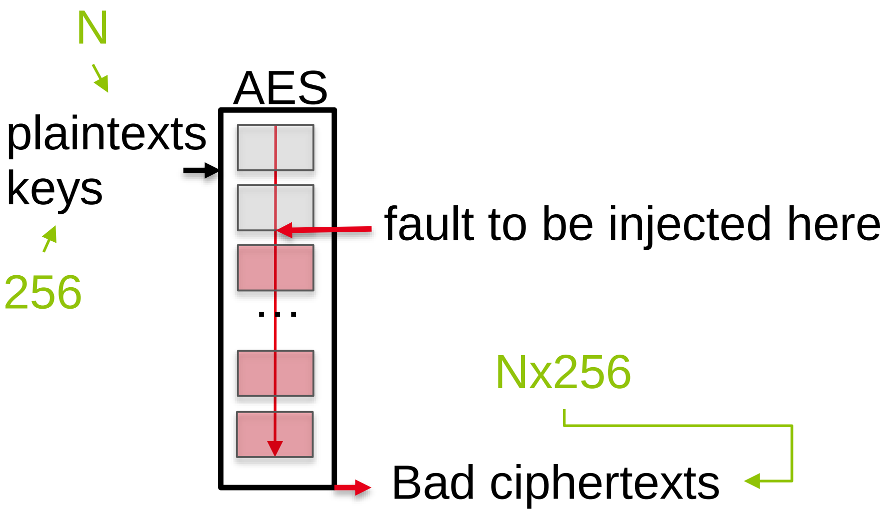
PHYSICAL ATTACKS: FAULT INJECTION ATTACKS

- **General approach:**
  - Divide and conquer: the key is recovered bit by bit or byte by byte
  - Perform a fault during encryption
  - The encryption will generate a bad ciphertext
  - Compare the bad ciphertext with the reference one
- **Attack steps:**
  - Choose a target instruction or data



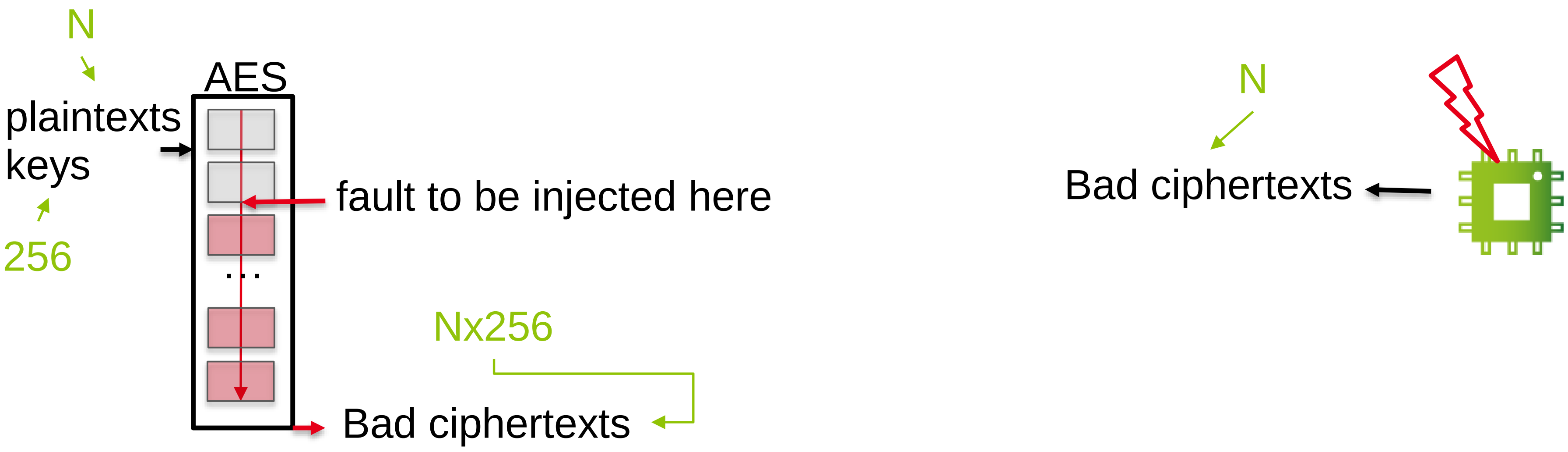
PHYSICAL ATTACKS: FAULT INJECTION ATTACKS

- **General approach:**
  - Divide and conquer: the key is recovered bit by bit or byte by byte
  - Perform a fault during encryption
  - The encryption will generate a bad ciphertext
  - Compare the bad ciphertext with the reference one
- **Attack steps:**
  - Choose a target instruction or data
  - Compute the effect of the fault for all keys and plaintexts on the ciphertext
    - Use a model of the fault like instruction skip or data nullified



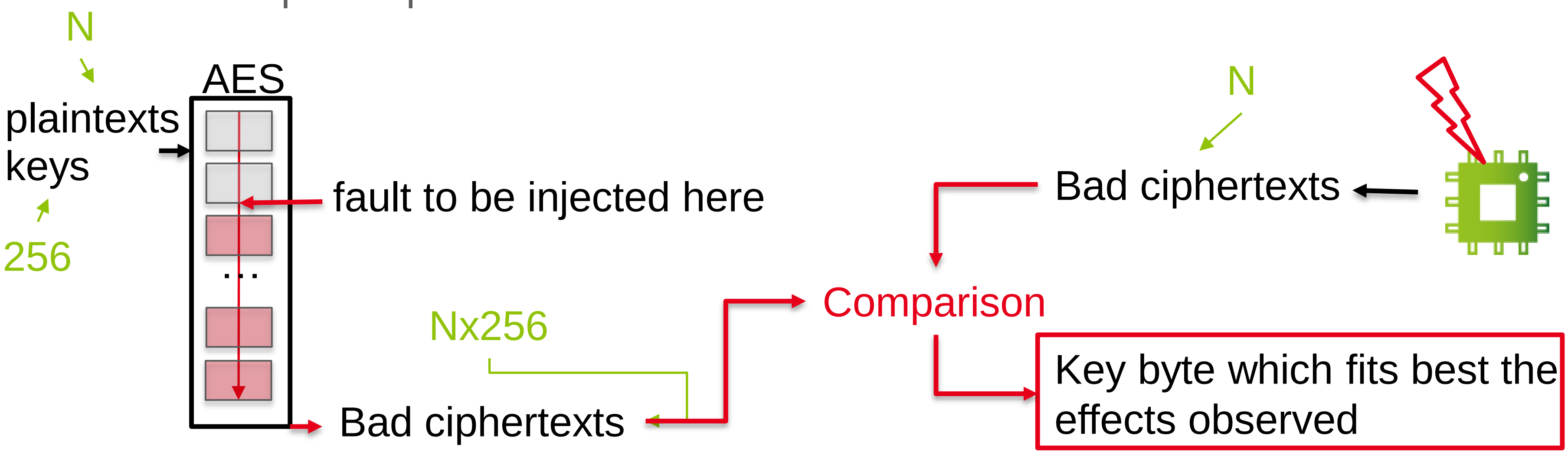
PHYSICAL ATTACKS: FAULT INJECTION ATTACKS

- **General approach:**
  - Divide and conquer: the key is recovered bit by bit or byte by byte
  - Perform a fault during encryption
  - The encryption will generate a bad ciphertext
  - Compare the bad ciphertext with the reference one
- **Attack steps:**
  - Choose a target instruction or data
  - Compute the effect of the fault for all keys and plaintexts on the ciphertext
  - Collect the ciphertexts for all plaintexts while faulting the chip



PHYSICAL ATTACKS: FAULT INJECTION ATTACKS

- **General approach:**
  - Divide and conquer: the key is recovered bit by bit or byte by byte
  - Perform a fault during encryption
  - The encryption will generate a bad ciphertext
  - Compare the bad ciphertext with the reference one
- **Attack steps:**
  - Choose a target instruction or data
  - Compute the effect of the fault for all keys and plaintexts on the ciphertext
  - Collect the ciphertexts for all plaintexts while faulting the chip
  - Compare ciphertexts obtained with the theoretical ones

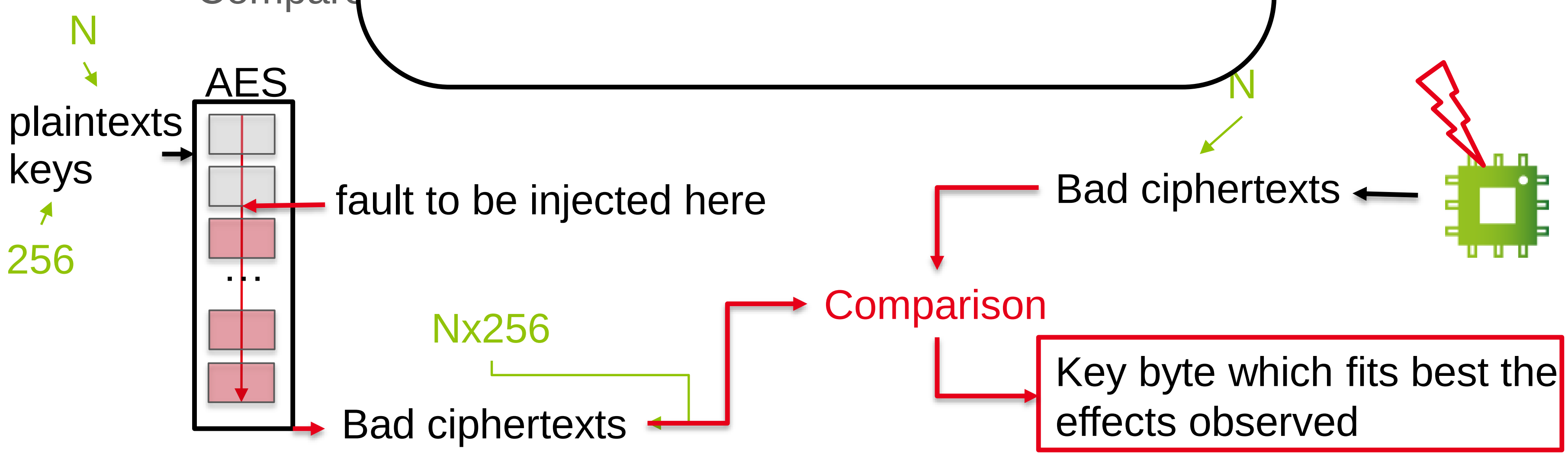


PHYSICAL ATTACKS: FAULT INJECTION ATTACKS

- **General approach:**
  - Divide an… by byte
  - Perform
  - The encr
  - Compare
- **Attack step**
  - Choose
  - Compute … the ciphertext
  - Collect th… ip
  - Compare

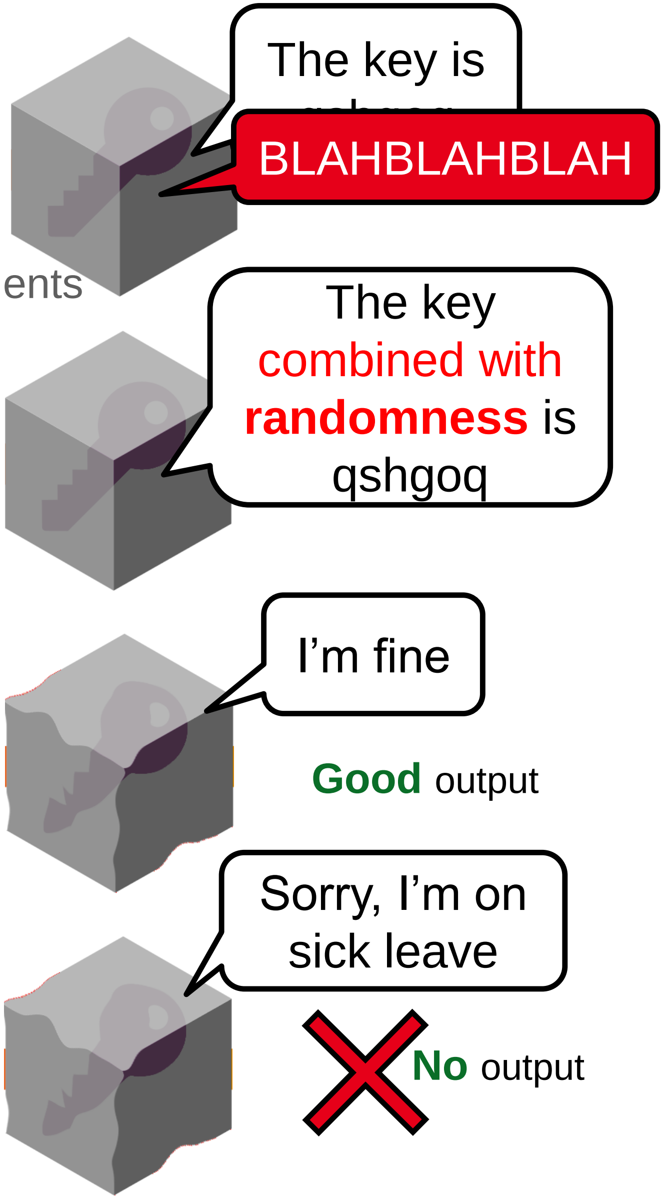
This is an example of how fault injections can be used.

BUT: they can target other kind of applications! (bootloaders, verifypin, …)



COUNTERMEASURES

- **Side-channel:**
  - Hiding
    - Lower the SNR (Signal Noise Ratio) in measurements
  - Masking
    - Break the direct link between emissions and the key
- **Fault injection attacks:**
  - Fault tolerance
    - A fault won’t change the behavior of the program
  - Fault detection
    - A fault will be detected and put the program/chip in a predefined state

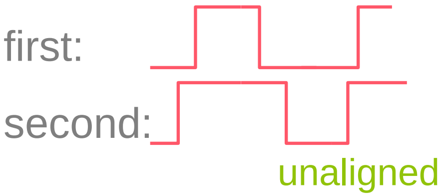


HARDWARE COUNTERMEASURES

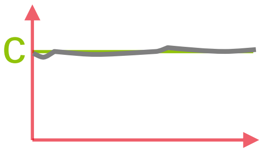
- **Side-channel:**
  - Dual rail with precharge logic
    - 0 and 1 are encoded with (0,1) and (1,0) couples
    - Output of each gate is precharged with either (0,0) or (1,1)
    - Hamming weight and Hamming distance are independent of data

HW(01)=1
HW(10)=1 } =
HD(00,01)=1
HD(00,10)=1 } =

  - Insert noise
    - Random voltage scaling
    - Variable clock speed (temporal desynchronization)



  - Filter power consumption
    - Make the power consumption as constant as possible

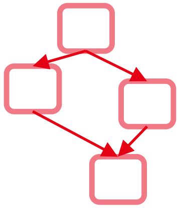


HARDWARE COUNTERMEASURES

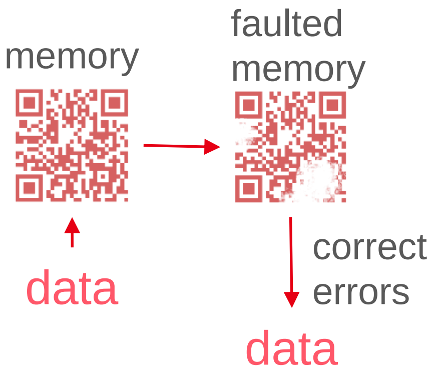
- **Fault injection attacks:**
  - Encapsulation
    - Prevent the attack by making the access to components hard

  - Detector of light emission / magnetic field
    - Detect signals which may be related to a fault injection

  - Integrity
    - Check the absence of control flow corruption (CFI)
    - Check data integrity



  - Error correcting memory
    - The memory is able to correct a certain number of errors in the data



HARDWARE COUNTERMEASURES

- **Side-channel:**
  - Dual rail with precharge logic
  - Insert noise
  - Filter power consumption

- **Fault injection attacks:**
  - Encapsulation
  - Detector of light emission / magnetic field
  - Control flow integrity
  - Error correcting memory

- **Problems / Limitations:**
  - Requires expertise
  - Takes time to implement
  - Costly hardware
  - Impossible to update
  - Countermeasure is applied everywhere, even on uncritical code

SOFTWARE COUNTERMEASURES

- **Side-channel:**
  - Instructions shuffling & Temporal desynchronization
    - Make alignment of measurements fail
    - Dependency analysis between instructions based on registers used or defined

```
for (i=0; i<n; i++) {
   k = rand(possible_values);
   T[k]=T[k]+1;
}
```

iterate in random order

```
if (rand(2)) {      →   asm {
                          add r3, r3, #1
                          sub r6, r7, #3
                        }
} else {            →   asm {
                          sub r6, r7, #3
                          add r3, r3, #1
}                       }
```

choose randomly at runtime
between the 2 forms

  - Masking
    - Combine the key with a random number to change the profile of the leakage
    - All the algorithm is modified so that everything is computed using the masked key

```
mask = rand();
masked_key = key xor mask;
```

```
a = a xor key;
b = a;
return b;
```

→

```
a = a xor masked_key;
b = a;
return b xor mask;
```

everything is computed masked
the mask is removed from the result at the end

SOFTWARE COUNTERMEASURES

- **Fault injection attacks:**
  - Code duplication
    - Some parts of the code are duplicated / Duplication of all instructions
    - Tolerance of one instruction-skip fault

```
if (password == "ok") {
    if (password == "ok") { … }
}
```

duplicate code

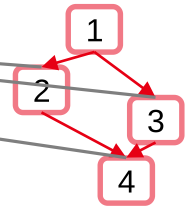
add r3, r4, #1 ⇔ add r3, r4, #1
add r3, r4, #1

duplicate instructions

  - Control flow integrity
    - At each basic block, check that we come from a legitimate basic block
    - Detection of instruction-modification fault that change the control flow

check that we come from 1

check that we come from 2 or 3



  - Error detecting codes throughout the algorithms
    - Add a parity bit to the variables and keep trace of it
    - Detection of data-corruption fault

011001010 → Ok

011001011 → Error !

SOFTWARE COUNTERMEASURES

- **Side-channel:**
  - Instructions shuffling & Temporal desynchronization
  - Masking
- **Fault injection attacks:**
  - Code duplication
  - Control flow integrity
  - Error detecting codes throughout the algorithms
- **Problems:**
  - Requires expertise
  - Takes time to implement
  - Implementation on every critical functions
  - Compilation can optimize out countermeasures
  - Performance cost

SOFTWARE COUNTERMEASURES

- **Side-channel:**
  - Instructions shuffling & Temporal desynchronization
  - Masking

- **Fault injection attacks:**
  - Code duplication
  - Control flow integrity
  - Error detecting codes throughout the algorithms

- **Problems:**
  - Requires expertise
  - Takes time to implement
  - Implementation on every critical functions
  - Compilation can optimize out countermeasures
  - Performance cost

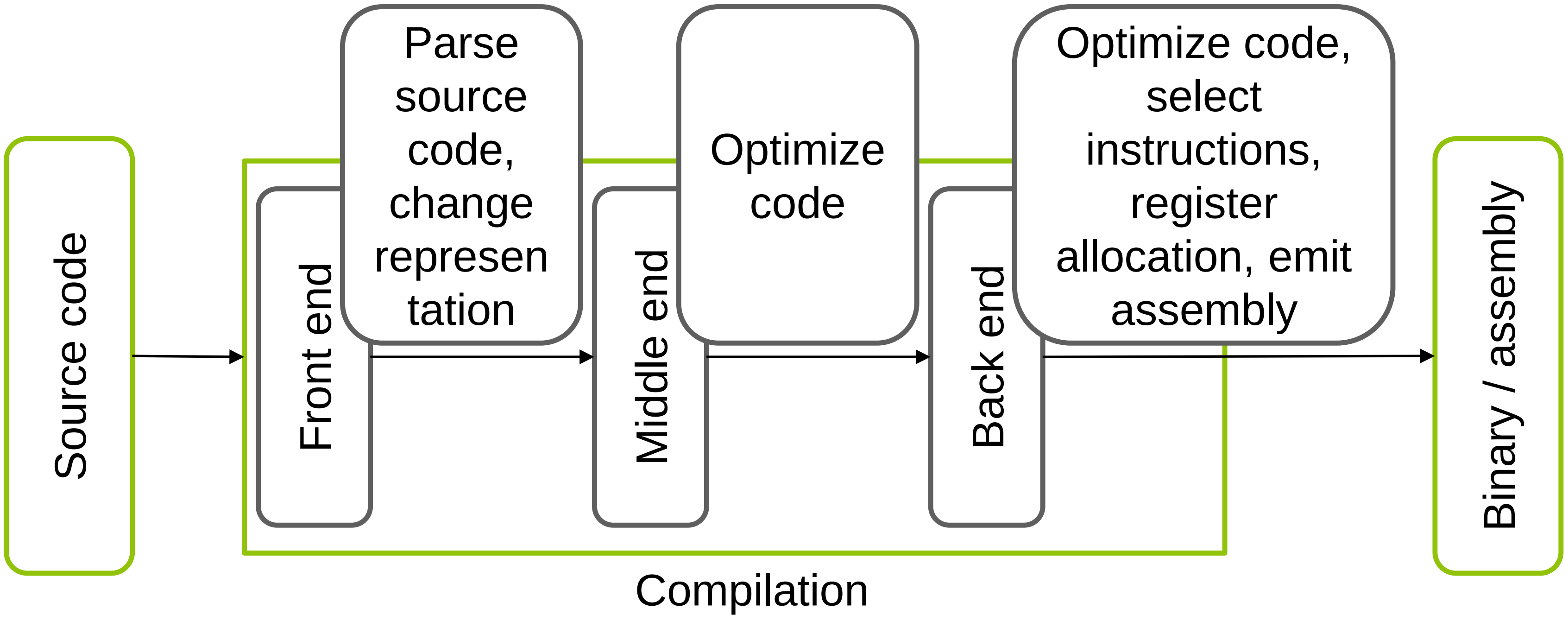
Automatically apply them ?

SOFTWARE COUNTERMEASURES

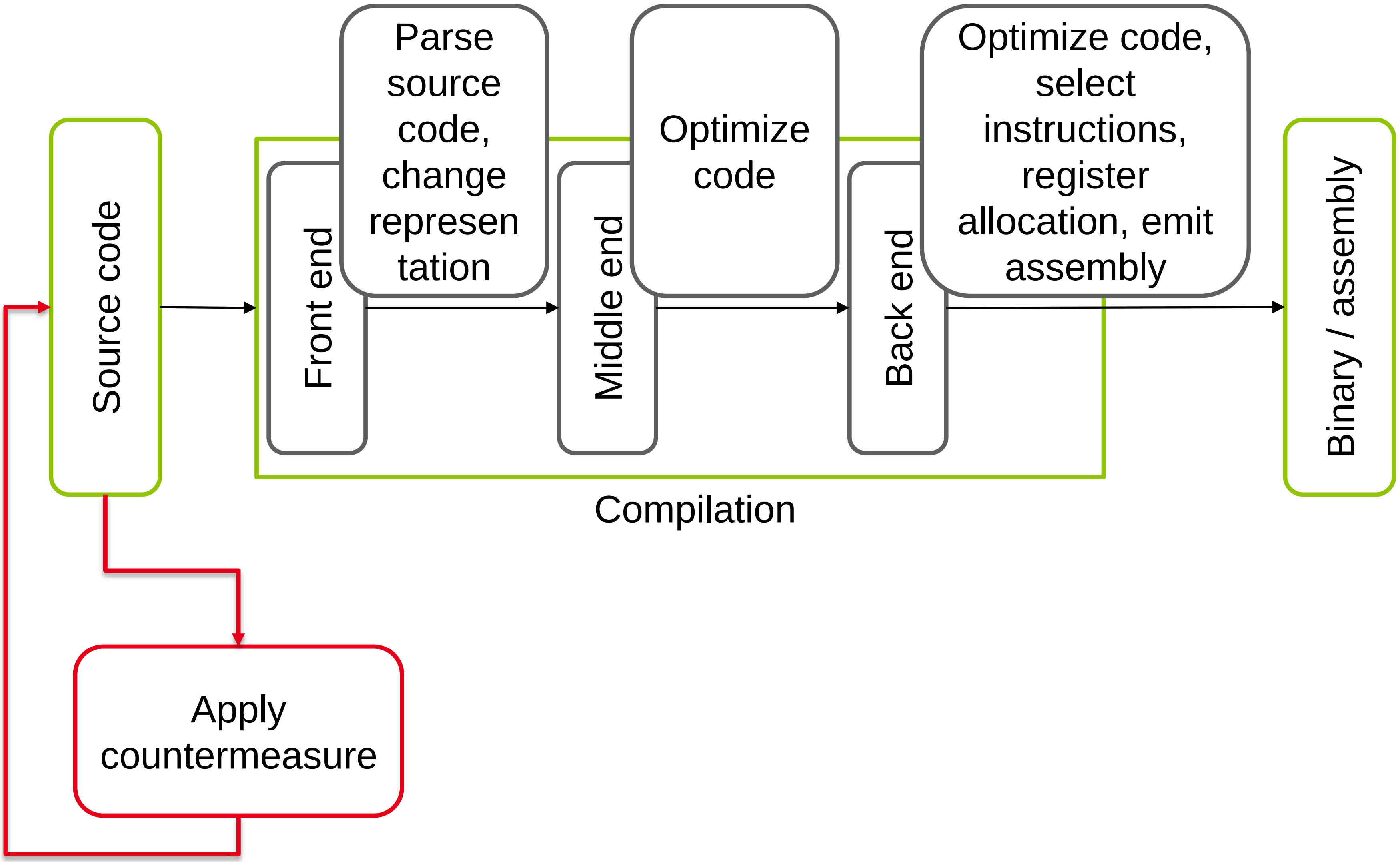
- **Side-channel:**
  - Instructions shuffling & Temporal desynchronization
  - Masking
- **Fault injection attacks:**
  - Code duplication
  - Control flow integrity
  - Error detecting codes throughout the algorithms
- **Problems:**
  - Requires expertise
  - Takes time to implement
  - Implementation on every critical functions
  - Compilation can optimize out countermeasures
  - Performance cost

Automatically apply them ?
HOW ?

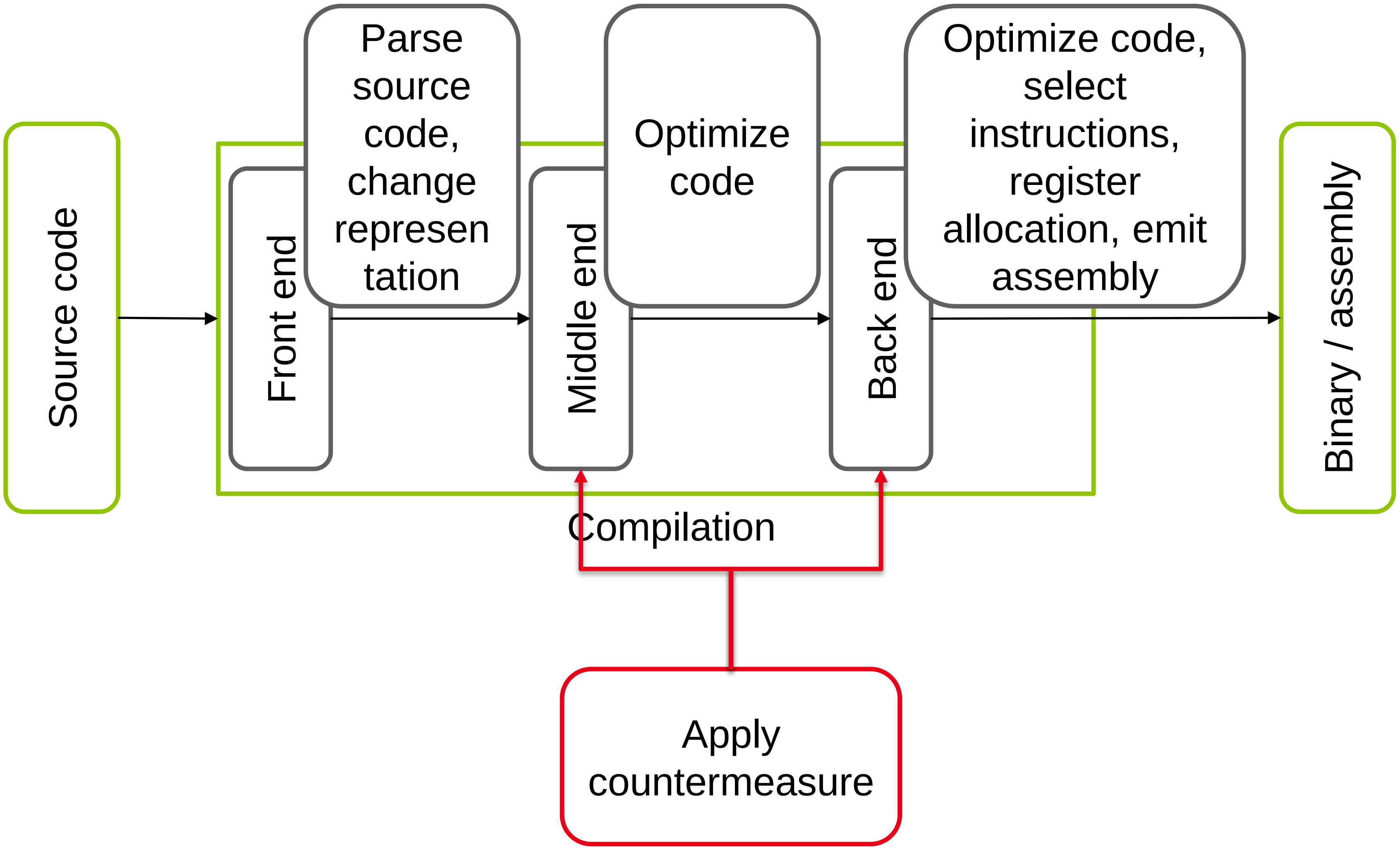
DIFFERENT LEVELS OF APPLICATION



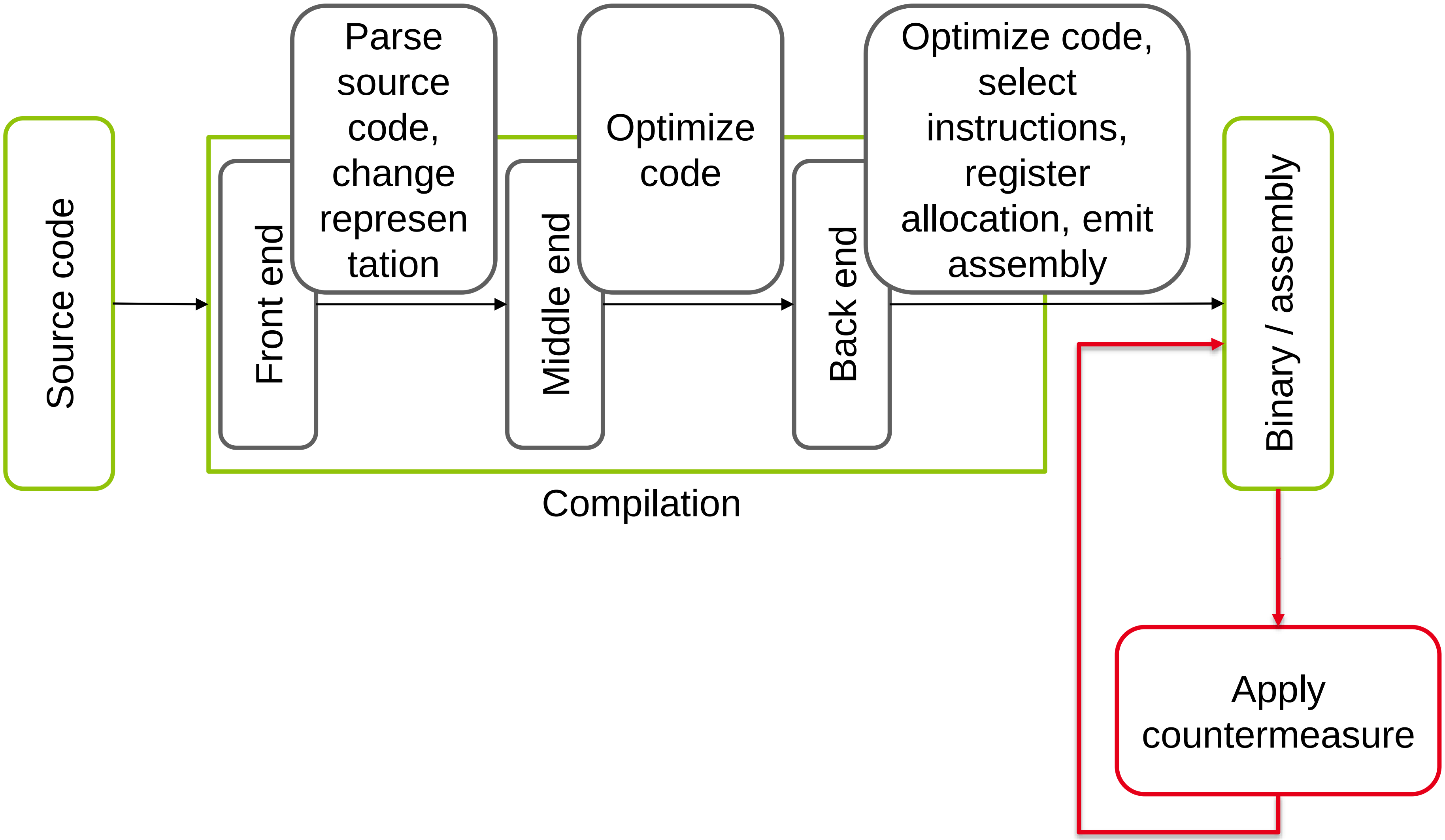
DIFFERENT LEVELS OF APPLICATION



DIFFERENT LEVELS OF APPLICATION



DIFFERENT LEVELS OF APPLICATION



COUNTERMEASURES: HOW TO APPLY THEM ?
→ SOURCE CODE

- **Steps to do once:**
  - Write a parser
  - Write a transformation pass for critical parts
  - Write a file emitter for targeted format

- **Steps to do for every file:**
  - Transform file
  - Compile file
  - Disassemble file
  - **Check that countermeasures are still here**

- **Disabling compiler optimizations (-O0) to skip the checking phase is a bad idea**
  - Horrible performance
  - Register spilling → new leakage

- **References that use this approach:** [Eldib, LNCS, 2014] [Lalande, LNCS, 2014] [Luo, ASAP, 2015]

COUNTERMEASURES: HOW TO APPLY THEM ?
→ WITHIN THE COMPILER

- **Steps to do once:**
  - Update the parser
  - Add a transformation pass to transform critical parts
  - Check once for all that later transformations do not threaten the countermeasure
  - If necessary, deactivate or transform some of them

- **Steps to do for every file:**
  - Compile file } no need to be a security expert here

- **The code resulting is correctly optimized**

- **References that use this approach:** [Agosta, IEEE TCAD, 2015] [Agosta, DAC, 2012] [Agosta, DAC, 2013] [Barry, CS2, 2016] [Bayrak, IEEE TC, 2015] [Malagón, Sensors, 2012] [Moss, LNCS, 2012]
  - [Bayrak, IEEE TC, 2015]: hybrid approach between the “assembly” and “within the compiler” approaches. Uses the compiler to **decompile** a binary file up to an intermediate representation before applying the countermeasure.

COUNTERMEASURES: HOW TO APPLY THEM ?
→ ASSEMBLY CODE

- **Steps to do once:**
  - Write a parser
  - Write analysis passes which reconstruct some higher level information if necessary
  - Write the transformation
  - Write a file emitter

- **Steps to do for every file:**
  - Compile the file
  - Disassemble it
  - Transform it
  - Reassemble it

  } no need to be a security expert here

- **The resulting code is secured but performance can be affected**
  - Compiler uses registers as if they won't be used for something else
  - The need for additional registers while applying countermeasure may lead to register spilling

- **References that use this approach:** [Bayrak, DAC, 2011] [Moro, 2014] [Rauzy, JCEN, 2016]

COUNTERMEASURES: HOW TO APPLY THEM ? → DETAILED EXAMPLES

Level	Team	Approach
Source code	Lalande & al. Eldib & al.	• CFI applied on C code • Use clang as a parser and apply Masking with a SMT solver
Within the compiler	Agosta & al. Barry & al.	• Modified LLVM (new passes & modified passes). Hiding applied automatically. • Modified LLVM (new passes & modified passes). Instruction Duplication applied automatically.
Assembly code	Bayrak & al. Moro & al.	• Random precharging applied automatically. • Instruction Duplication applied automatically.

- J.-F. Lalande, K. Heydemann, and P. Berthomé. Software Countermeasures for Control Flow Integrity of Smart Card C Codes. In European Symposium on Research in Computer Security, pages 200–218. Springer, 2014.
- H. Eldib and C. Wang. Synthesis of Masking Countermeasures Against Side Channel Attacks. In International Conference on Computer Aided Verification, pages 114–130. Springer, 2014.
- G. Agosta, A. Barenghi, G. Pelosi, and M. Scandale. The MEET Approach: Securing Cryptographic Embedded Software Against Side Channel Attacks. IEEE TCAD, 34(8):1320–1333, 2015.
- T. Barry, D. Couroussé, and B. Robisson. Compilation of a Countermeasure Against Instruction-Skip Fault Attacks. In Proceedings of the Third Workshop on Cryptography and Security in Computing Systems, pages 1–6. ACM, 2016.
- A. Bayrak, F. Regazzoni, P. Brisk, F.-X. Standaert, and P. Ienne. A first step towards automatic application of power analysis countermeasures. pages 230–235, 2011.
- N. Moro, K. Heydemann, E. Encrenaz, and B. Robisson. Formal Verification of a Software Countermeasure Against Instruction Skip Attacks. Journal of Cryptographic Engineering, 4(3):145–156, 2014.

COUNTERMEASURES: HOW TO APPLY THEM ? → DETAILED EXAMPLES

Level	Team	Approach
Source code	Lalande & al. Eldib & al.	• CFI applied on C code • Use clang as a parser and apply Masking with a SMT solver
Within the compiler	Agosta & al. **Barry & al.**	• Modified LLVM (new passes & modified passes). Hiding applied automatically. • Modified LLVM (new passes & modified passes). Instruction Duplication applied automatically.
Assembly code	Bayrak & al. **Moro & al.**	• Random precharging applied automatically. • Instruction Duplication applied automatically.

- J.-F. Lalande, K. Heydemann, and P. Berthomé. Software Countermeasures for Control Flow Integrity of Smart Card C Codes. In European Symposium on Research in Computer Security, pages 200–218. Springer, 2014.
- H. Eldib ... Confer...
- G. Ago ... Software Agains...
- T. Barr ... cks. In Procee ... 016.
- A. Bay ... ower analysis...
- N. Moro, K. Heydemann, E. Encrenaz, and B. Robisson. Formal Verification of a Software Countermeasure Against Instruction Skip Attacks. Journal of Cryptographic Engineering, 4(3):145–156, 2014.

For the same countermeasure, compiler approach reduced performance overhead from **x2.86** to **x1.92** and size overhead from **x2.90** to **x1.16** for MiBench AES

COUNTERMEASURES: HOW TO APPLY THEM ?
→ IN A NUTSHELL

Level	Pros	Cons
Source code	• More or less straightforward	• Countermeasure can be optimized out during compilation • Assembly code MUST be checked after compilation
Within the compiler	• Provide security AND performance • Optimizations can be controlled	• Harder to implement. • Requires to have access to the compiler source code
Assembly code	• Countermeasure not optimized out • Can even secure binary programs without their source code	• Can be hard to take all instructions into account or to do high level transformations • Performance more affected

CONCLUSION

- **Physical attacks are an important threat for cyber-physical systems**
  - They are the only effective way to break encryption
  - Their range of target is broader than encryption
  - Best security levels are reached by **combining hardware and software** countermeasures
- **Securing is costly**
  - Automatic application of **software** countermeasures or automatic design of **hardware** with countermeasures can reduce this cost
- **Compilation is usually forgotten in potential threats to countermeasures**
  - source code ≠ binary
- **Securing during compilation is valuable**
  - Enables to optimize the performance cost of a countermeasure
- **Hardware has to be taken into account too**
  - binary ≠ what is really executed
  - Speculative execution within the processor

CONCLUSION

- **Physical attacks are an important threat for cyber-physical systems**
  - They are the only effective way to break encryption
  - Their range of target is broader than encryption
  - Best security levels are reached by **combining hardware and software** countermeasures
- **Securing is costly**
  - Automatic application of **software** countermeasures or automatic design of **hardware** with countermeasures can reduce this cost
- **Compilation is usually forgotten in potential threats to countermeasures**
  - source code ≠ binary
- **Securing during compilation is valuable**
  - Enables to optimize the performance cost of a cour
- **Hardware has to be taken into account too**
  - binary ≠ what is really executed
  - Speculative execution within the processor

Pay attention to these!

REFERENCES

- [Bayrak, DAC, 2011] A. G. Bayrak, F. Regazzoni, P. Brisk, F.-X. Standaert, and P. Ienne, 'A first step towards automatic application of power analysis countermeasures', presented at the Proceedings - Design Automation Conference, 2011, pp. 230–235.
- [Moro, 2014] N. Moro, 'Security of assembly programs against fault attacks on embedded processors', Theses, Université Pierre et Marie Curie - Paris VI, 2014.
- [Rauzy, JCEN, 2016] P. Rauzy, S. Guilley, and Z. Najm, 'Formally proved security of assembly code against power analysis: A case study on balanced logic', Journal of Cryptographic Engineering, vol. 6, no. 3, pp. 201–216, 2016.
- [Agosta, IEEE TCAD, 2015] G. Agosta, A. Barenghi, G. Pelosi, and M. Scandale, 'The MEET Approach: Securing Cryptographic Embedded Software Against Side Channel Attacks', IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 8, pp. 1320–1333, Aug. 2015.
- [Agosta, DAC, 2012] G. Agosta, A. Barenghi, and G. Pelosi, 'A code morphing methodology to automate power analysis countermeasures', in DAC Design Automation Conference 2012, 2012, pp. 77–82.
- [Agosta, DAC, 2013] G. Agosta, A. Barenghi, M. Maggi, and G. Pelosi, 'Compiler-based side channel vulnerability analysis and optimized countermeasures application', in Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE, 2013, pp. 1–6.
- [Barry, CS2, 2016] T. Barry, D. Couroussé, and B. Robisson, 'Compilation of a Countermeasure Against Instruction-Skip Fault Attacks', in Workshop on Cryptography and Security in Computing Systems, vienna, Austria, 2016.
- [Bayrak, IEEE TC, 2015] A. G. Bayrak, F. Regazzoni, D. Novo, P. Brisk, F.-X. Standaert, and P. Ienne, 'Automatic application of power analysis countermeasures', IEEE Transactions on Computers, vol. 64, no. 2, pp. 329–341, 2015.
- [Malagón, Sensors, 2012] P. Malagón, G. de, M. Zapater, J. M. Moya, and Z. Banković, 'Compiler optimizations as a countermeasure against side-channel analysis in MSP430-based devices', Sensors (Switzerland), vol. 12, no. 6, pp. 7994–8012, 2012.

REFERENCES

- [Moss, LNCS, 2012] A. Moss, E. Oswald, D. Page, and M. Tunstall, ‘Compiler assisted masking’, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 7428 LNCS, pp. 58–75, 2012.
- [Eldib, LNCS, 2014] H. Eldib and C. Wang, ‘Synthesis of masking countermeasures against side channel attacks’, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 8559 LNCS, pp. 114–130, 2014.
- [Lalande, LNCS, 2014] J.-F. Lalande, K. Heydemann, and P. Berthomé, ‘Software countermeasures for control flow integrity of smart card c codes’, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 8713 LNCS, no. PART 2, pp. 200–218, 2014.
- [Luo, ASAP, 2015] P. Luo, L. Zhang, Y. Fei, and A. A. Ding, ‘Towards secure cryptographic software implementation against side-channel power analysis attacks’, in Application-specific Systems, Architectures and Processors (ASAP), 2015 IEEE 26th International Conference on, 2015, pp. 144–148.

Thank you for your attention

Questions?

Contact:
nicolas.belleville@cea.fr

THE MULTIPLE WAYS TO AUTOMATE THE APPLICATION OF SOFTWARE COUNTERMEASURES AGAINST PHYSICAL ATTACKS: PITFALLS AND GUIDELINES

Belleville Nicolas [1]
Barry Thierno [1]
Seriai Abderrahmane [1]
Couroussé Damien [1]
Heydemann Karine [2]
Robisson Bruno [3]
Charles Henri-Pierre [1]

[1] Univ Grenoble Alpes, CEA, List, F-38000 Grenoble, France
firstname.lastname@cea.fr
[2] Sorbonne Universités, UPMC, Univ. Paris 06, CNRS,LIP6,UMR 7606 75005 Paris, France
firstname.lastname@lip6.fr
[3] CEA/EMSE, Secure Architectures and Systems Laboratory CMP, 880 Route de Mimet, 13541 Gardanne, France
firstname.lastname@cea.fr

Commissariat à l'énergie atomique et aux énergies alternatives
17 rue des Martyrs | 38054 Grenoble Cedex
www.cea-tech.fr

Établissement public à caractère industriel et commercial | RCS Paris B 775 685 019