Unboxing the White-Box

riscure

Who are we?

- All Principal Security Analyst @Riscure
- Cristofaro Mune
  - Keywords: Software, Reversing, Exploit, Fault Injection…
  - Previous work on Mobile and Embedded Exploitation
- Eloi Sanfelix
  - Keywords: Software security, RE, Exploiting, SCA/FI, CTF
- Job de Haas
  - Keywords: Embedded, Side Channel Analysis, Fault Injection
  - All-round from network pentester to SoC evaluator

What and why…

- White-Box cryptography → Protect keys in untrusted environment
- Increasingly relevant in security solutions
- The idea: Porting Hardware attacks to Software…

…***it works! Extremely effective approach***

- Relevant not only on WBC:
    - Potentially applicable to all Software-based crypto solution

Introduction

Key recovery attacks

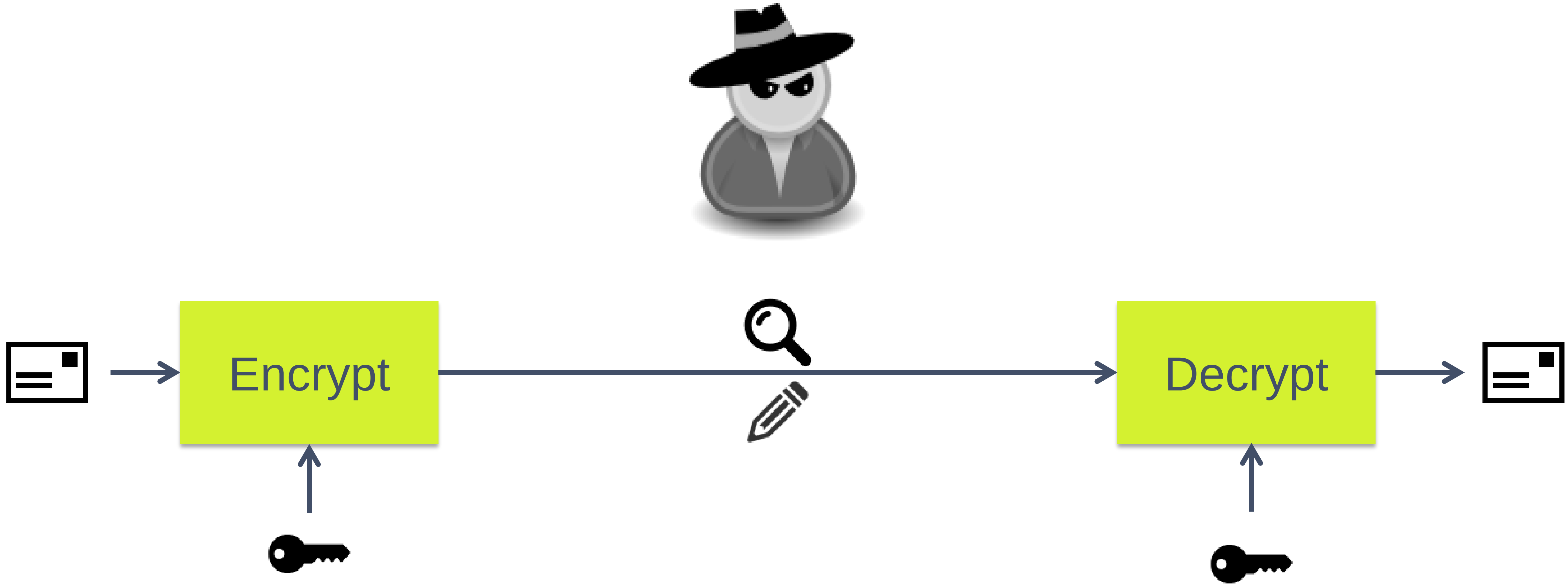
Conclusion

Introduction

Key recovery attacks

Conclusion

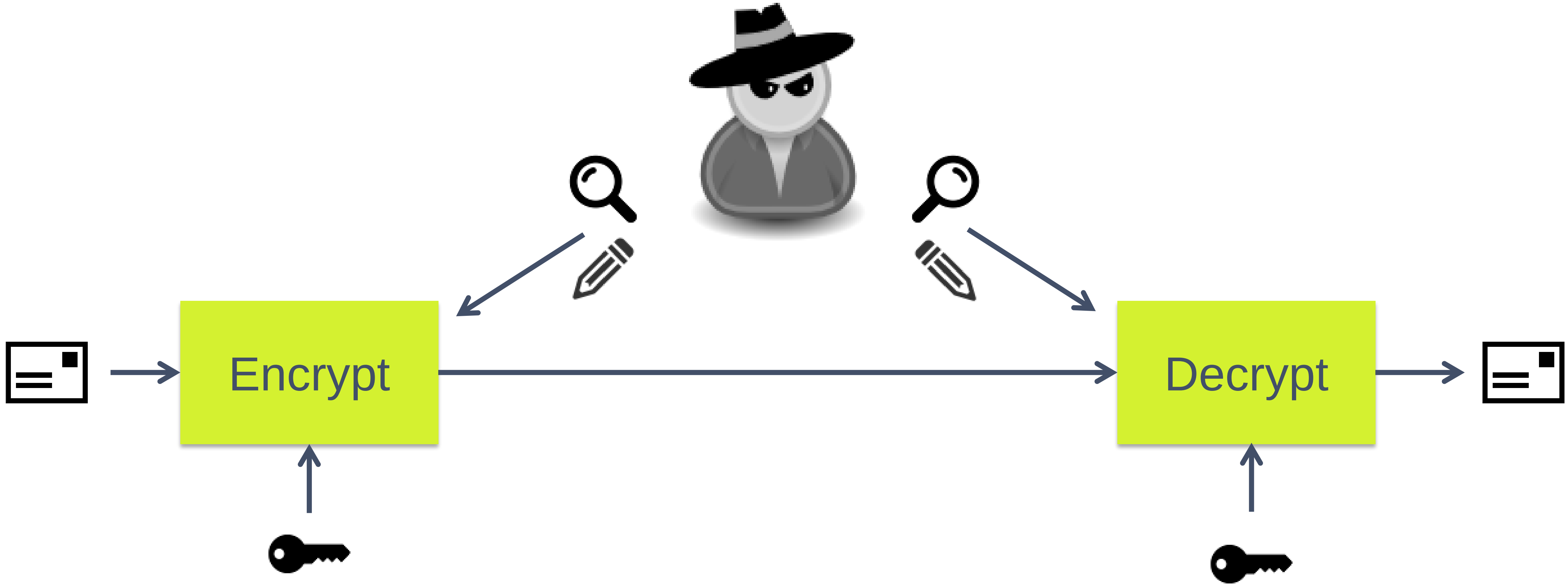
Black-Box Security

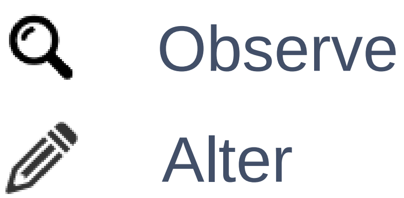


Observe

Alter

Gray-Box Security

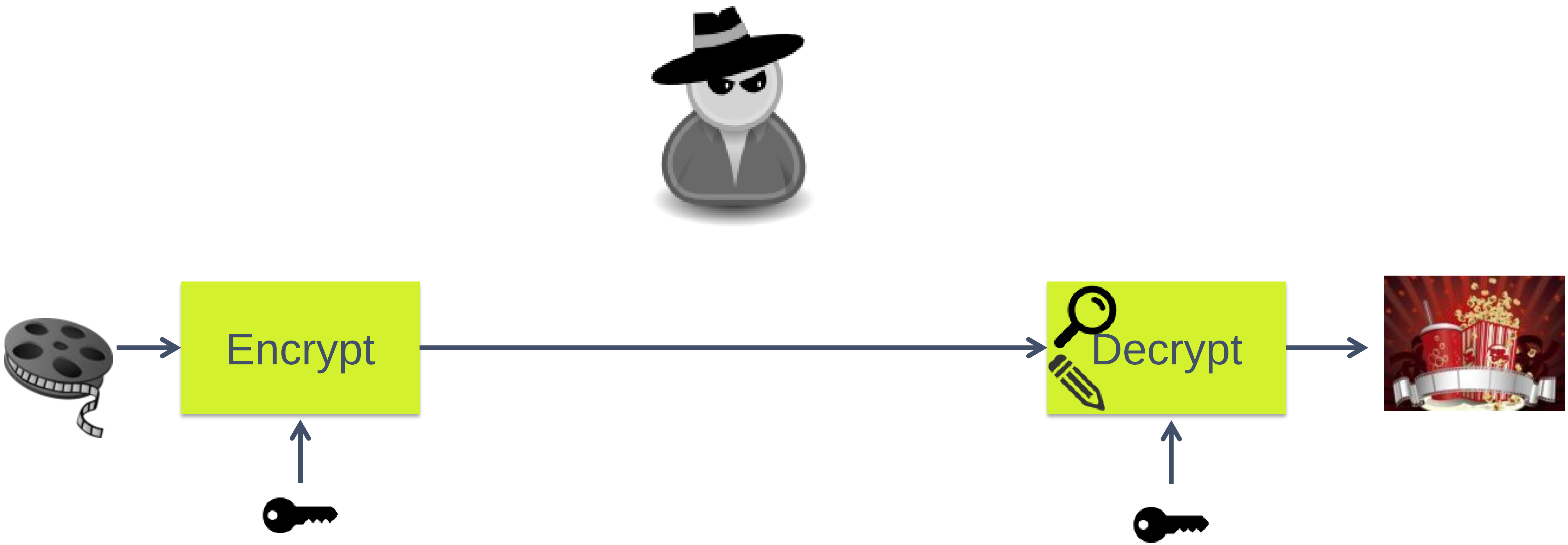




Sign of the times…

Sign of the times…

White-Box Security



Observe

Alter

White-Box Security

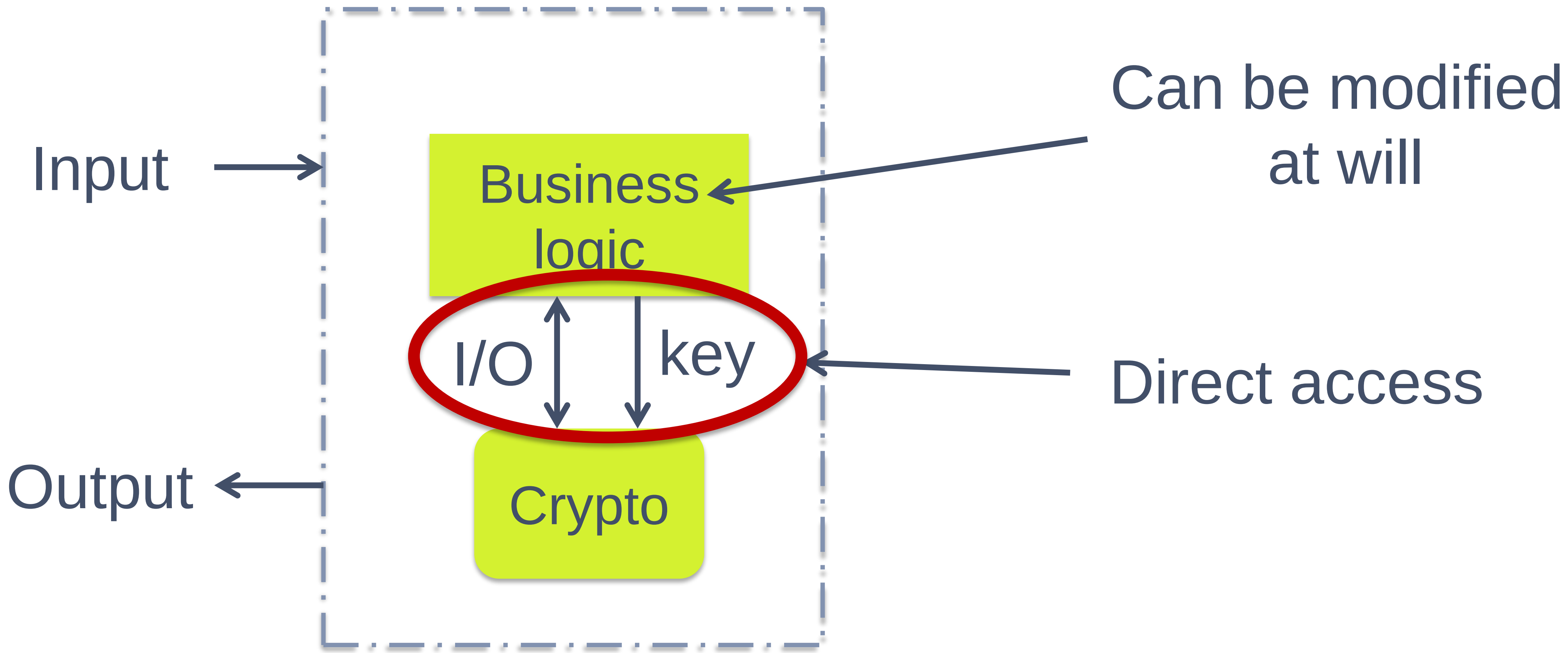
Observe

Alter

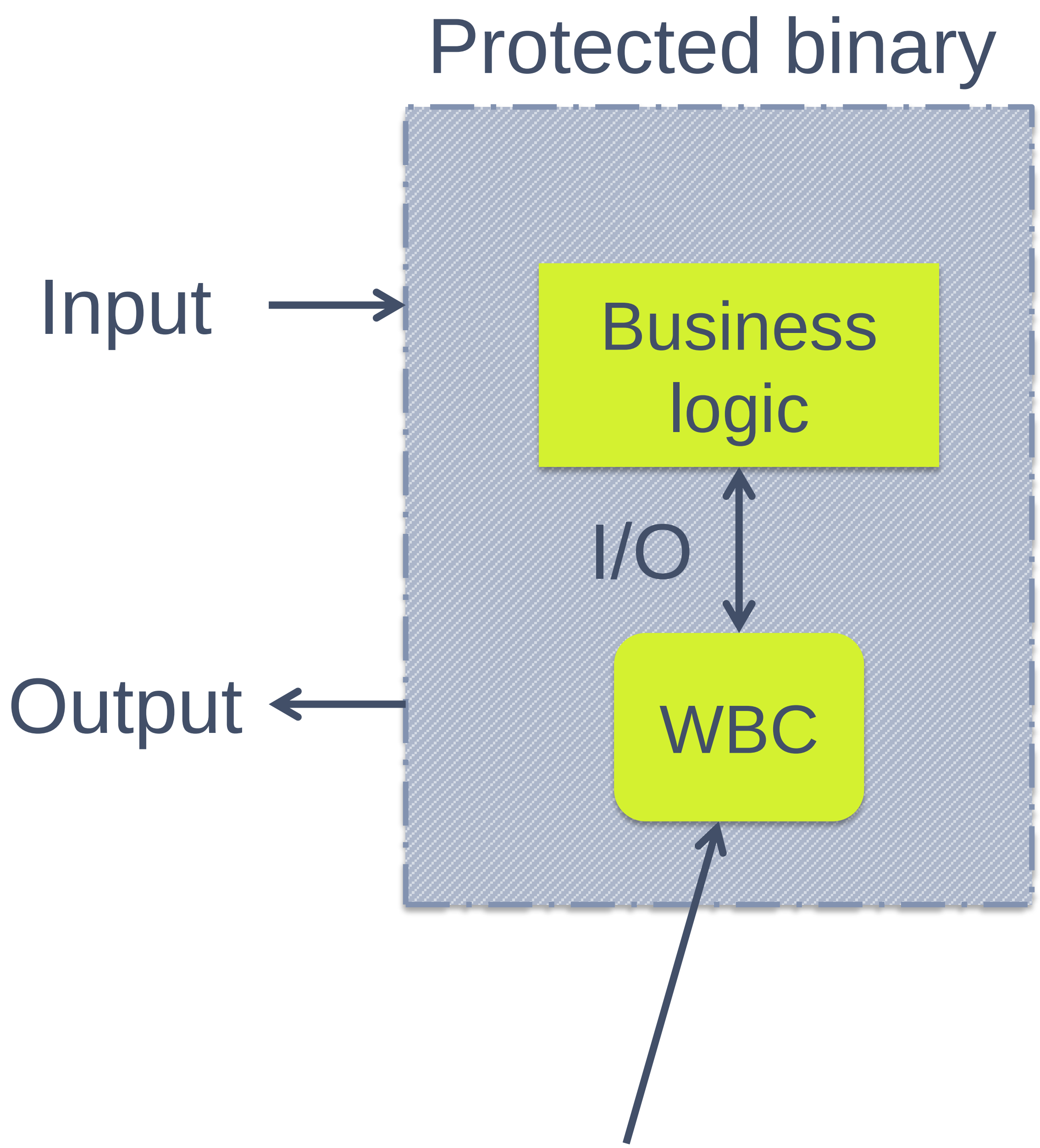
White-Box Cryptography

- Protection against **key extraction** in the white-box security model
- A technique that allows merging a key into a given crypto algorithm:
  - Described for the first time in 2002 by S. Chow et al.
  - Available for AES and DES
- Lookup tables used for applying mathematical transforms to data
- Remove the distinction between keys and crypto algorithm code.

Software in the White-Box context



Software Protection



Focus on this part only

Obfuscation

- Control-flow obfuscation
- Data obfuscation

Anti-analysis and anti-tamper

- Detect debugger/emulator
- Detect hooks and modifications

Device binding

- Bind code to current device

How does WBC work?

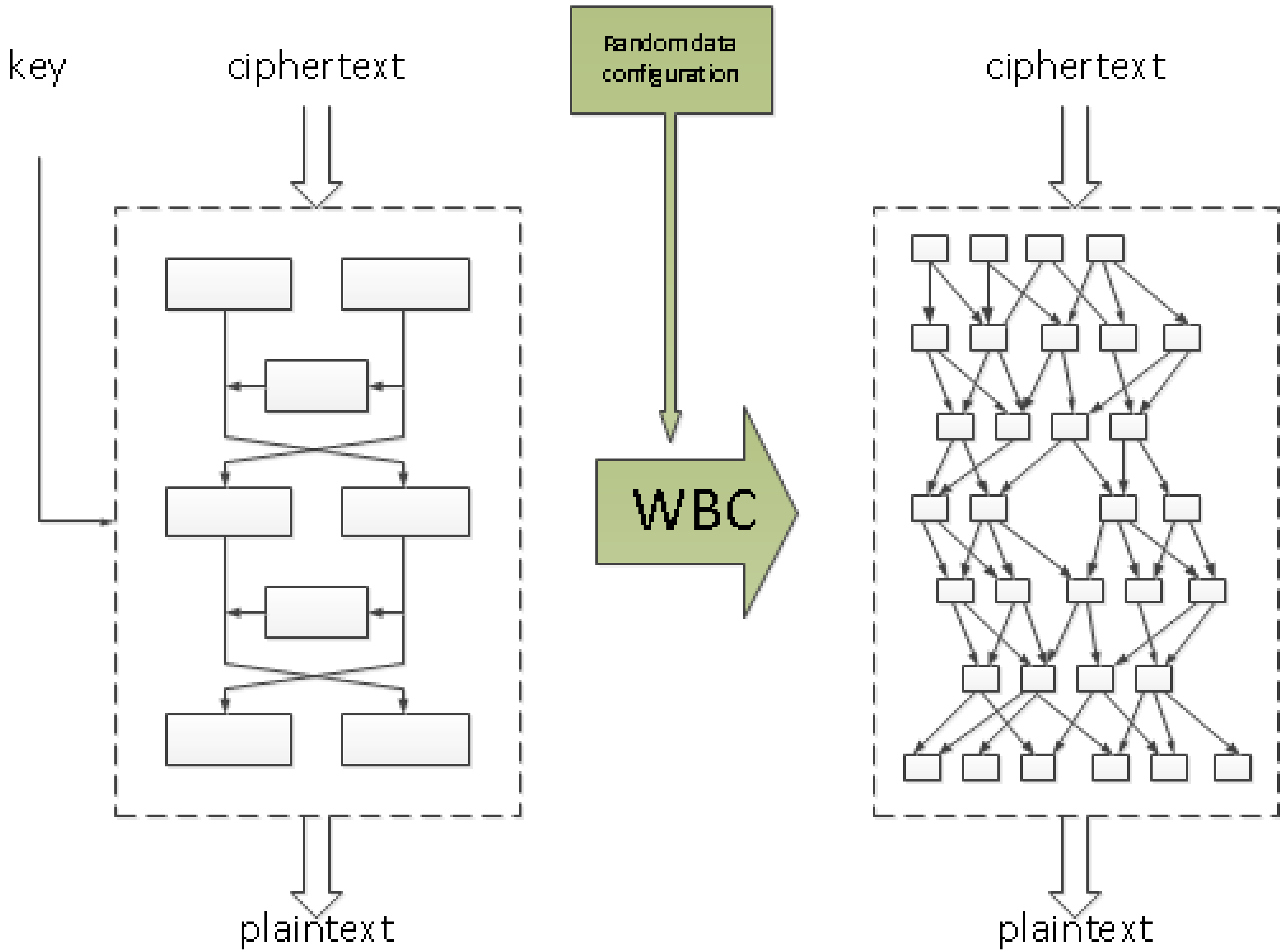
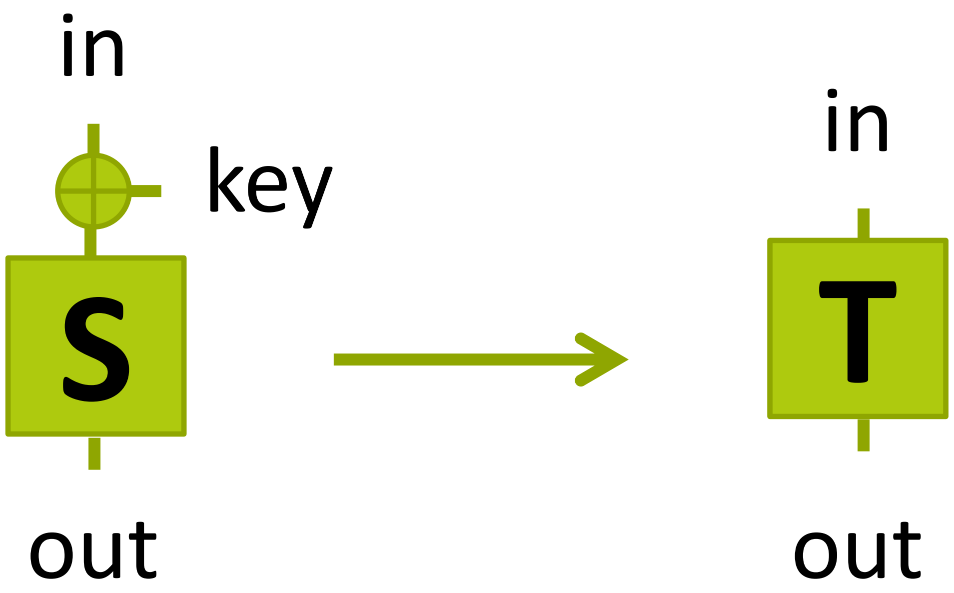


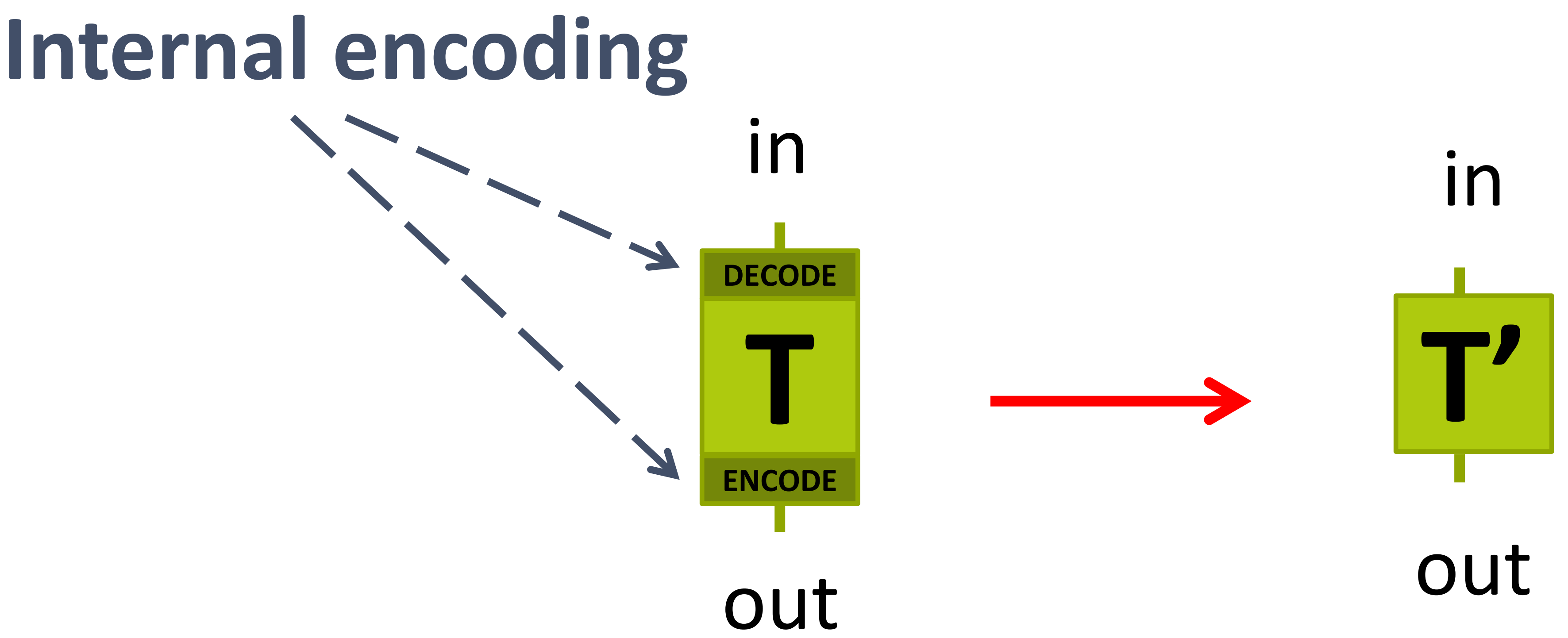
Image source: whiteboxcrypto.com

WBC Construction: partial evaluation




```python
def __init__(self, lut, key):

    self._lut = []

    for i in xrange(len(lut)):
        si = lut[i ^ key]
        self._lut.append(si)
```

WBC Construction: encoding




```
def __init__(self, lut, key, inbij, outbij):

    self._lut = []

    for i in xrange(len(lut)):
        ii = inbij.inv(i)
        si = lut[ii ^ key]
        msi = outbij.apply(si)
        self._lut.append(msi)
```

Example code

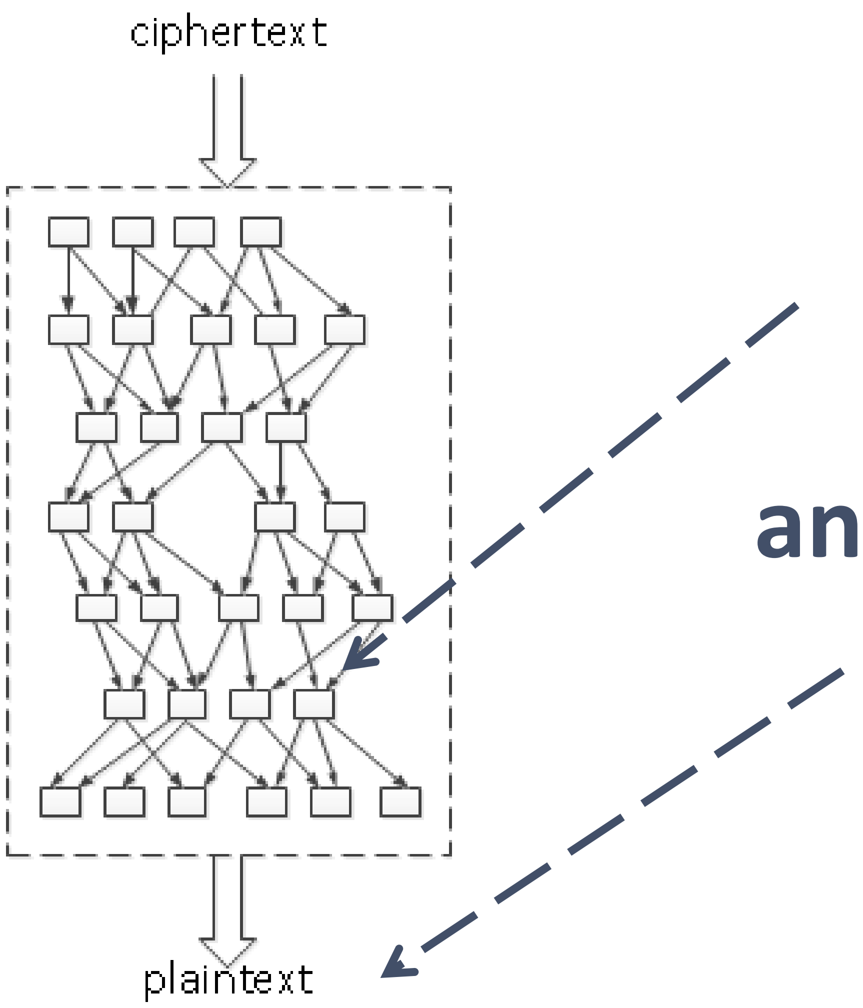
```c
void aes128_enc_wb_final(unsigned char in[16], unsigned char out[16])
{
    memcpy(out, in, 16);

    /// Let's start the encryption process now
    for (size_t i = 0; i < 9; ++i)
    {
        ShiftRows(out);

        for (size_t j = 0; j < 4; ++j)
        {
            unsigned int aa = Tyboxes[i][j * 4 + 0][out[j * 4 + 0]];
            unsigned int bb = Tyboxes[i][j * 4 + 1][out[j * 4 + 1]];
            unsigned int cc = Tyboxes[i][j * 4 + 2][out[j * 4 + 2]];
            unsigned int dd = Tyboxes[i][j * 4 + 3][out[j * 4 + 3]];

            out[j * 4 + 0] = (Txor[Txor[(aa >>  0) & 0xf][(bb >>  0) & 0xf]][Txor[(cc >>  0) & 0xf][(dd >>  0) & 0xf]]) |
                            ((Txor[Txor[(aa >>  4) & 0xf][(bb >>  4) & 0xf]][Txor[(cc >>  4) & 0xf][(dd >>  4) & 0xf]]) << 4);
            out[j * 4 + 1] = (Txor[Txor[(aa >>  8) & 0xf][(bb >>  8) & 0xf]][Txor[(cc >>  8) & 0xf][(dd >>  8) & 0xf]]) |
                            ((Txor[Txor[(aa >> 12) & 0xf][(bb >> 12) & 0xf]][Txor[(cc >> 12) & 0xf][(dd >> 12) & 0xf]]) << 4);
            out[j * 4 + 2] = (Txor[Txor[(aa >> 16) & 0xf][(bb >> 16) & 0xf]][Txor[(cc >> 16) & 0xf][(dd >> 16) & 0xf]]) |
                            ((Txor[Txor[(aa >> 20) & 0xf][(bb >> 20) & 0xf]][Txor[(cc >> 20) & 0xf][(dd >> 20) & 0xf]]) << 4);
            out[j * 4 + 3] = (Txor[Txor[(aa >> 24) & 0xf][(bb >> 24) & 0xf]][Txor[(cc >> 24) & 0xf][(dd >> 24) & 0xf]]) |
                            ((Txor[Txor[(aa >> 28) & 0xf][(bb >> 28) & 0xf]][Txor[(cc >> 28) & 0xf][(dd >> 28) & 0xf]]) << 4);
        }
    }

    /// Last round which is a bit different
    ShiftRows(out);

    for (size_t j = 0; j < 16; ++j)
    {
        unsigned char x = Tboxes_[j][out[j]];
        out[j] = x;
    }
}
```

Source: https://doar-e.github.io/

External encoding

Potential attacks on WBC (I)

Data manipulation – Fault Injection (FI)

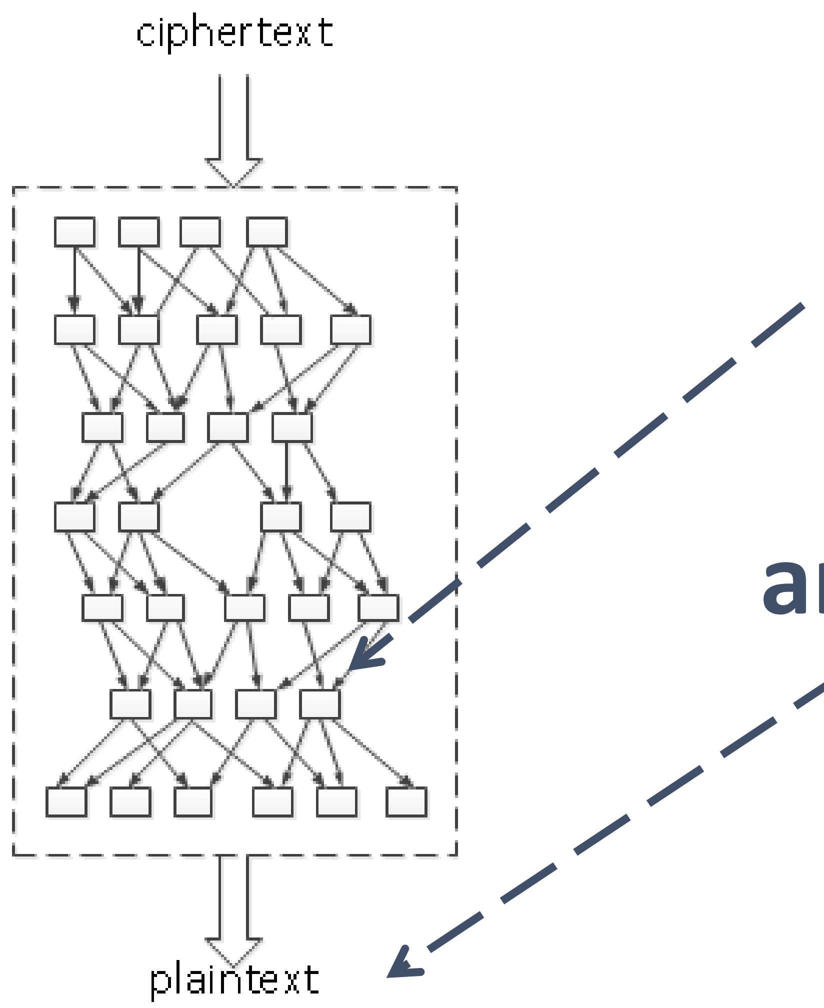


Modify data here

and observe changes to the output

Potential attacks on WBC (II)

Process manipulation – Fault injection (FI)

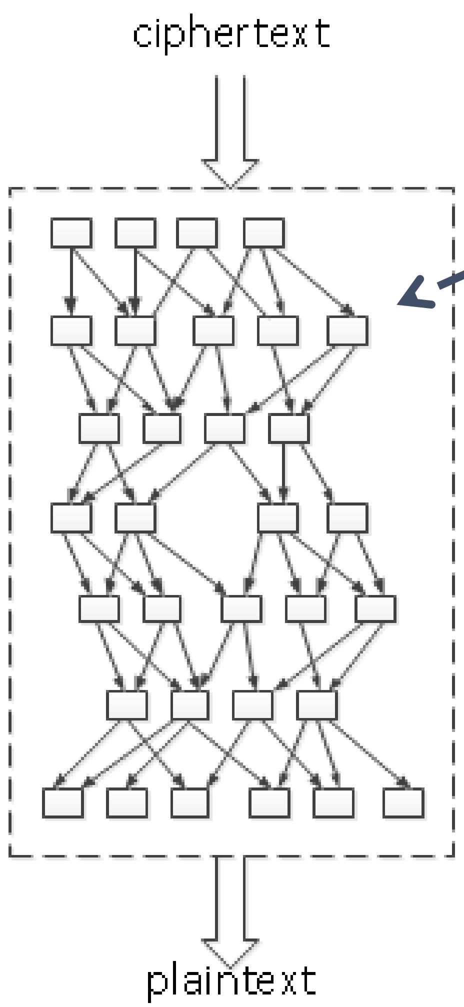


Modify code flow here

and observe changes to the output

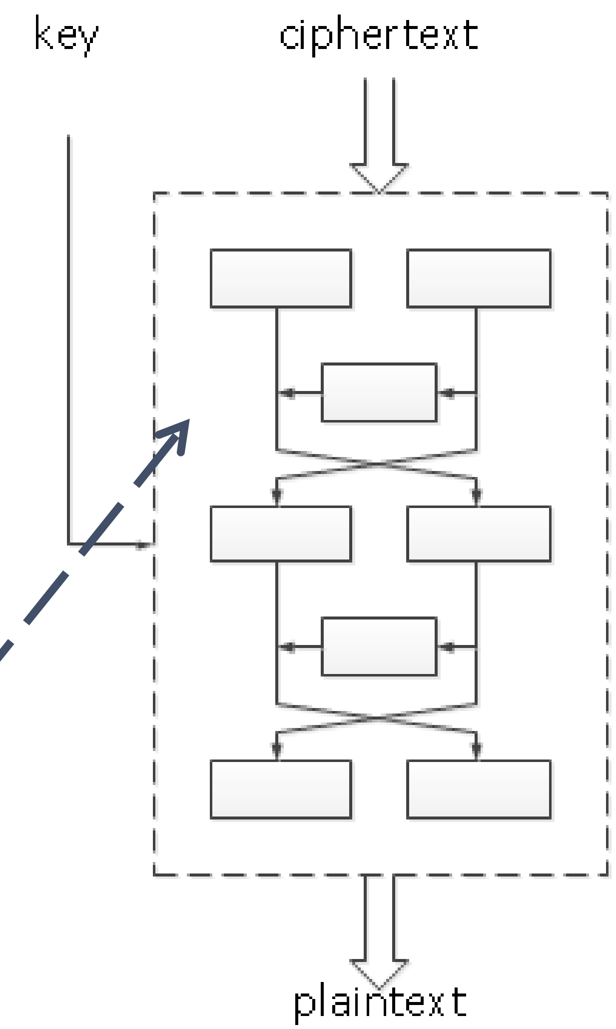
Potential attacks on WBC (III)

Side channel analysis (SCA) / intermediate data analysis



Observe data here

and compare it to expected data here



WBC attack literature

- Attacks for all academic WBC proposals
  - Focus on key extraction
  - **Type of transformations assumed known**
  - Concrete transformation and key unknown
- In real life…
  - *we do not know much about the design!*
- Not many publicly documented SCA/FI on WBC
  - Implementation-specific DFA paper in 2002 [2]
  - Recent generic DPA-like attack in [3]*

* Authors coined the term Differential Computational Analysis

Introduction

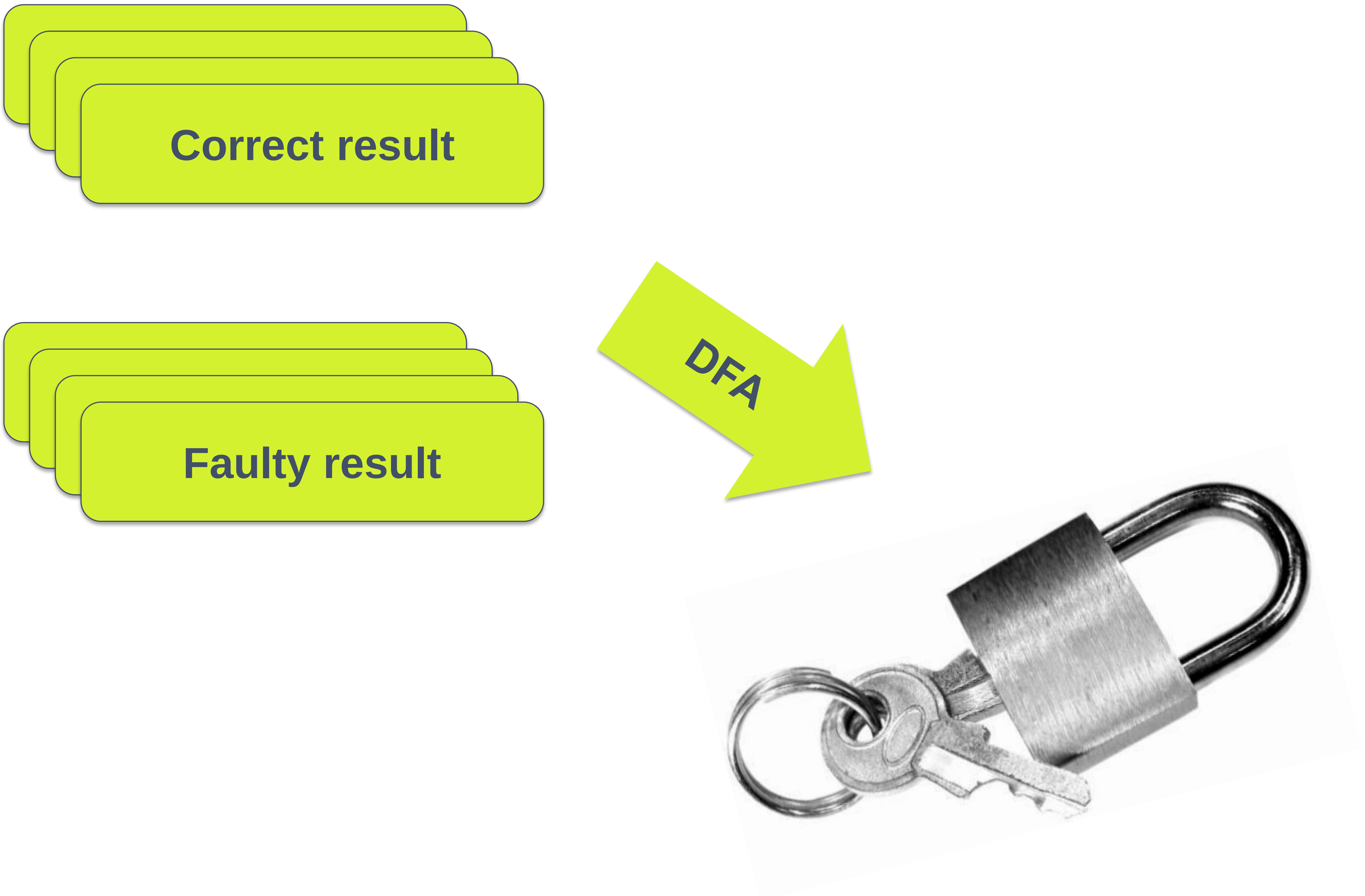
Key recovery attacks

Conclusion

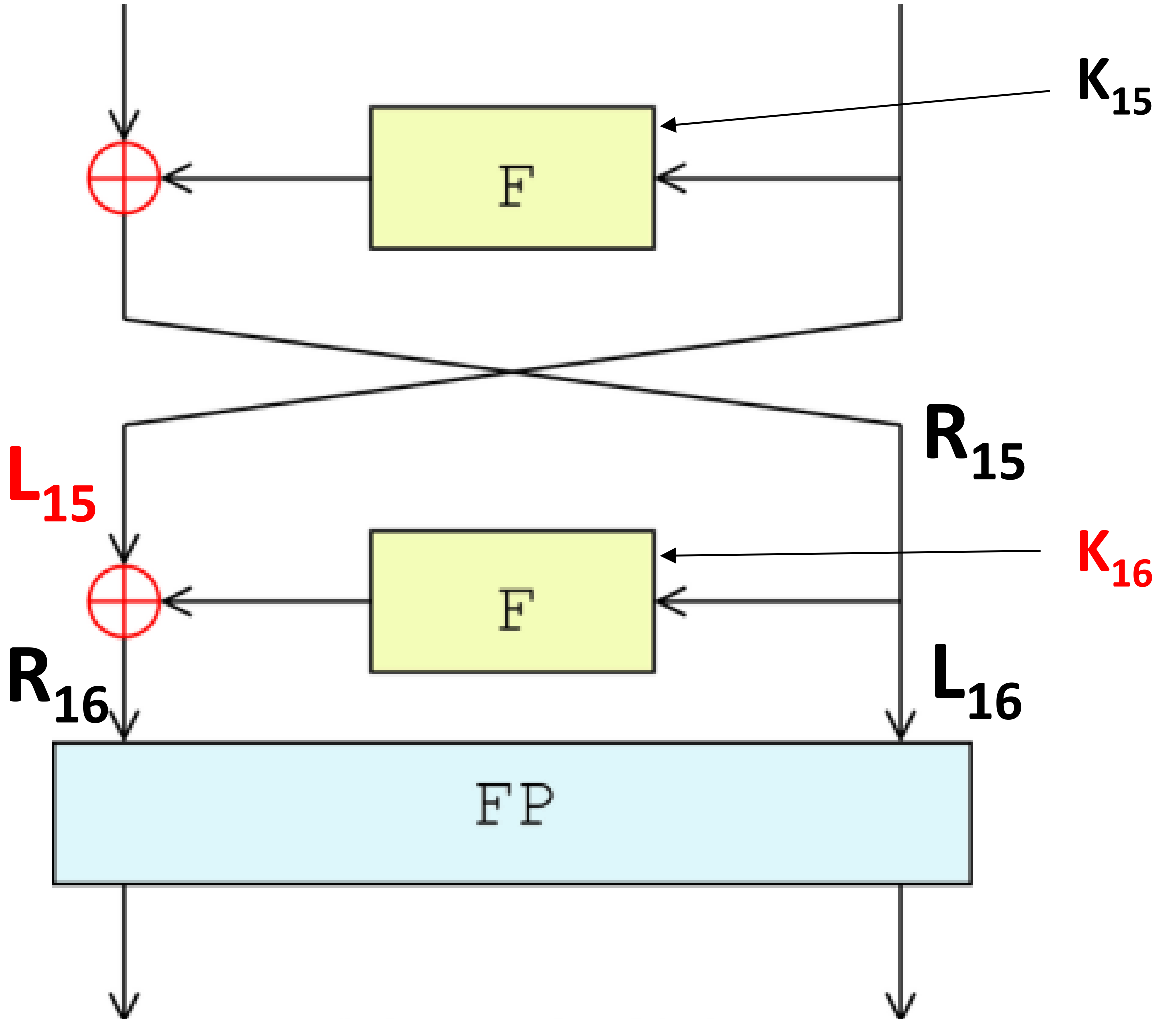
Fault Injection Attacks

…on WBC

Differential Fault Analysis

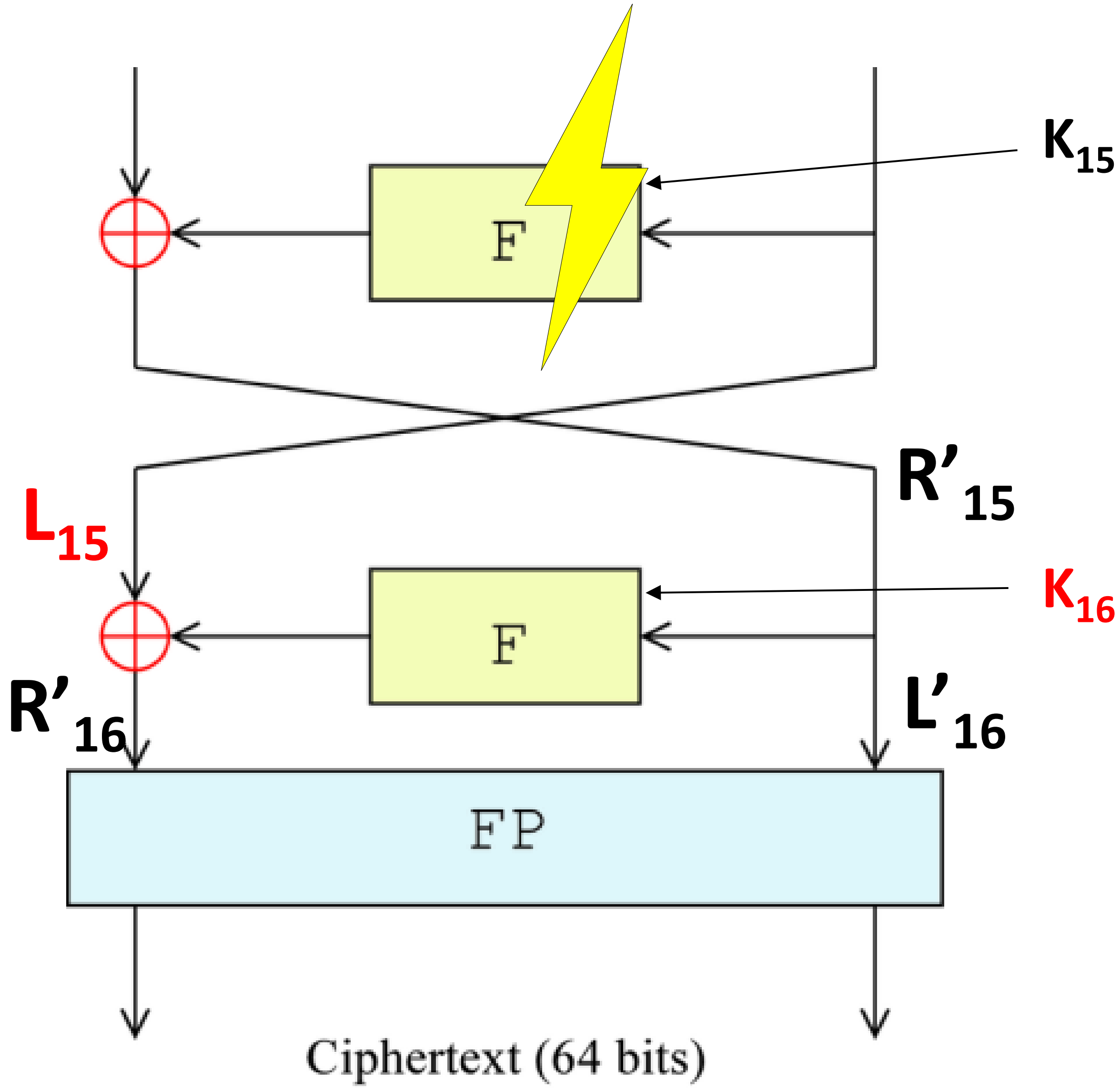


DFA computation for DES



$$R_{16} = F(R_{15}, K_{16}) \oplus L_{15}$$

DFA computation for DES



$$R_{16} = F(R_{15}, K_{16}) \oplus L_{15}$$

$$R'_{16} = F(R'_{15}, K_{16}) \oplus L_{15}$$

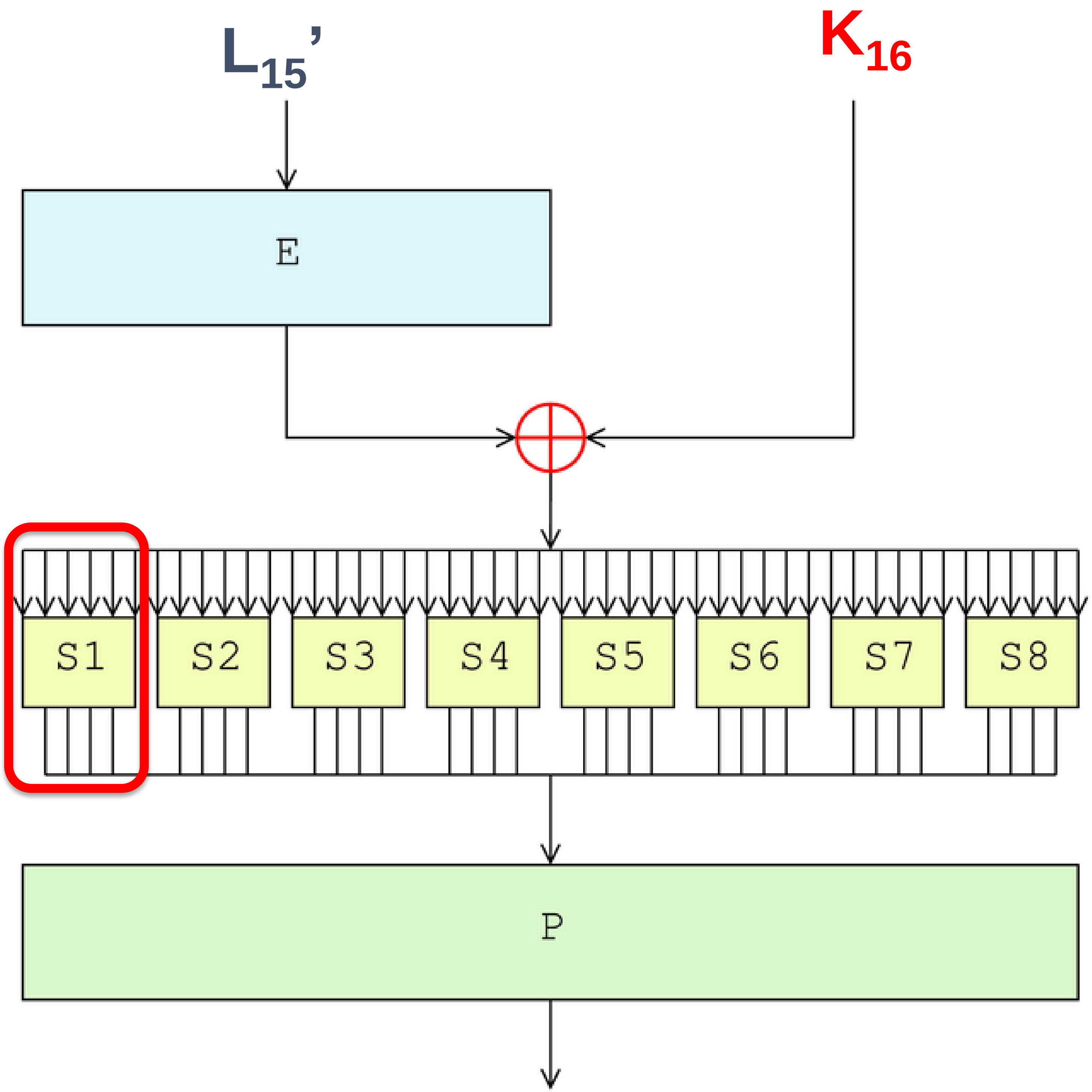
XOR

$$R_{16} \oplus R'_{16} = F(R_{15}, K_{16}) \oplus F(R'_{15}, K_{16})$$

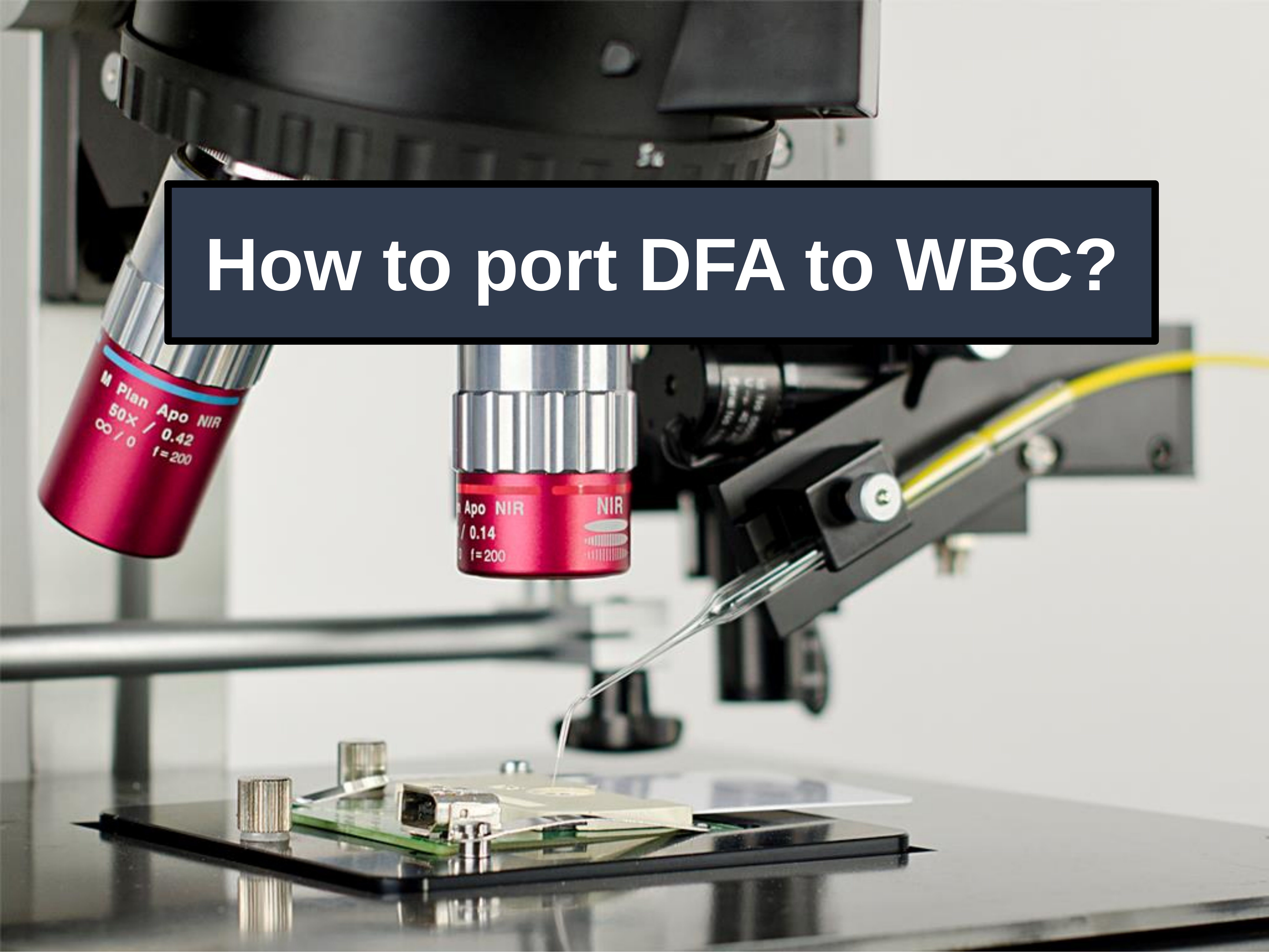
Divide and conquer

Independent
6-bit sub-keys

Independent
6-bit search



How to port DFA to WBC?



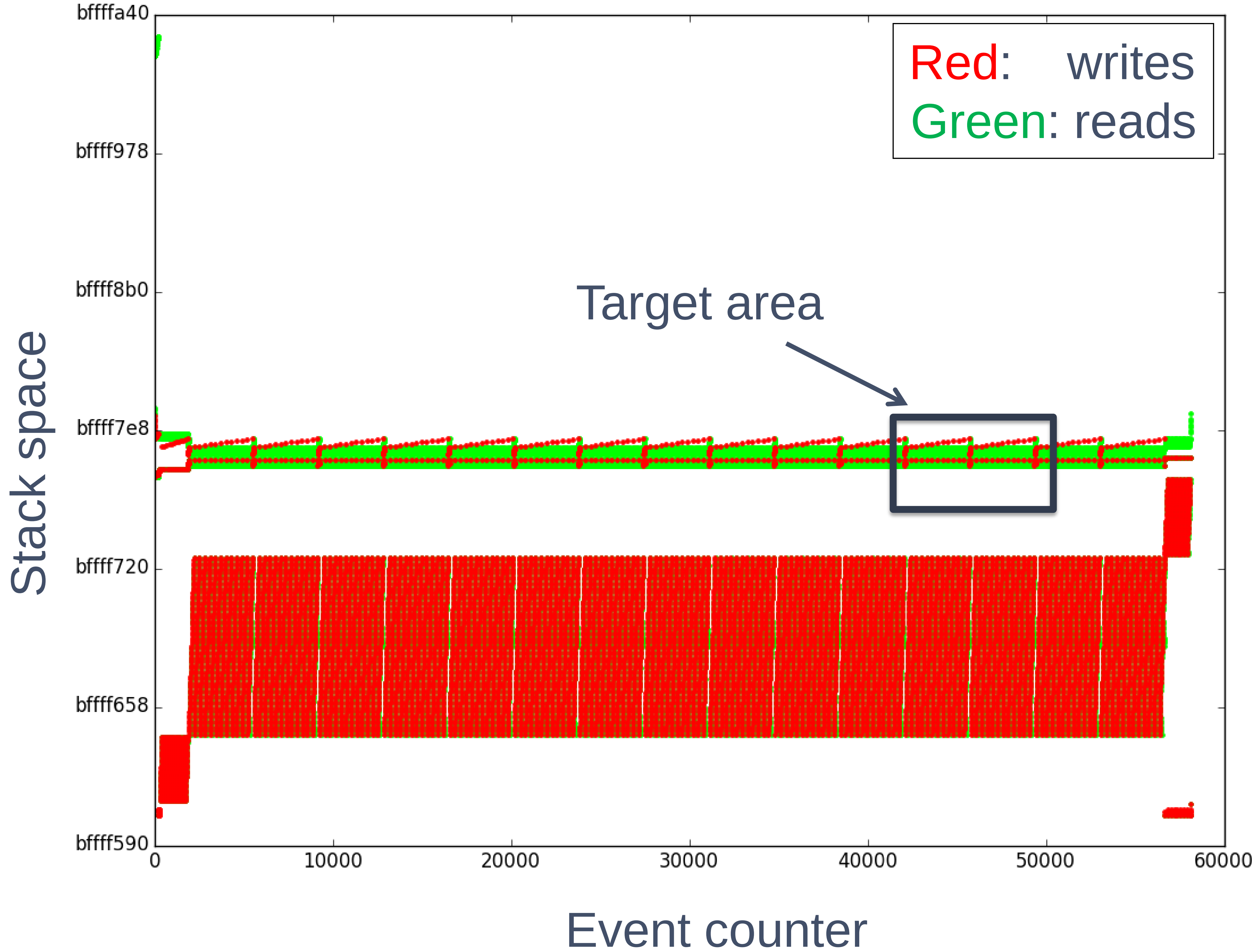
DFA attack process

1. **Location of fault injection point**
2. **Fault injection and ciphertext collection**
   - Multiple options available
3. **Fault analysis**
   - We use our own tools
   - Some AES DFA examples on GitHub

Example target: wbDES

- Binary DES encryption WBC
  - Challenge posted at whiteboxcrypto.com
- DES key hidden within lookups
  - Key value is 0x30 0x32 0x34 0x32 0x34 0x36 0x32 0x36
- We’ll demo all our attacks on this target

STEP 1: Locating the injection point



STEP 2: Fault injection

1. Select target event within target region
2. Modify data read by that event

```
def hook_mem_access_fault(uc, access, address, size, value, user_data):
    global output, evtId, fault
    evtId += 1
    pc = uc.reg_read(UC_X86_REG_EIP)

    targetId = user_data[0]
    if access == UC_MEM_READ:
        typ = "r"
        value = u32(uc.mem_read(address, size))
        if should_fault(evtId, targetId, fault, address, size):
            print "FAULTING AT ", targetId
            # Already faulted this time
            fault = False
            # Random bit in this event
            bitfault = 1 << random.randint(0, size*8 -1)
            uc.mem_write(address, pack(value ^ bitfault, size))
```

If event id within target region

Invert a random bit

STEP 3: Analysis

DEMO

Summary DFA results

Implementation	Fault injection	Results
Wyseur (DES)	Unicorn script	Broken in 40 faults
Hack.lu 2009 (AES)	Debugger script	Broken in 90 faults
SSTIC 2012 (DES)	Modified lifted code	Broken in 60 faults
Karroumi (AES)	Modified source code	Broken in 80 faults
NSC 2013 (encoded AES)	N/A	Not broken – encoding makes DFA not feasible

Side Channel Attacks

…on WBC

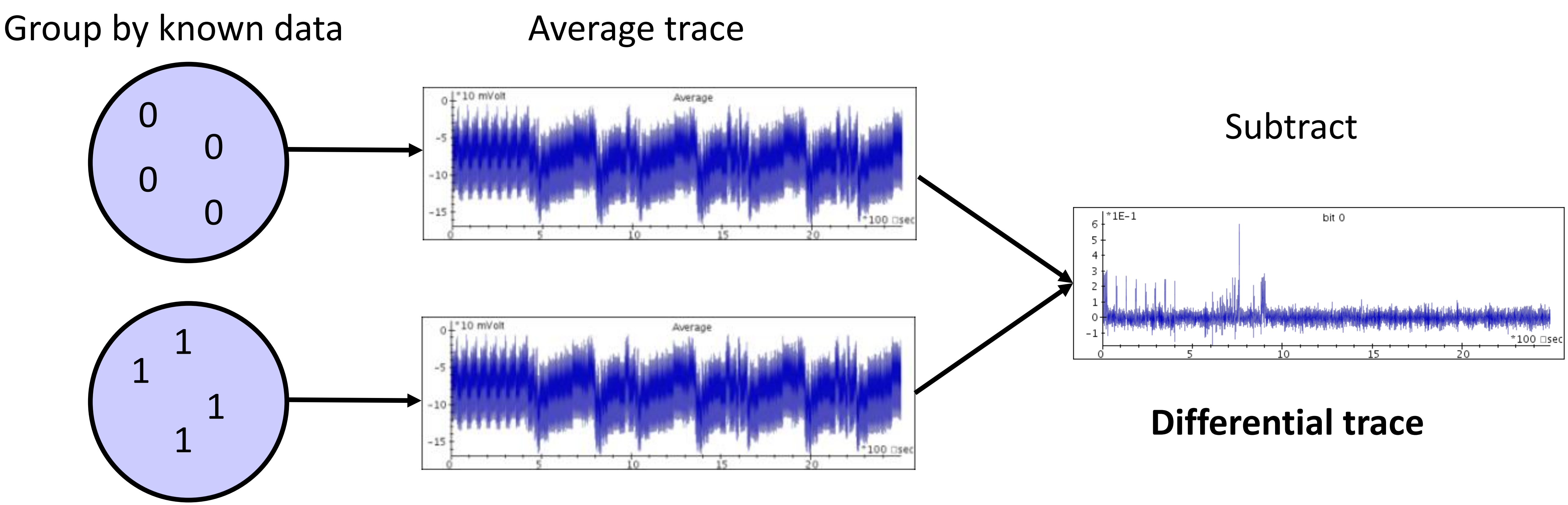
What is a DPA attack?

Differential **P**ower **A**nalysis attack

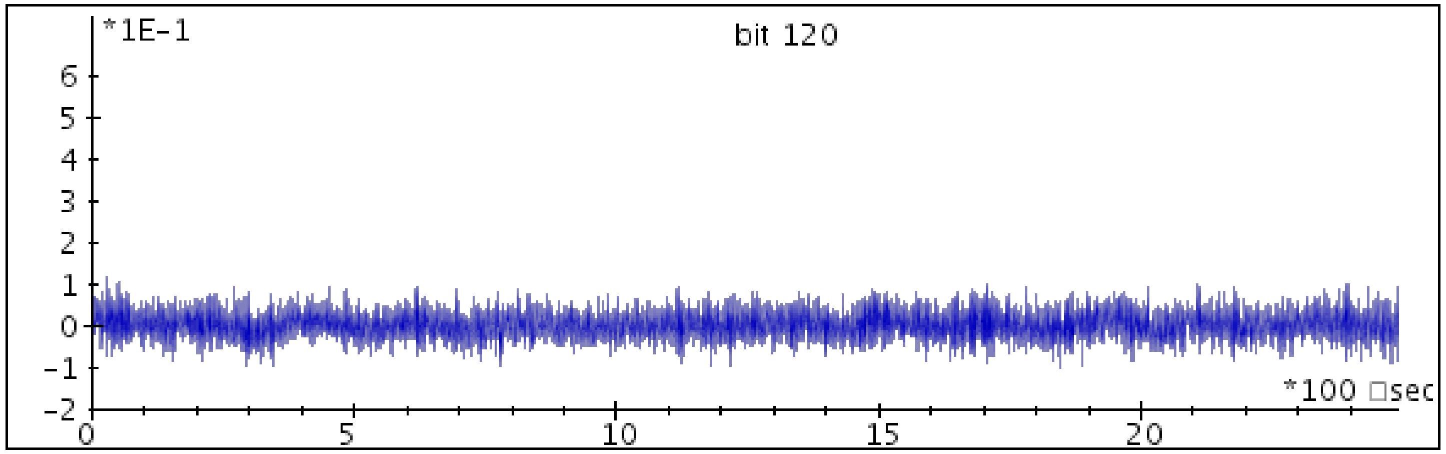
First proposed ~1998 by Paul Kocher to attack on smart cards:

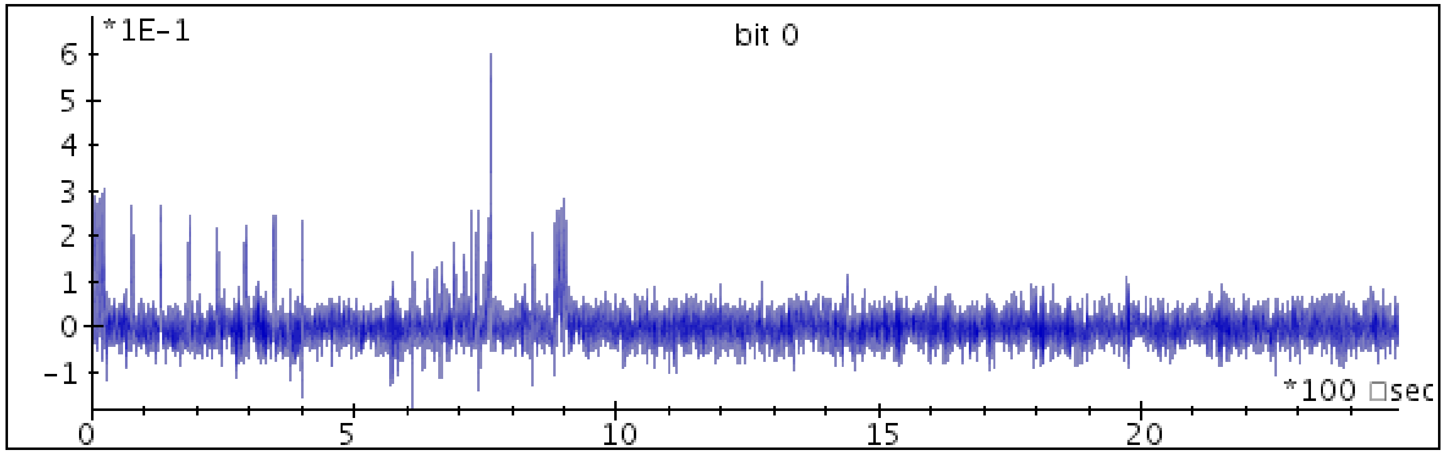
- ✓ Measuring power consumption of a crypto execution
- ✓ Take multiple measurements for different inputs
- ✓ Infer information about the key from the difference of these

Differential trace



Hypothesis testing

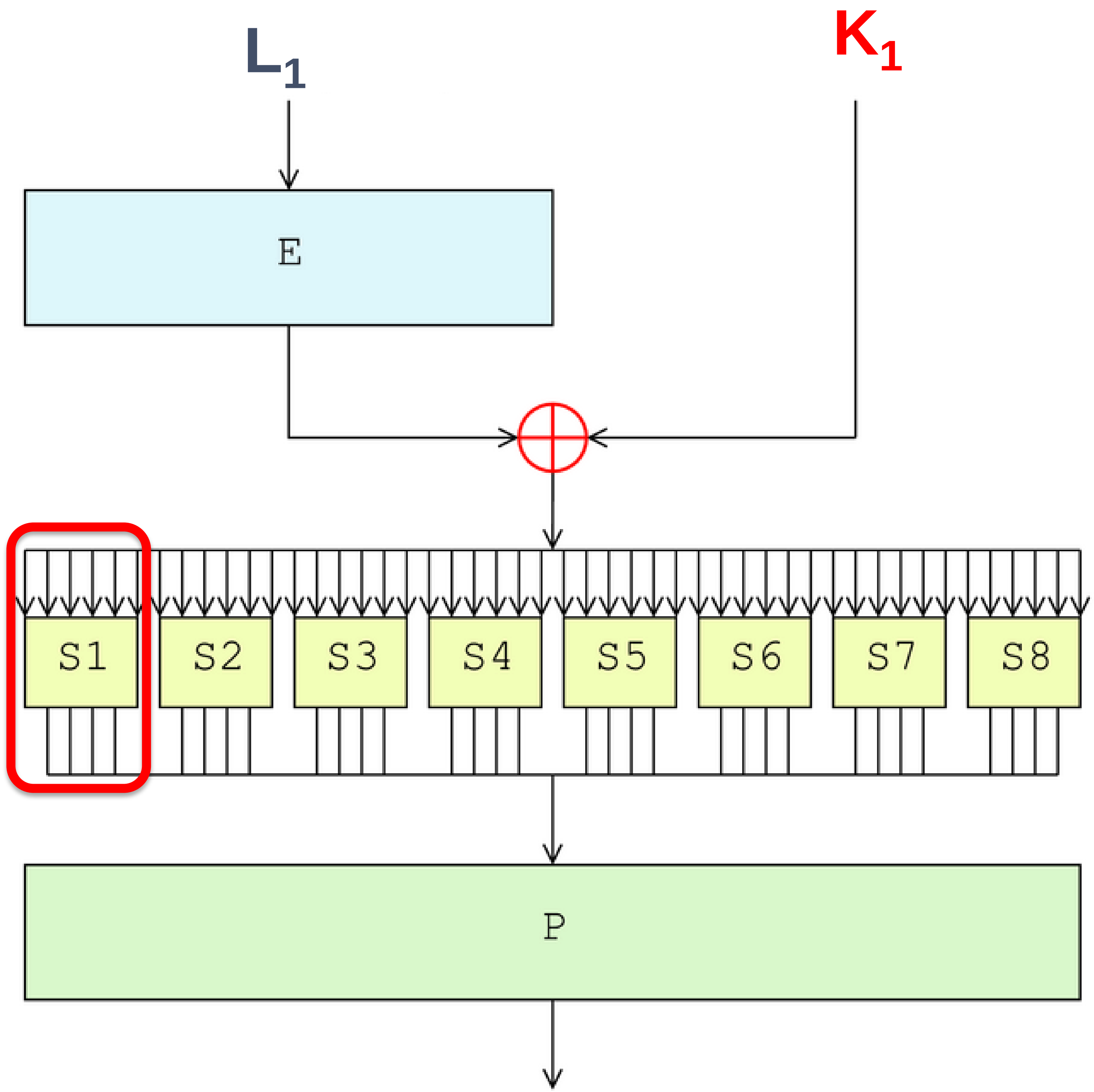




Divide and conquer

Independent
6-bit sub-keys

Independent
6-bit search



Generalization of differential SCA attacks

Take multiple "measurements" of behavior of crypto operations for different data

Predict behavior for sub keys based on the same data and "leakage" model

Apply statistical methods to distinguish the "best" sub key

- Difference of means
- Correlation
- Mutual Information analysis, Linear regression analysis, …

Find correct guesses for all sub keys to determine key

To our surprise….

It works on White Box Crypto out-of-the-box!!!

SCA attack process

1. Instrument WBC to collect “measurements”
   - Again:

2. Execute WBC with random inputs multiple times

3. Collect “measurement – input/output pairs” in useable form

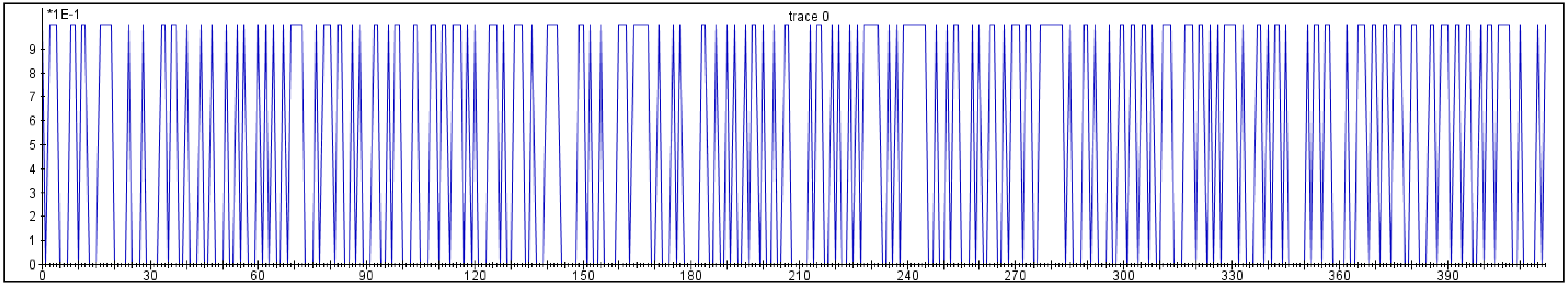
4. SCA Analysis

STEP1: Capture measurement

- Grab the data using any method that fits your target
  - Instrument execution (eg. PIN, Valgrind)
  - Capture stack snapshots per crypto round (Hooking, debugger)
  - Use emulators and record (QEMU, Unicorn, PANDA)
- Capture any information during execution that might leak
  - All reads/writes to memory
  - Lower bits of addresses of memory accesses
  - All register contents

STEP2+3: Execute + Collect

- Provide/inject random input data, capture output data
  - Program arguments
  - Use instrumentation from STEP 1
- Store it in a way that allows testing key guesses
  - Store as single bit samples
  - Assure alignment between multiple captures



STEP 4: SCA Analysis

Same target as for DFA: wbDES

Same hidden key: 0x30 0x32 0x34 0x32 0x34 0x36 0x32 0x36

DEMO

Summary SCA results

Implementation	Attacked intermediate	Results	Results NXP [3]
Wyseur (DES)	Round output	Broken in 75 traces	Broken in 65 traces
Hack.lu 2009 (AES)	S-Box output	Broken in 16 traces	Broken in 16 traces
SSTIC 2012 (DES)	Round output	Broken in 16 traces	Broken in 16 traces
Karroumi (AES)	S-Box and GF(256) inverse	Broken in ~2000 traces	Broken in ~500 traces
NSC 2013 (encoded AES)	N/A	Not broken	Not broken – encoding makes DPA not feasible

What does it mean?

No detailed knowledge required

- Of WBC implementation
- Where the WBC is processed exactly

No manipulation required

➢ A secret random input/output encoding is the *only* barrier

But:

These random encodings do not work for many real world applications

Introduction

Key recovery attacks

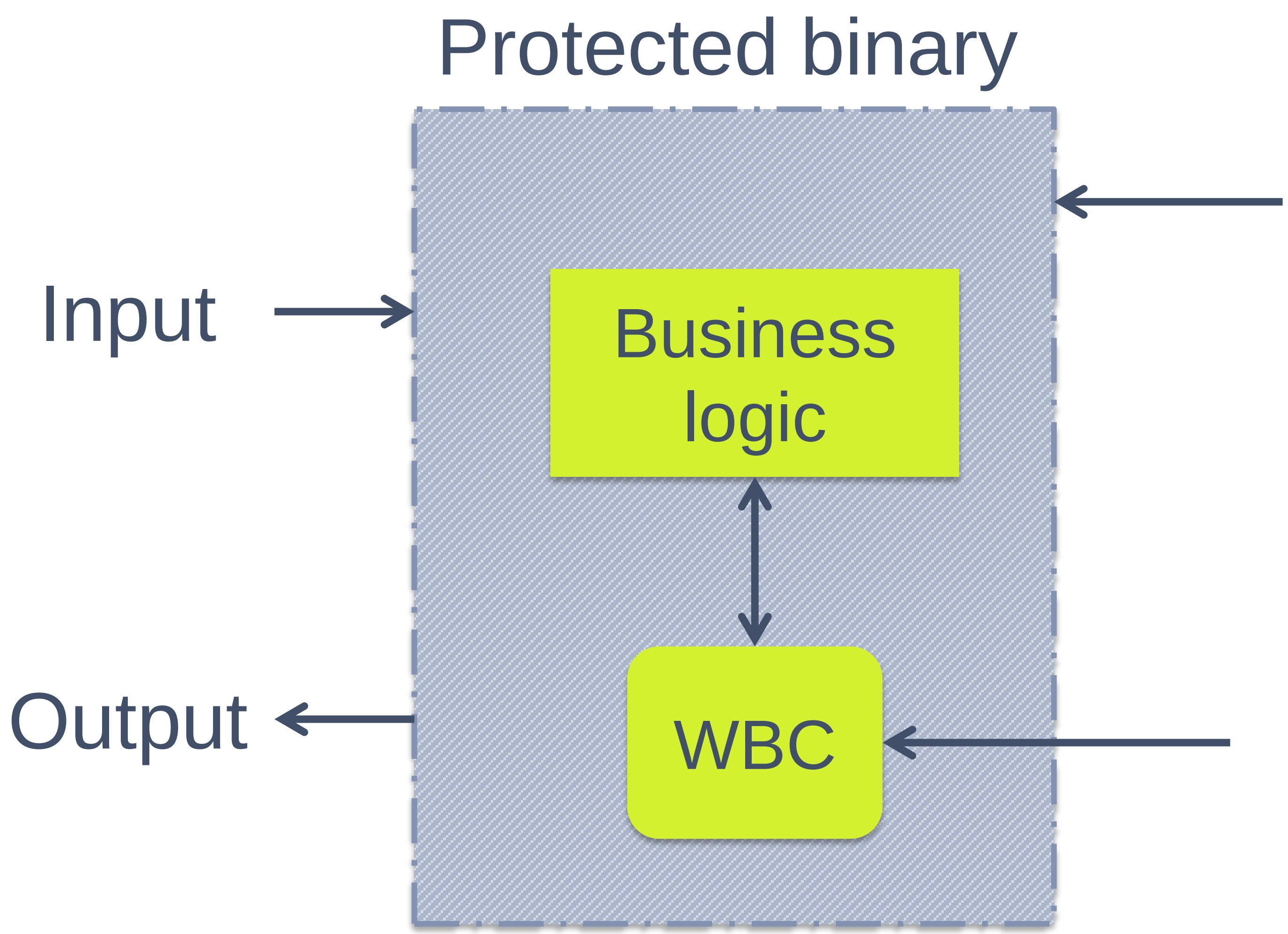
Conclusion

Is White-Box Crypto dead?

Is WBC broken and useless?

- SCA/FI on standard WBC very effective:
  - Very limited knowledge required
  - RE skills might be needed
  - Countermeasures and risk mitigation required
- Broken several open-source and commercial WBC
  - Commercial implementations typically require more RE skills
- But…
  - Not regular software crypto → more complex attacks
  - Software protection layers can be a deterrent
  - With renewability it can be good enough

How to make it stronger?



Robustness against advanced SW RE

Robustness against key extraction attacks (SCA, FI, algebraic, …)

But how?

Side Channel Analysis attacks

- Must prevent statistical dependence between intermediates and key
- Typical countermeasures based on randomness difficult in white-box scenario

Differential Fault Analysis attacks

- Double-checks on encoded data → might be bypassed if detected!
- Carry redundant data along computation?
- Break fault models by propagating faults?

Do you have any other ideas?

riscure

Challenge your security

Thank you!!

eloi@riscure.com
@esanfelix

mune@riscure.com
@pulsoid

dehaas@riscure.com

References

[1] http://crypto.stanford.edu/DRM2002/whitebox.pdf

[2] http://crypto.stanford.edu/DRM2002/drm1.pdf

[3] https://eprint.iacr.org/2015/753

[4] https://www.cosic.esat.kuleuven.be/publications/thesis-152.pdf

[5] https://www.cosic.esat.kuleuven.be/publications/thesis-235.pdf