riscure

Unboxing the whitebox

Jasper van Woudenberg @jzvw
CTO Riscure North America

ICMC’16

Riscure

Certification

- Pay TV, EMVco, smart meter, CC

Evaluation & consultancy

- Mobile (TEE/HCE/**WBC**)
- Secure architecture / design

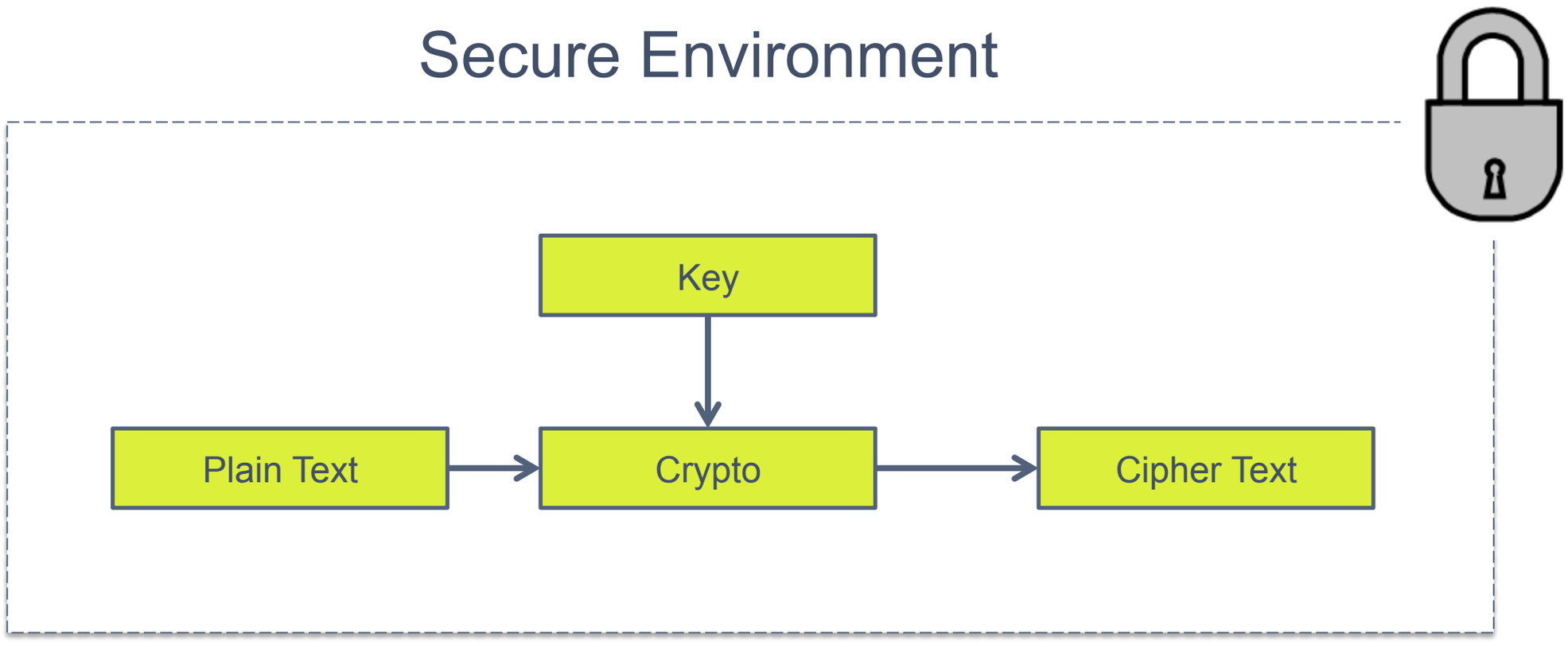
Tools

- Side channel analysis, fault injection

Training

- **Whitebox crypto**, SCA, FI

Protecting the keys…



Internal state needs to be protected

What and why…

- White-Box cryptography -> secure software crypto in an untrusted environment
- Being used today in Pay TV DRMs, mobile payments, …
- This talk: applying hardware attack concepts to software environments

Introduction

Key recovery attacks

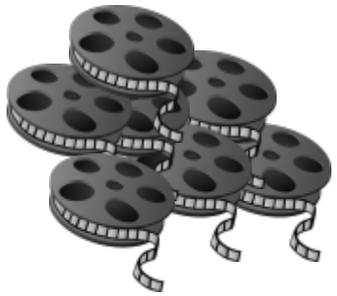
Conclusion

Sign of the times…

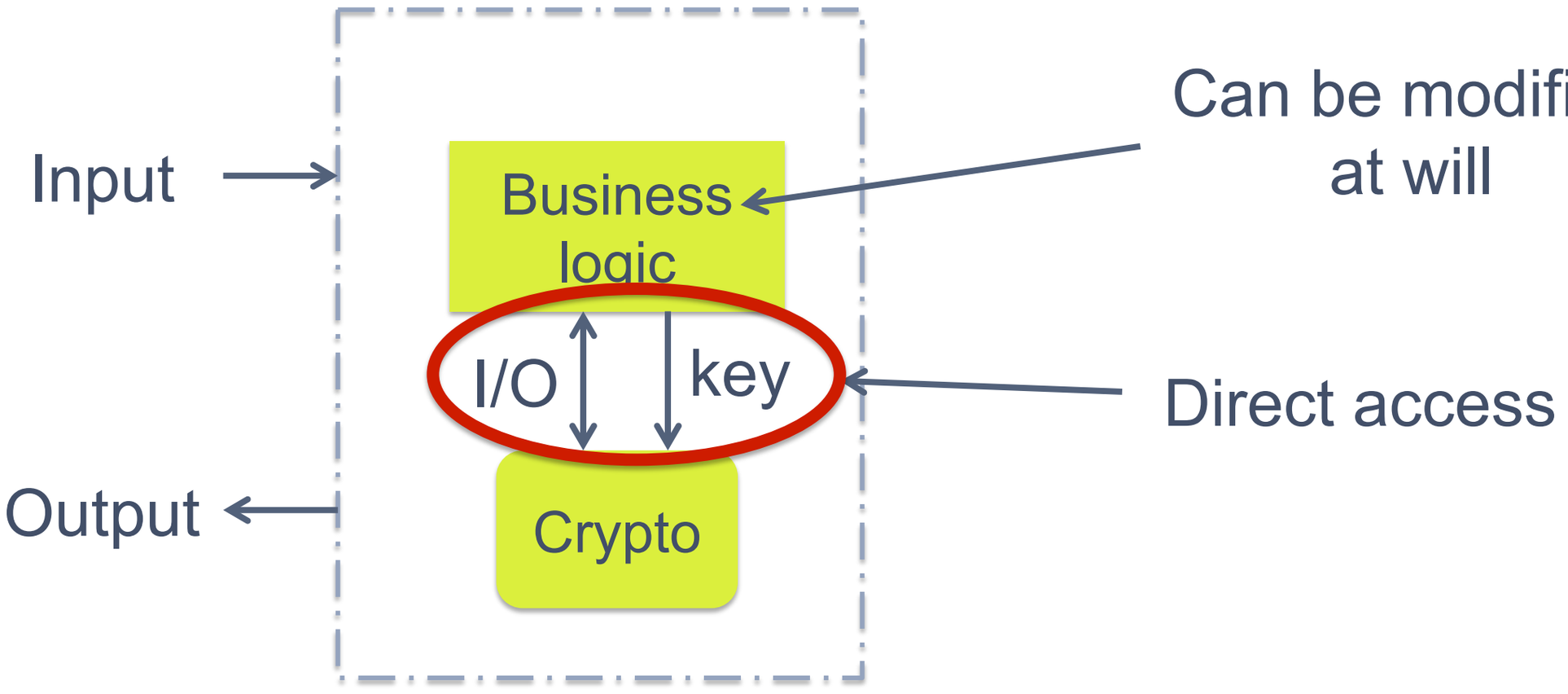
riscure

Sign of the times…

riscure



Software in the White-Box context



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.
- “Pure WBC” weakness: cloning/lifting

How does WBC work?

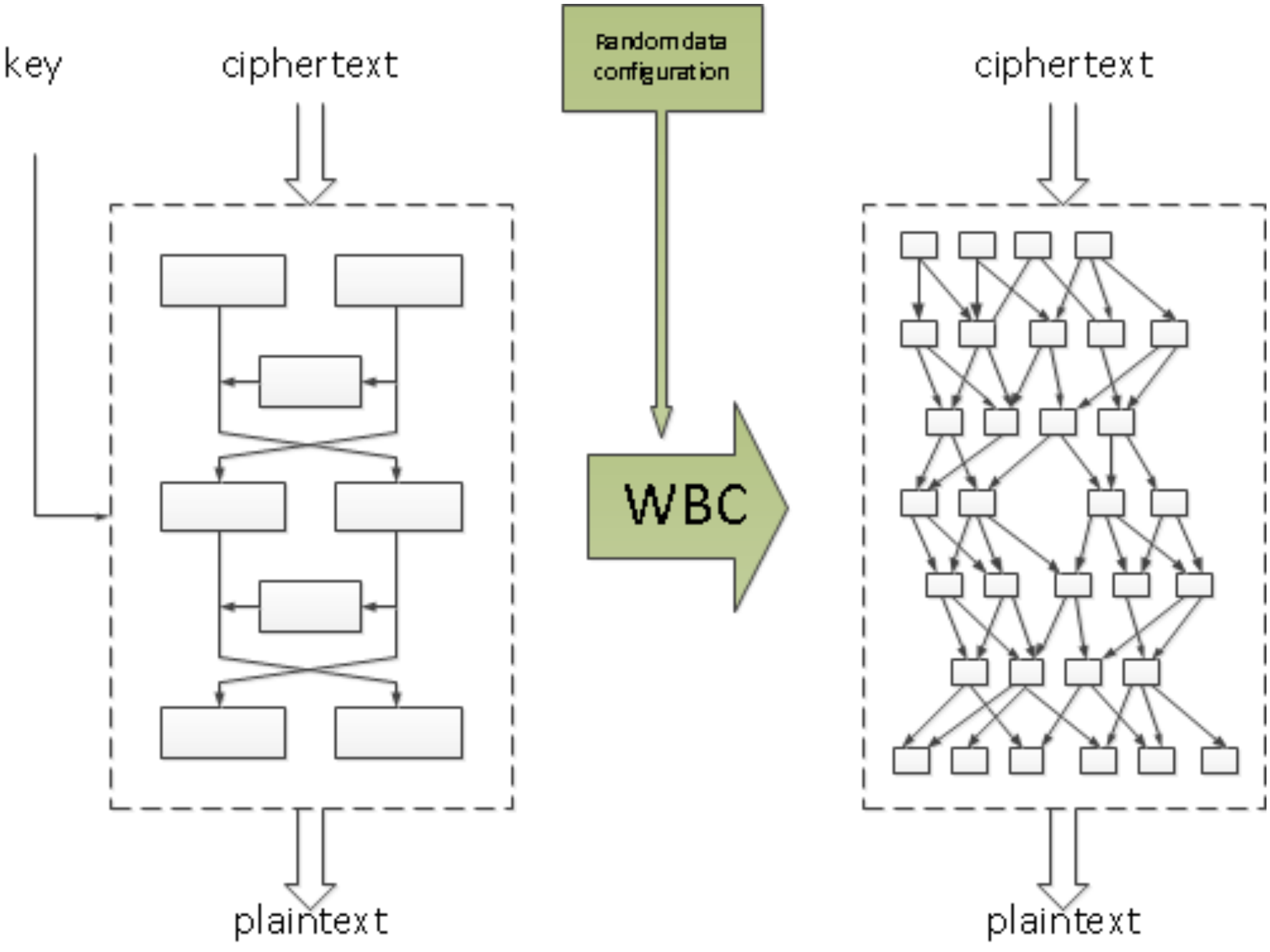


Image source: whiteboxcrypto.com

WBC Construction: partial evaluation

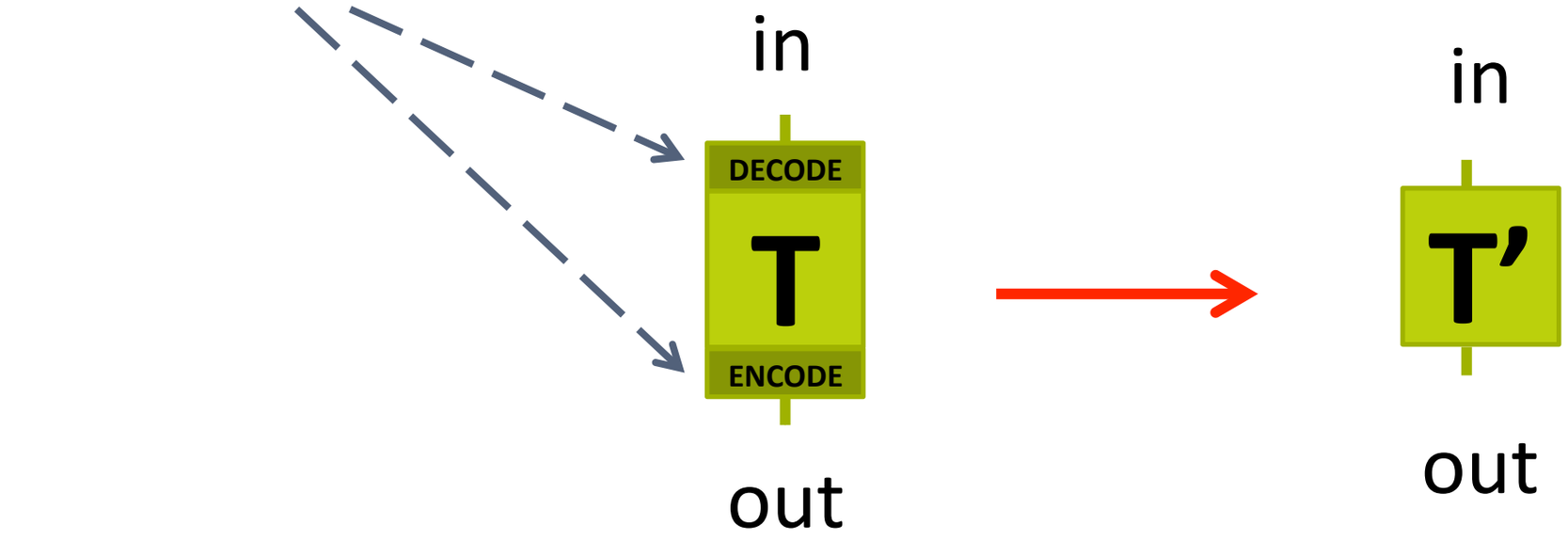
```
def __init__(self, lut, key):

    self._lut = []

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

WBC Construction: encoding

Internal encoding

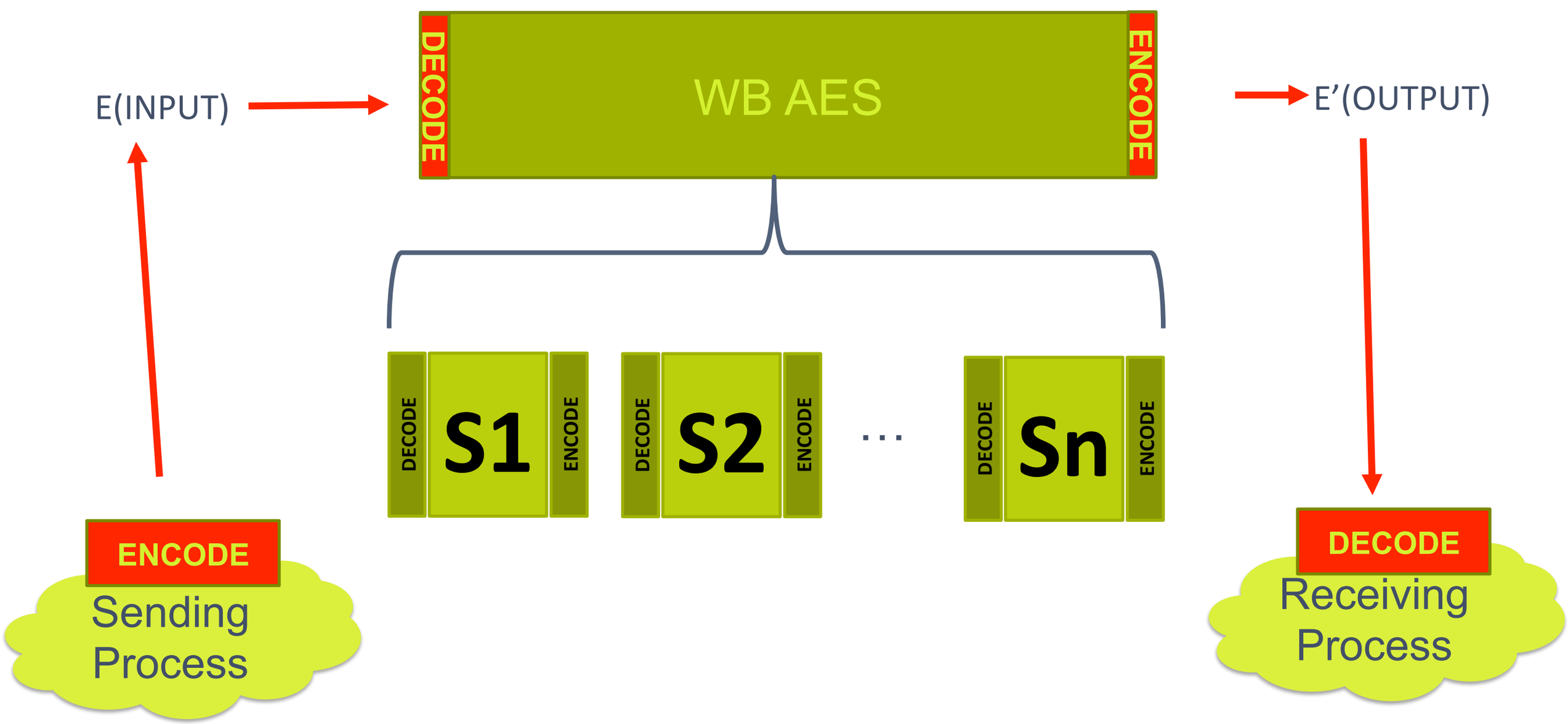



```
def __init__(self, lut, key, inbij, outbij):

    self._lut = []

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

External encoding



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]*

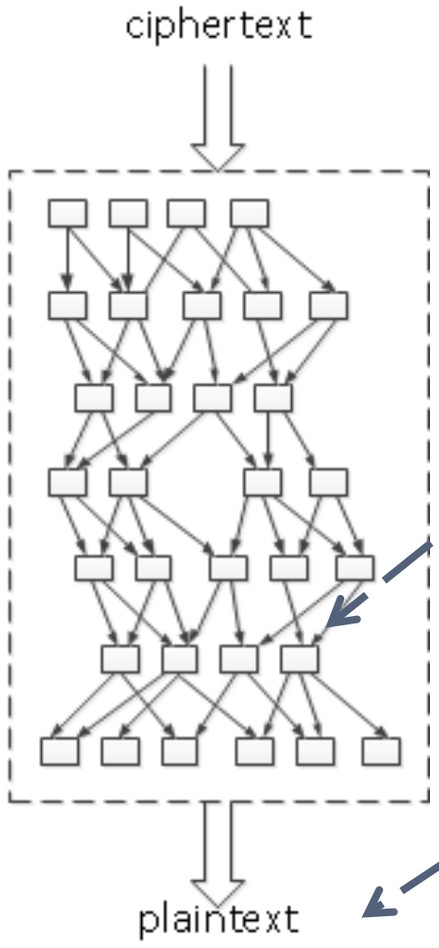
* Authors coined the term Differential Computational Analysis

Potential attacks on WBC (I)

Side channel analysis (SCA) / intermediate data analysis

Potential attacks on WBC (II)

Data manipulation – Fault Injection (FI)

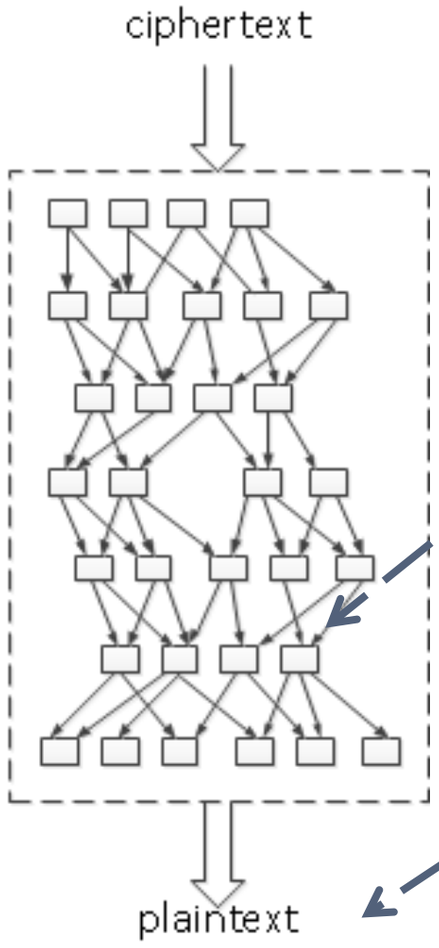


Modify data here

and observe changes to the output

Potential attacks on WBC (III)

Process manipulation – Fault injection (FI)



Modify code flow here

and observe changes to the output

Introduction

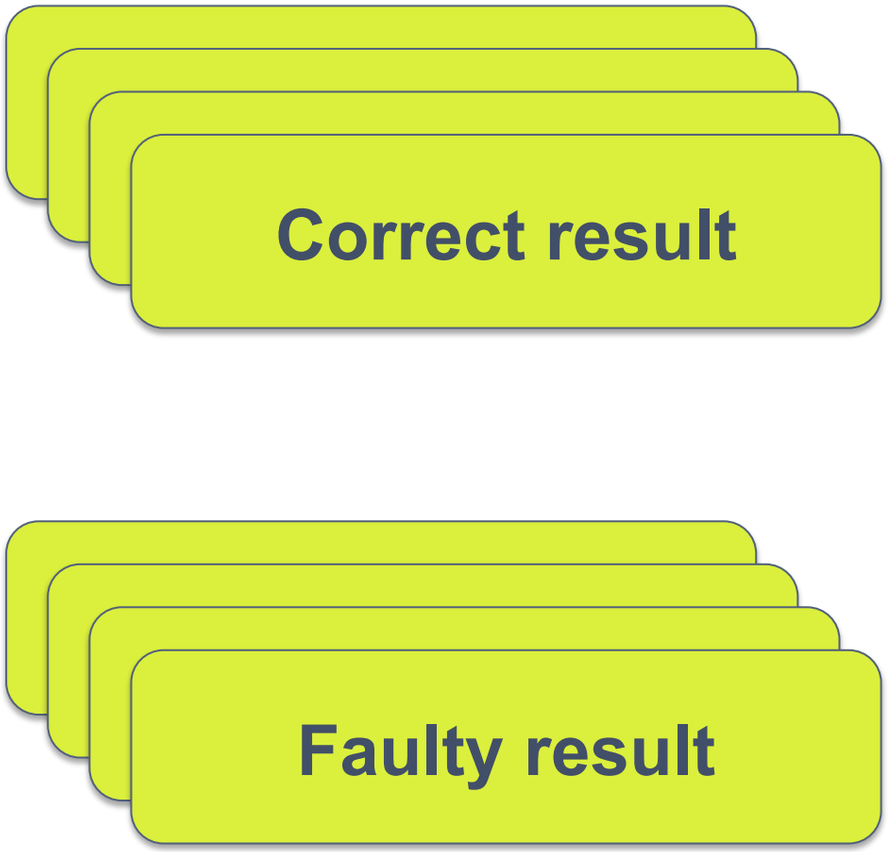
Key recovery attacks

Conclusion

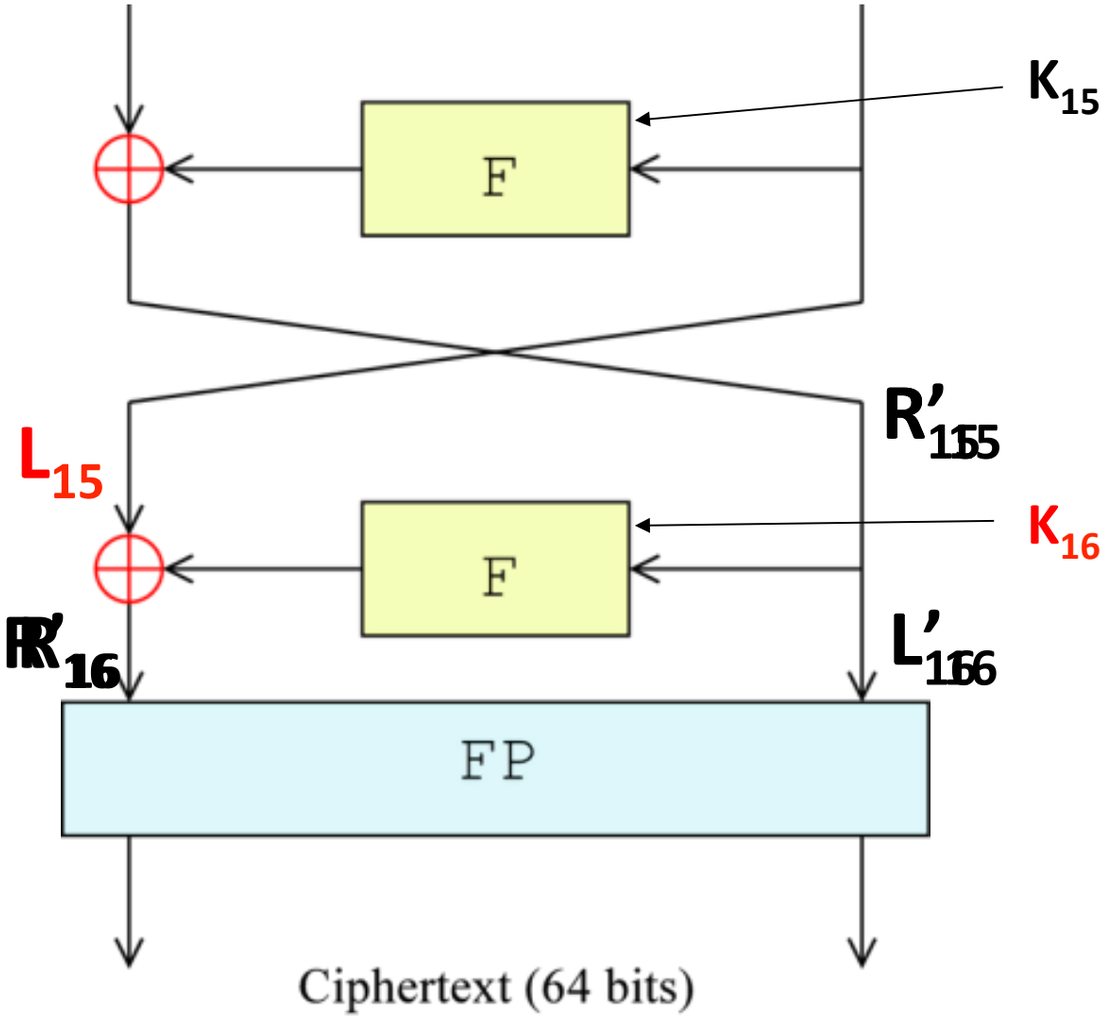
Fault Injection Attacks
…on WBC

Differential Fault Analysis

riscure



DFA computation for DES



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

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

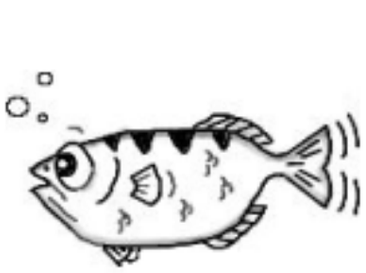
XOR

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

How to port DFA to WBC?

DFA attack process

1. **Location of fault injection point**
2. **Fault injection and ciphertext collection**
   - Multiple options available



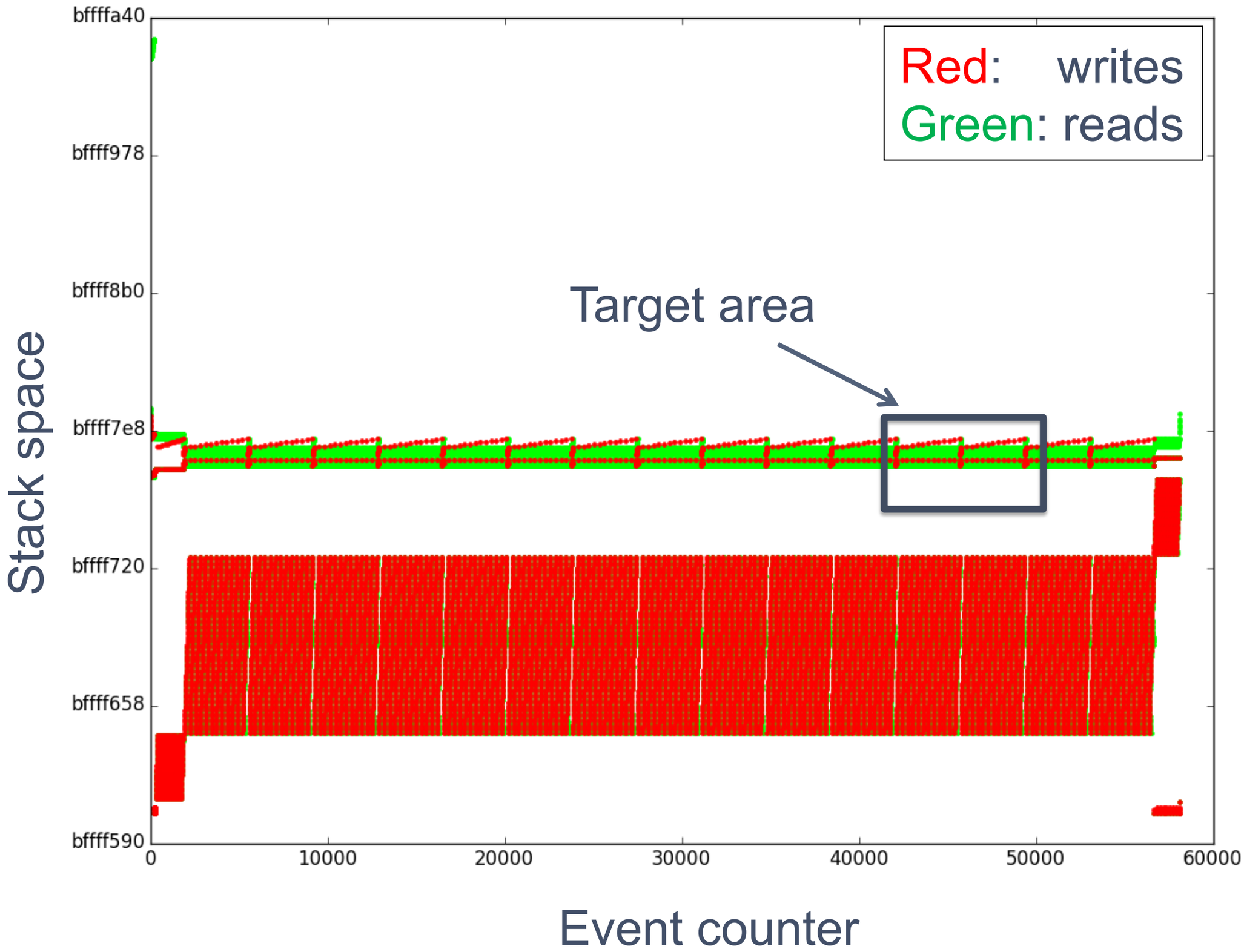
GDB
The GNU Project Debugger

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
3. Collect input/output pairs

```
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

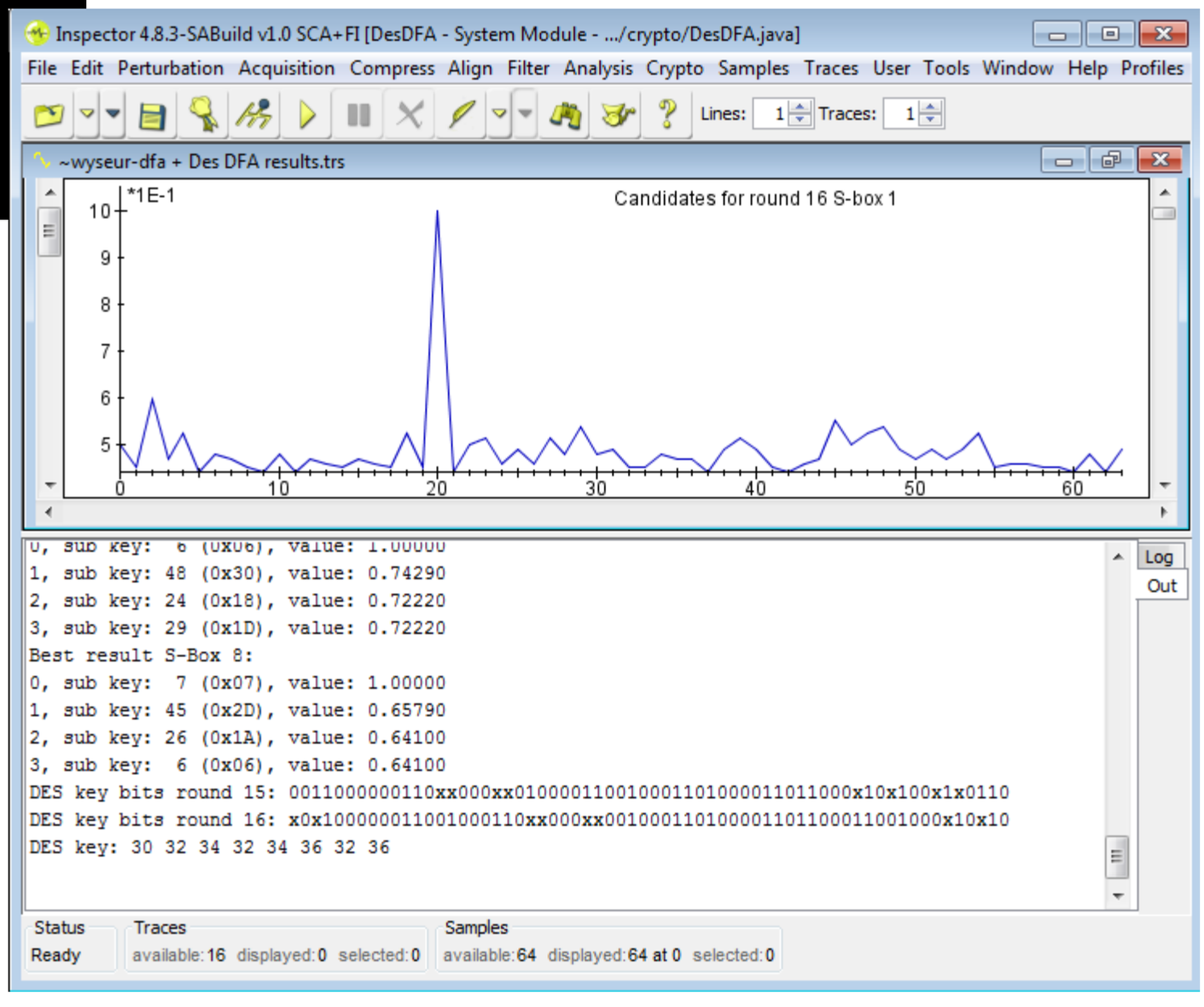
riscure

DEMO

Demo screenshots

riscure

```
FAULTING AT  44286
FINISHED  94
FAULTING AT  55284
FINISHED  95
FAULTING AT  44137
FINISHED  96
FAULTING AT  54337
FINISHED  97
FAULTING AT  52593
FINISHED  98
FAULTING AT  47387
FINISHED  99
user@debian:/media/sf_Shared/final/wbDES DFA$
```



Summary DFA results

riscure

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

riscure

Side Channel Attacks

…on WBC

What is a DPA attack?

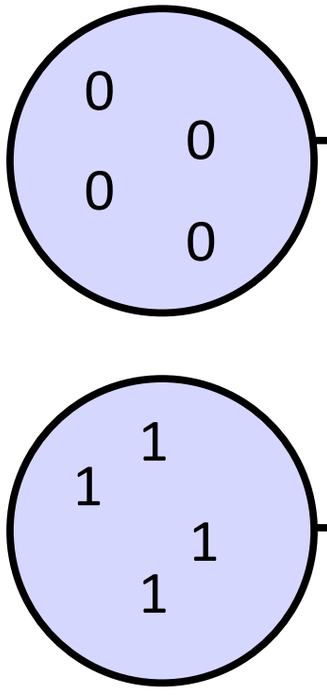
Differential **P**ower **A**nalysis attack

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

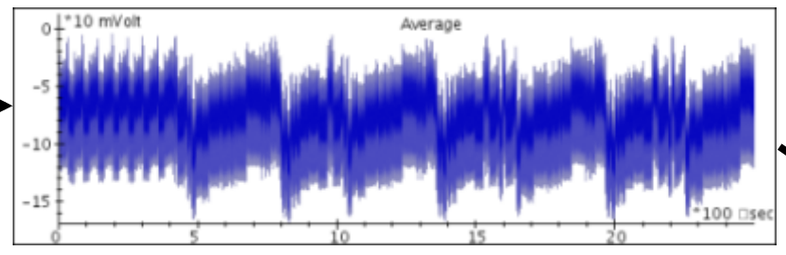
- ✓ Measuring power consumption of a crypto execution
- ✓ Take multiple measurements for different inputs
- ✓ Infer information about the key from the difference of these

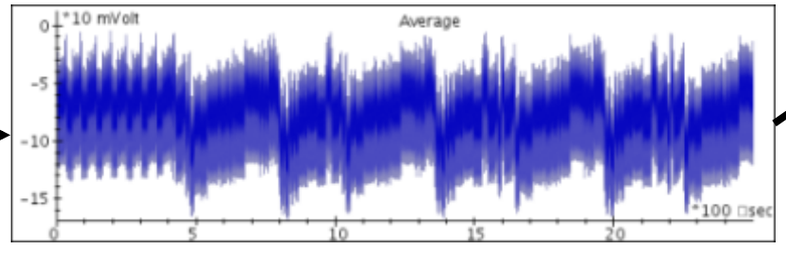
Differential trace

Group by known data

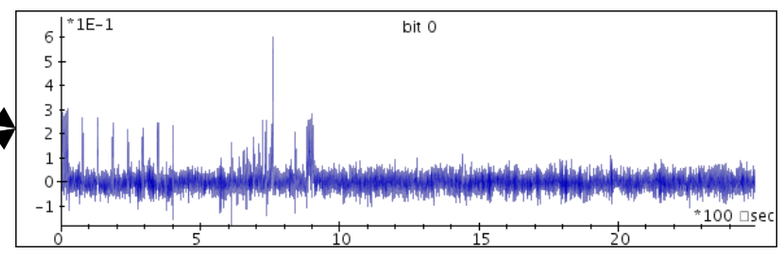


Average trace



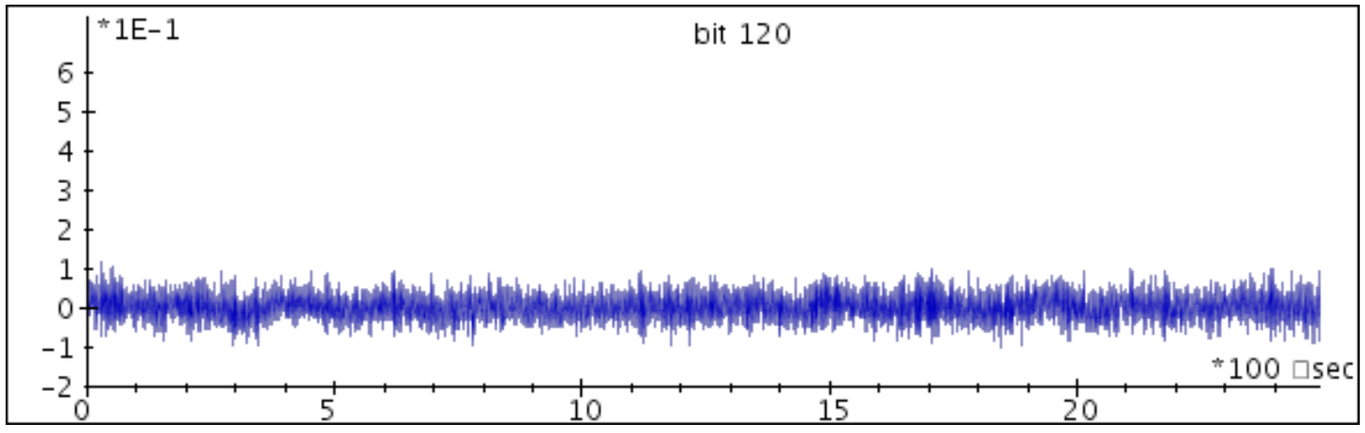


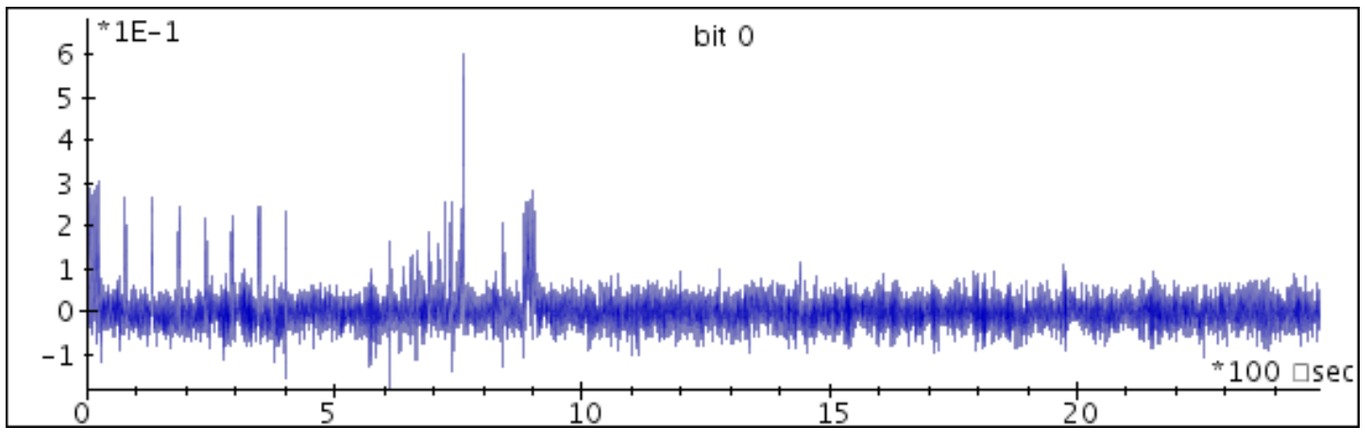
Subtract



Differential trace

Hypothesis testing





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….

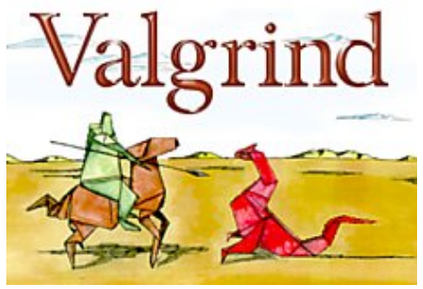
It works on W -the-box!!!

SCA attack process

riscure

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

GDB
The GNU Project Debugger



2. Execute WBC with random inputs multiple times
3. Collect “measurement – input/output pairs” in useable form
4. SCA Analysis

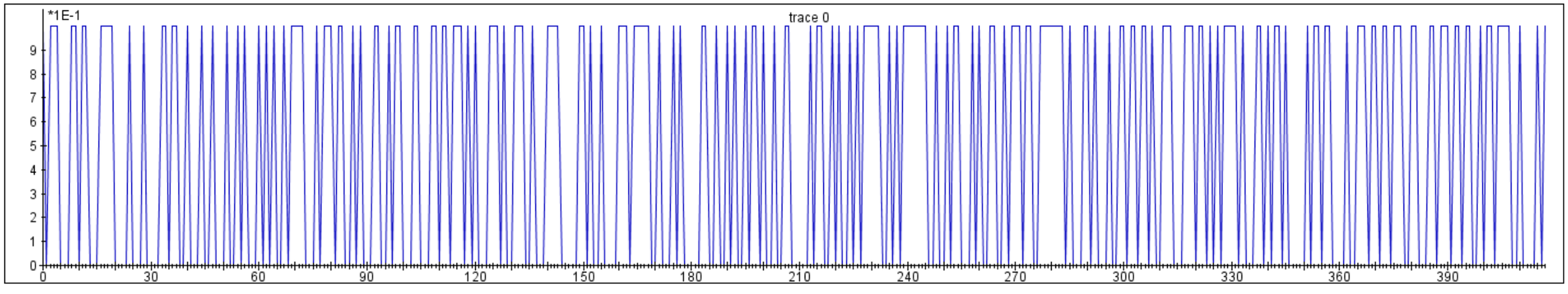
STEP1: Capture measurement

riscure

- 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

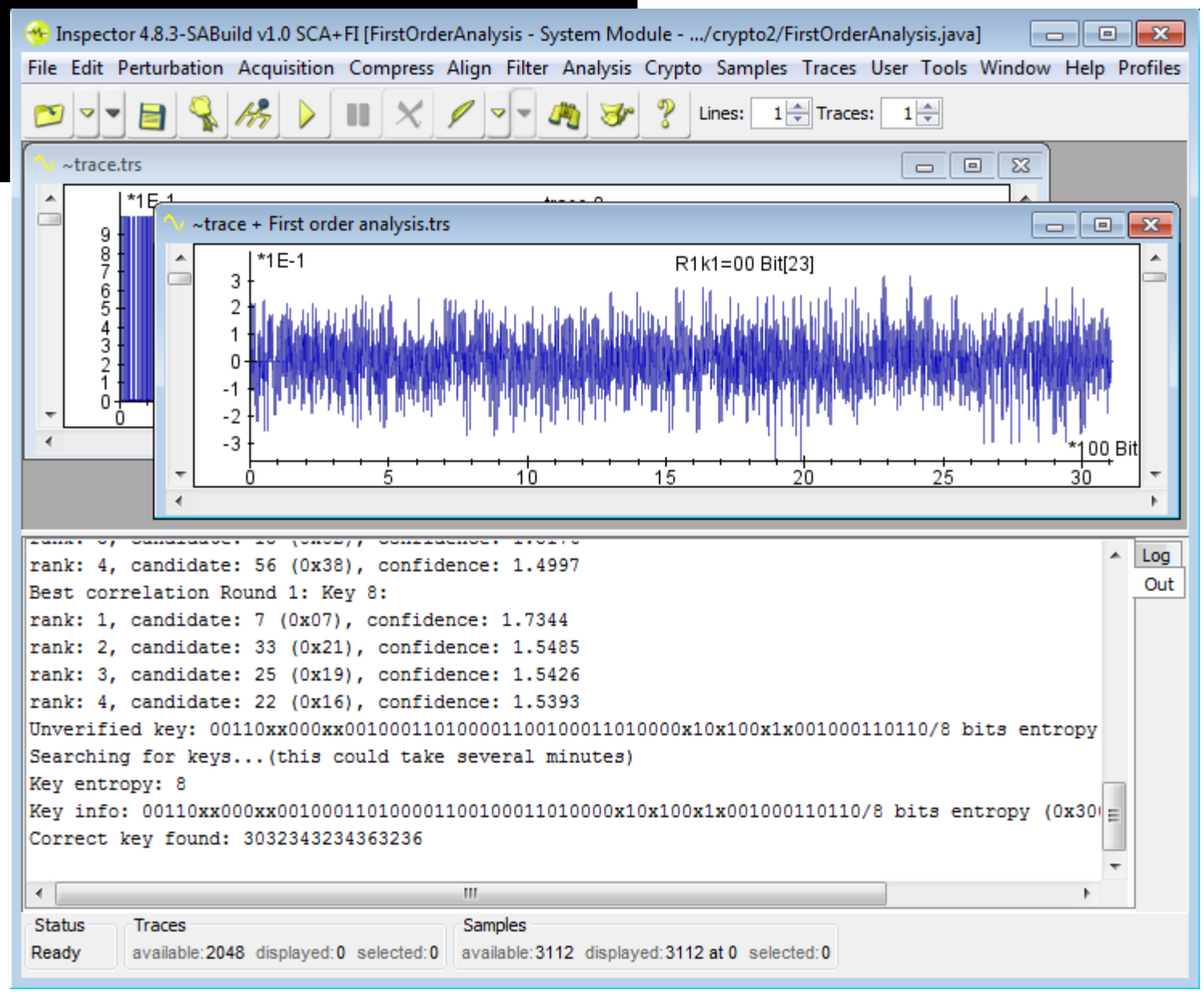
Same target as for DFA: wbDES

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

DEMO

Demo screenshots

```
user@debian:~/code/pin-2.14-71313-gcc.4.4.7-linux/source/tools/wb-tracer$ ./run_wbDES.py
00 B5CC1878C8C9AB32 -> 696F25C1432AB183
01 5C6EDCDC5E52862A -> D95CCBA31FE7FCB3
02 C029FC404069C227 -> 096A26FE3DFB57FD
03 37FA41234FA217AA -> CBD281D4F4D1B03A
04 A7C49F85784E6765 -> 3B690034D64C6569
05 A32B01B894D4FCEF -> DD4B776144649DFF
06 FDF184A9A566B81E -> E6A6BA2E37B2FD79
07 43C399F7C380B24E -> 8A55809833EE4996
08 1710F845C70C934B -> A0F48B4C00FABCBA
09 45C296736876E72B -> 22AB259AA4CF7086
10 D2DF591286CF524A -> E8BB628B34CC04AE
11 7B070875BA2741A9 -> 743F7C00A3BDB5AD
12 239E3A1908D9B4C6 -> 552FFC99C257B24E
13 5E9A9D7506B948E6 -> 68EACF2CA0D157DA
14 7CBFD8330944EB75 -> 0E297F0DB62B9CE3
```



Summary SCA results

riscure

Implementation	Attacked intermediate	Results
Wyseur (DES)	Round output	Broken in 75 traces
Hack.lu 2009 (AES)	S-Box output	Broken in 16 traces
SSTIC 2012 (DES)	Round output	Broken in 16 traces
Karroumi (AES)	S-Box and GF(256) inverse	Broken in ~2000 traces
NSC 2013 (encoded AES)	N/A	Not broken

Introduction

Key recovery attacks

Conclusion

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

After instrumentation, push-button attack

Mitigating WBC leakage

riscure

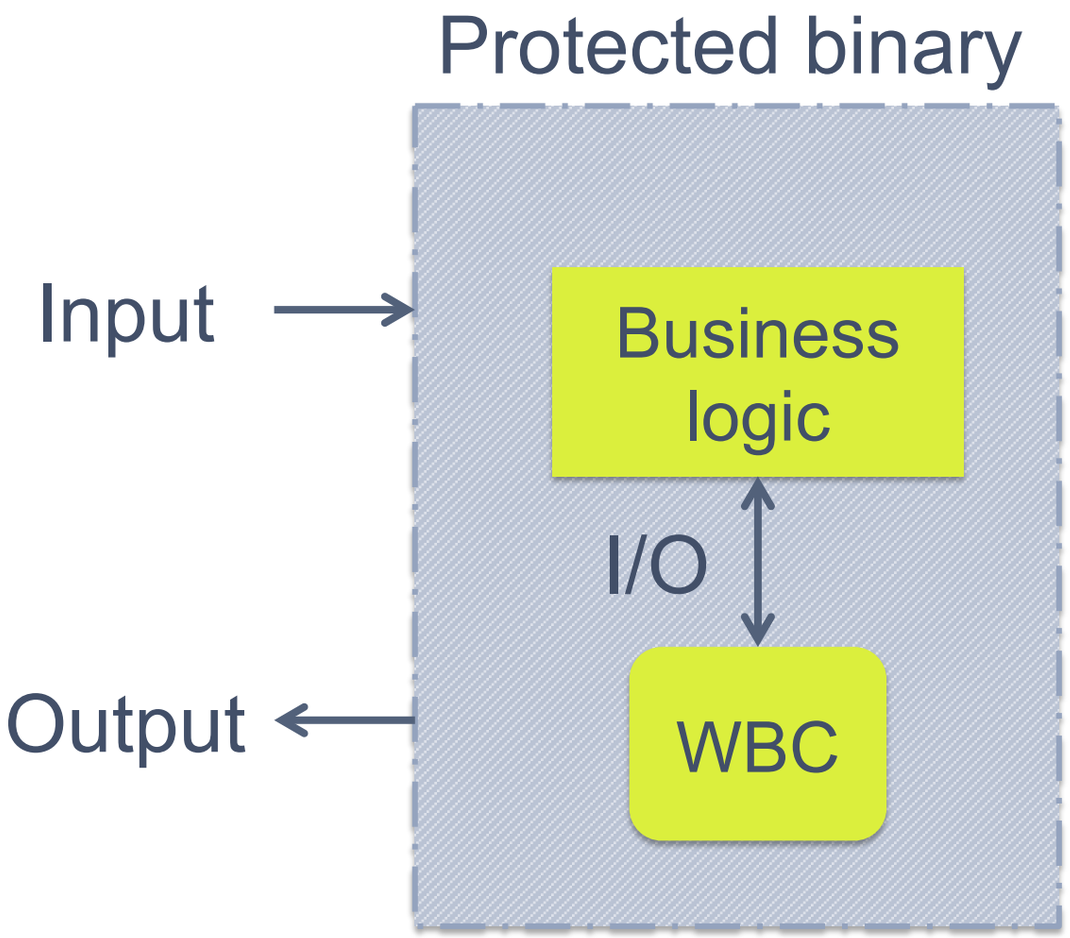
Side Channel Analysis attacks

- 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?

Mitigating WBC attacks



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

Take it away

- Like many security tech, WBC **mitigates** attacks
- Key extraction effort we've seen: days to few months

- WBC designers:
  - Validate your assumptions(!)
  - Always combine strong RE / cloning countermeasures
- WBC integrators:
  - Ask for all countermeasures
  - Ask for quantified attack resistance against all known attacks

riscure

Challenge your security

Contact: Jasper van Woudenberg (@jzvw)

vanwoudenberg@na.riscure.com

Riscure B.V.
Frontier Building, Delftechpark 49
2628 XJ Delft
The Netherlands
Phone: +31 15 251 40 90

www.riscure.com

Riscure North America
71 Stevenson Street, Suite 400
San Francisco, CA 94105
USA
Phone: +1 650 646 99 79

inforequest@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