Recent Progress on White-Box Attacks

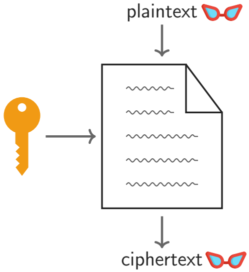
Junwei Wang

Journée “Protection du Code et des Données”

Paris Saclay, Dec 13th 2018

CryptoExperts
Université du Luxembourg
Université Paris 8 Vincennes-Saint-Denis

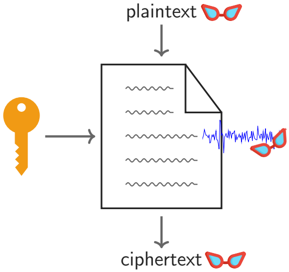
White-Box Treat Model



black-box model

knowing the cipher
observing I/O behavior

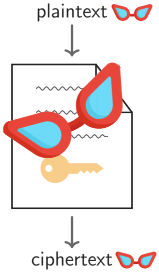
e.g. linear/differential cryptanalysis



gray-box model

+ side-channel leakages
(power/EM/time/$\cdots$)

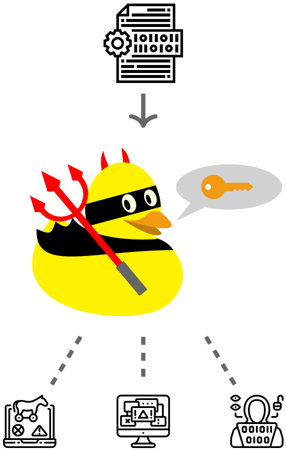
e.g. differential power analysis [KJJ99]



white-box model [CEJvO02]

owing the binary
controlling the environment

White-Box Treat Model



- **Goal:** to extract a cryptographic key, $\cdots$
- **Where:** from a software impl. of cipher
- **Who:**
  - malwares
  - co-hosted applications
  - user themselves
  - $\cdots$
- **How:** (*by all kinds of means*)
  - analyze the code
  - spy on the memory
  - interfere the execution
  - $\cdots$

Typical Applications

Digital Content Distribution

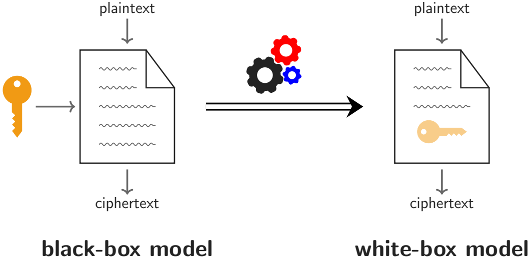
videos, musics, games, e-books, $\cdots$

Host Card Emulation

mobile payment without a *secure element*

White-Box Compiler

A **white-box complier** takes as input a *secret key* and generates a "white-box secure" program implementing some specific crypto. algo. with the specified secret key.



"white-box security" [DLPR13]

- Unbreakability (**this talk**)
- One-wayness
- Incompressibility
- Traceability

No provably secure white-box complier for standard block ciphers is known.

Cryptographic Obfuscation

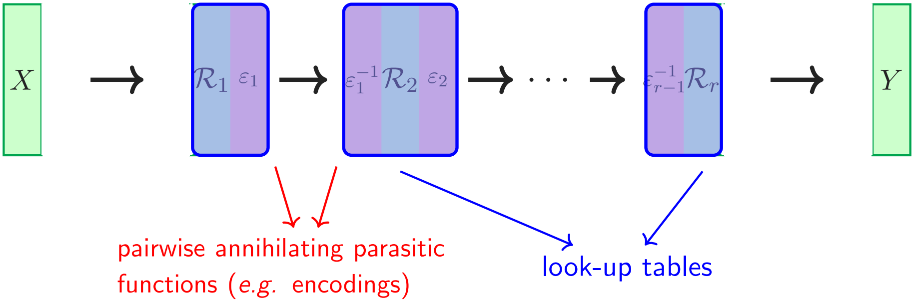
An **obfuscator** makes programs "unintelligible" while preserving their functionalities.

- **Virtual Black-Box** (VBB) **Obfuscation**
  - Nothing is learned from the obfuscated programs except their I/Os.
  - (**Impossibility**) VBB is impossible in general! [BGI+01]
  - VBB for point functions exist. [Wee05]
  - *Can we VBB obfuscate a block cipher ?*
- **Indistinguishability Obfuscation** (iO)
  - Literally, it hides the origin of an obfuscated program
  - Has many implications [SW14]
  - Candidate constructions exist [GGH+13,···]
  - *Does not imply unbreakability directly !*

Overview

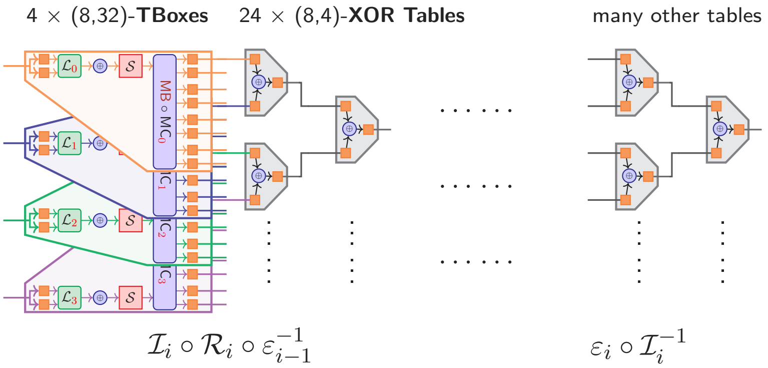
1. White-Box Context
2. Practical Countermeasures and Attacks
3. Showcase: Break A White-Box Implementation
4. Study of *Differential Computation Analysis*

Practical White-Box Compiler: Sketch



1. Represent the cipher into a *network* of transformations
2. Obfuscate the network by encoding *adjacent* transformations
3. Store the encoded transformations into look-up tables

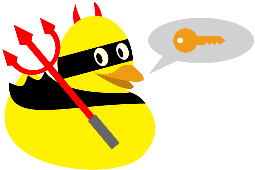
Illustration: Protect One AES Column [CEJvO02]



14KB memory and 56 table look-ups needed to compute $\varepsilon_i \circ \mathcal{R}_i \circ \varepsilon_{i-1}^{-1}$

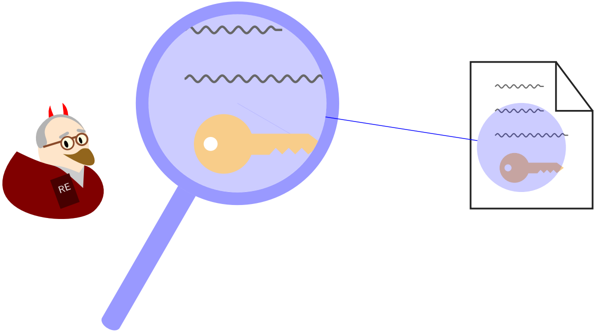
[1] The i-th round function $\mathcal{R}_i = \text{MC} \circ \text{SB} \circ \text{ARK}_i$ and $\mathcal{I}_i$ represents the intermediate encoding

White-Box Attacks



- Specific attacks
- Generic attacks
- Combined analyses

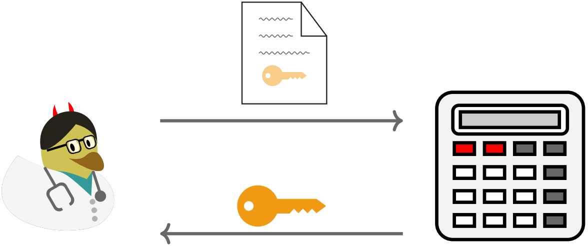
Specific Attacks



- to (partially) recover the design of a particular impl.
- usually by reverse engineering
- requiring skilled experts
- time-consuming

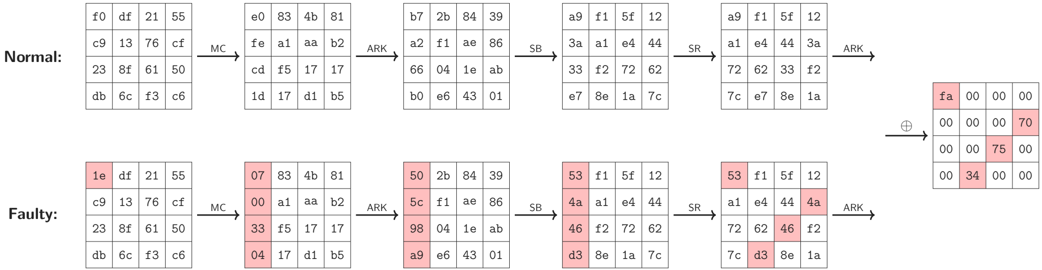
Trending: secret design paradigm *a.k.a* security through obscurity

Generic Attacks



- Generic and automatic
- Without knowing the protections
- *e.g. differential computation attacks* (DCA) and *differential fault attacks* (DFA)

Differential Fault Attack against AES



- Modify a state byte between last two `MixColumns`
  - How: statically / dynamically
  - Expecting certain differential patterns (thanks to ShiftRow)
- Very few faulty executions are required to recover a column of key bytes

A Showcase

Break the Winning Implementation of *CHES* 2017 CTF

– joint work with Louis Goubin, Pascal Paillier, Matthieu Rivain

CHES 2017 Capture the Flag Challenge

The WhibOx Contest

An ECRYPT White-Box Cryptography Competition

WhibOx Contest

- **Goal**: confront designers and attackers in the secret design paradigm
- **Designers**: invited to submit AES-128 implementations in C
  - with secret chosen key
  - source code $\leq$ 50MB
  - compiled binary $\leq$ 20MB
  - RAM consumption $\leq$ 20MB
  - execution time $\leq$ 1 second
- **Breakers**: invited to recover the hidden keys
- *Not required to* disclose their identities & underlying techniques

WhibOx Contest

- The competition lasted for about 4 months.
- Results:
  - 94 submissions were **all broken** by 877 individual breaks
  - Most (86%) of them were alive for < 1 day
- Scoreboard (top 5): ranked by **surviving time**

id	designer	first breaker	score	#days	#breaks
777	cryptolux	team_cryptoexperts	406	28	1
815	grothendieck	cryptolux	78	12	1
753	sebastien-riou	cryptolux	66	11	3
877	chaes	You!	55	10	2
845	team4	cryptolux	36	8	2

🎉 **cryptolux**: Biryukov, Udovenko
🎉 **team_cryptoexperts**: Goubin, Paillier, Rivain, Wang

The Winning Implementation

- Multi-layer protections
  - **Inner**: encoded Boolean circuit with error detection
  - **Middle**: bitslicing
  - **Outer**: virtualization, randomly naming, duplications, dummy operations

- Code size: ~28 MB
- Code lines: ~2.3k
- 12 global variables:
  - `pDeoW`: computation state (2.1 MB)
  - `JGNNvi`: program bytecode (15.3 MB)

available at: `https://whibox-contest.github.io/show/candidate/777`

The Winning Implementation

∼1200 functions: simple but obfuscated

- An array of pointers: to 210 useful functions
- Semantically equivalent to 20 different functions
  - bitwise operations, bit shifts
  - table look-ups, assignment
  - control flow primitives
  - ...

```
void xSnEq (uint UMNsVLp, uint KtFY, uint vzJZq) {
  if (nIlajqq () == IFWBUN (UMNsVLp, KtFY))
    EWwon (vzJZq);
}

void rNUiPyD (uint hFqeIO, uint jvXpt) {
  xkpRp[hFqeIO] = MXRIWZQ (jvXpt);
}

void cQnB (uint QRFOf, uint CoCiI, uint aLPxnn) {
  ooGoRv[(kIKfgI + QRFOf) & 97603] =
    ooGoRv[(kIKfgI + CoCiI) | 173937] & ooGoRv[(kIKfgI + aLPxnn) | 39896];
}

uint dLJT (uint RouDUC, uint TSCaTl) {
  return ooGoRv[763216 ul] | qscwtK (RouDUC + (kIKfgI << 17), TSCaTl);
}
```

Attack Overview

1. Reverse engineering $\Rightarrow$ a Boolean circuit
   - readability preprocessing
     - functions / variables renaming
     - redundancy elimination
     - ...
   - de-virtualization $\Rightarrow$ a bitwise program
   - simplification $\Rightarrow$ a Boolean circuit
2. Single static assignment (SSA) transformation
3. Circuit minimization
4. Data dependency analysis
5. Key recovery with algebraic analysis

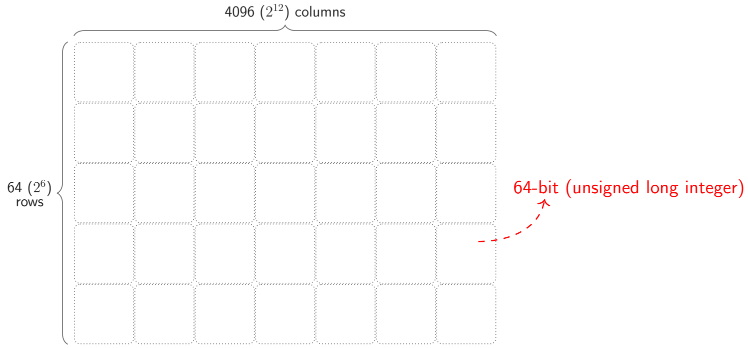
De-Virtualization

```c
char program[] = "...";          // 15.3 MB bytecode
void * funcptrs = "...";         // 210 function pointers

void interpretor() {
  uchar *pc = (uchar *) program;
  uchar *eop = pc + sizeof (program) / sizeof (uchar);
  while (pc < eop) {
    uchar args_num = *pc++;
    void (*fp) ();
    fp = (void *) funcptrs[*pc++];
    uint *arg_arr = (uint *) pc;
    pc += args_num * 8;
    if      (args_num == 0) { fp(); }
    else if (args_num == 1) { fp(arg_arr[0]); }
    else if (args_num == 2) { fp(arg_arr[0], arg_arr[1]); }
    // similar to args_num = 3, 4, 5, 6
  }
}
```

simulate VM $\Longrightarrow$ *bitwise* program with many loops of 64 cycles

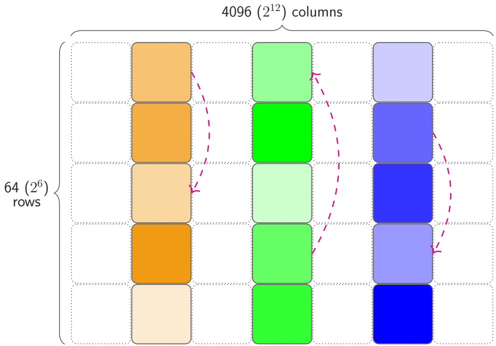
Computation State



global table of 2^{18} elements

$(= 64 \cdot 4096)$

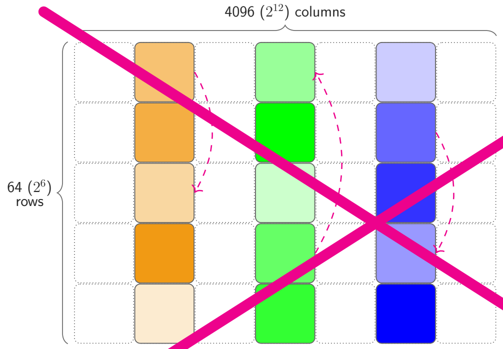
Bitwise Loops



Several loops only implement value **swap**s inside columns

CRYPTOEXPERTS

Bitwise Loops



Several loops only implement value **swaps** inside columns

Can be removed!

CryptoExperts

Obtaining Boolean Circuit

- A sequence of 64-cycle (non-overlapping) loops over 64-bit variables
  - **beginning**: 64 (cycles)$\times$64 (word length) bitslicing program
  - **before ending**: bit combination
  - **ending**: (possibly) error detection
- 64$\times$64 independent AES computations in parallel
  - Odd (3) number of them are real and identical
  - The rest use hard-coded fake keys
- Pick one real impl. $\Rightarrow$ a Boolean circuit with $\sim$**600k** gates

Single Static Assignment Form

$$
\begin{aligned}
x &= \cdots \\
y &= \cdots \\
z &= \neg x \\
x &= z \oplus y \\
y &= y \vee z \\
z &= x \vee y \\
&\vdots
\end{aligned}
\quad \Rightarrow \quad
\begin{aligned}
t_1 &= \cdots \\
t_2 &= \cdots \\
t_3 &= \neg t_1 \\
t_4 &= t_3 \oplus t_2 \\
t_5 &= t_2 \vee t_3 \\
t_6 &= t_4 \vee t_5 \\
&\vdots
\end{aligned}
$$

Each variable is only assigned once!

Circuit Minimization

Detect (over many executions) and remove:

- **Constant**:

$$t_i = 0 \text{ or } t_i = 1?$$

- **Duplicate**: *keep only one copy*

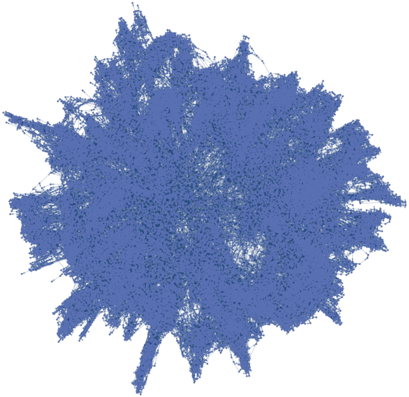
$$t_i = t_j?$$

- **Pseudorandomness**:

$$t_i \leftarrow t_i \oplus 1 \Rightarrow \text{same result}$$

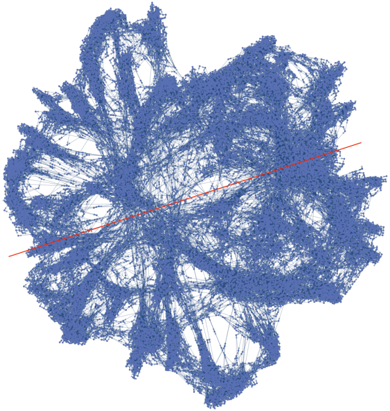
After several rounds, ∼600k ⇒∼280k gates **(53% smaller)**

Data Dependency Analysis



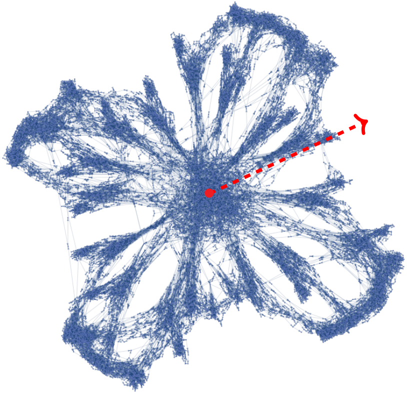
Data dependency graph (first 20% of the circuit)

Data Dependency Analysis



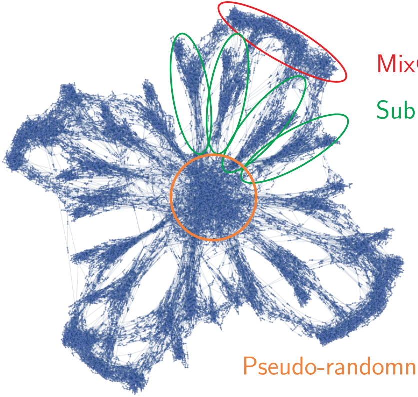
Data dependency graph (first 10% of the circuit)

Data Dependency Analysis



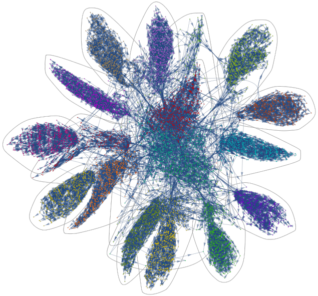
Data dependency graph (first 5% of the circuit)

Data Dependency Analysis



Data dependency graph (first 5% of the circuit)

Cluster Analysis



- Cluster $\Rightarrow$ variables in one SBox
- Identify outgoing variables:

$$s_1, s_2, \cdots, s_n$$

- Heuristically,

$$S(x \oplus k^*) = D(s_1, s_2, \cdots, s_n)$$

for some deterministic decoding function D.

Key Recovery

- **Hypothesis**: linear decoding function

$$D(s_1, s_2, \cdots, s_n) = a_0 \oplus \left(\bigoplus_{1 \le i \le n} a_i s_i \right)$$

  for some fixed coefficients $a_0, a_1, \cdots, a_n$.

- Record the s_i's over T executions:

$$\begin{bmatrix} 1 & s_1^{(1)} & \cdots & s_n^{(1)} \\ 1 & s_1^{(2)} & \cdots & s_n^{(2)} \\ 1 & \vdots & \ddots & \vdots \\ 1 & s_1^{(T)} & \cdots & s_n^{(T)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} S(x^{(1)} \oplus k)[j] \\ S(x^{(2)} \oplus k)[j] \\ \vdots \\ S(x^{(T)} \oplus k)[j] \end{bmatrix}$$

- Linear system solvable for $k = k^*$

Key Recovery

- And it works! For instance,
  - a cluster with 34 outgoing in 504 total points
  - collecting 50 computation traces
  - no solution for the $k \neq k^*$
  - one solution for each j for the $k = k^*$

$$
\begin{array}{l}
j = 0\text{: } 0,0,0,0,0,0 \\
j = 1\text{: } 0,0,0,0,0,0 \\
j = 2\text{: } 0,0,0,0,0,0 \\
j = 3\text{: } 0,0,0,0,0,0 \\
j = 4\text{: } 0,0,0,0,0,0 \\
j = 5\text{: } 0,0,0,0,0,0 \\
j = 6\text{: } 0,0,0,0,0,0 \\
j = 7\text{: } 0,0,0,0,0,0
\end{array}
\left[\begin{array}{l}
1,0,1,0,1,1,1,0,0,0,1,0,1,0,1 \\
1,0,0,1,1,0,1,1,1,1,1,1,1,0,0 \\
0,0,1,0,1,0,0,0,1,1,1,0,1,1,1 \\
0,0,0,1,1,0,0,0,1,1,1,0,1,1,1 \\
0,1,1,0,0,1,0,0,0,0,0,0,1,1,1 \\
0,0,0,0,1,0,0,0,0,0,0,0,0,0,1 \\
1,0,0,0,1,0,0,1,0,1,0,1,0,1,0 \\
0,1,0,0,0,0,1,0,0,1,1,0,0,0,0
\end{array}\right]
\begin{array}{l}
0,0,0,0,0,0,0,0,0,0,0,0,0,0 \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0 \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0 \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0 \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0 \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0 \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0 \\
0,0,0,0,0,0,0,0,0,0,0,0,0,0
\end{array}
$$

(15 × 8) binary matrix

$$
\underbrace{[s_7, s_8, \cdots, s_{21}]}_{\text{15 encoding variables}} \times M = \underbrace{[S(x \oplus k^*)[0], \cdots, S(x \oplus k^*)[7]]}_{\text{8 S-Box output bits}}
$$

- Repeat with remaining clusters... (14 subkeys recoverd)

Lesson Learned

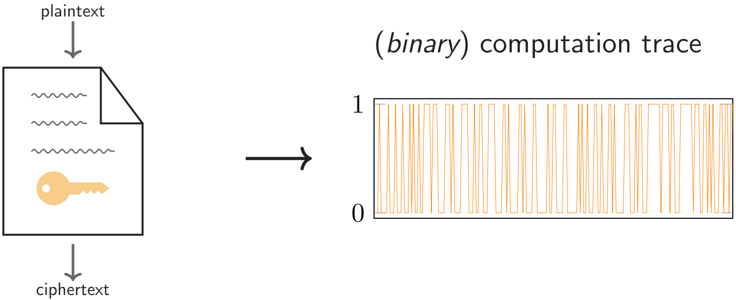
Security through obscurity is the only hope for industrial white-box demands currently, but it could be fragile in front of a motivated and skilled attacker.

Generic Attacks

A Study of *Differential Computation Analysis*

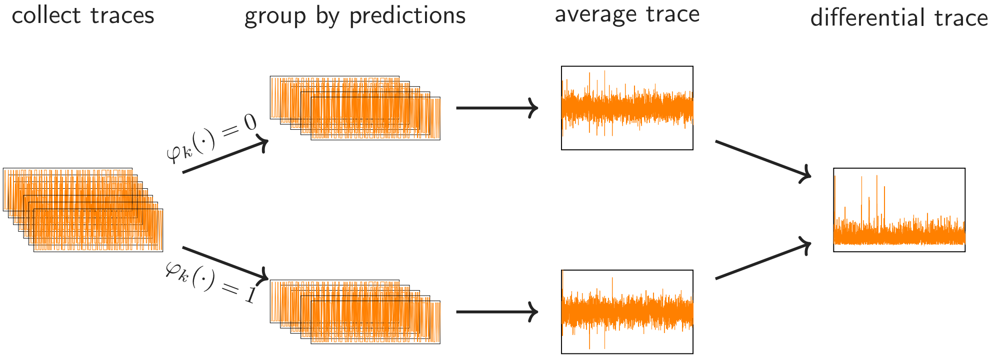
– joint work with Matthieu Rivain

Differential Computation Analysis (DCA)



- DPA techniques in white-box context [BHMT16]
- Instead of *power traces*, using *computation traces* usually consisting of runtime memory information
- Breaks many white-box designs

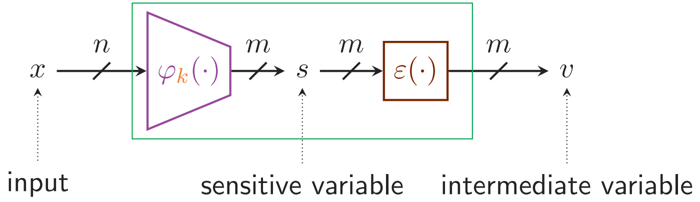
DCA Techniques



DCA Attack Limitations

1. Lack of in-depth understanding
   - *Only* known to work on **nibble** encodings [BBMT18]
   - *Only* known to work on the **first** and **last** rounds
   - Most results are only experimental and DCA success probability is unknown
2. Suboptimal exploitation of the information in the computation traces

Internal Encoding : Abstraction

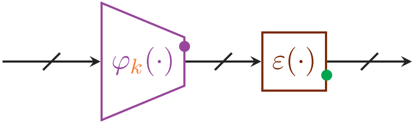


- A *key-dependent* (n, m) function φ_k in a block cipher
- A *random* selected m-bit bijection ε
- $\varepsilon \circ \varphi_k$, **leaked in the memory**, is an output of some **table look-up**
- To exploit the leakage of $\varepsilon \circ \varphi_k$, $n > m$ is necessary

DCA against Internal Encoding

Based on well-established theory – *Boolean correlation*, instead of *difference of means*: for any key guess k

$$\rho_k = \mathrm{Cor}\Big(\ \varphi_k(\cdot)[i]\ \ ,\ \ \varepsilon \circ \varphi_{k^*}(\cdot)[j]\ \Big)$$



CryptoExperts

ρ_{k^*} and $\rho_{k^\times}$: Distributions

- **Ideal** assumption: $(\varphi_k)_k$ are mutually independent random (n, m) functions

Correct key guess k^*,

$$\rho_{k^*} = 2^{2-m} N^* - 1$$

where

$$N^* \sim \mathcal{HG}(2^m, 2^{m-1}, 2^{m-1}) \ .$$

Only depends on m.

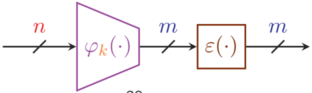
Incorrect key guess $k^\times$,

$$\rho_{k^\times} = 2^{2-n} N^\times - 1$$

where

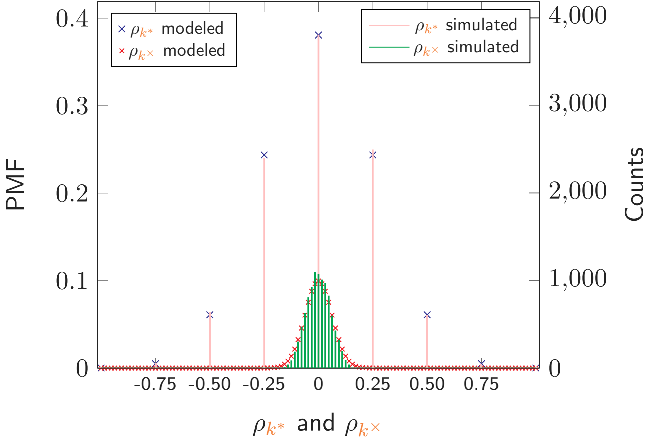
$$N^\times \sim \mathcal{HG}(2^n, 2^{n-1}, 2^{n-1}) \ .$$

Only depends on n.



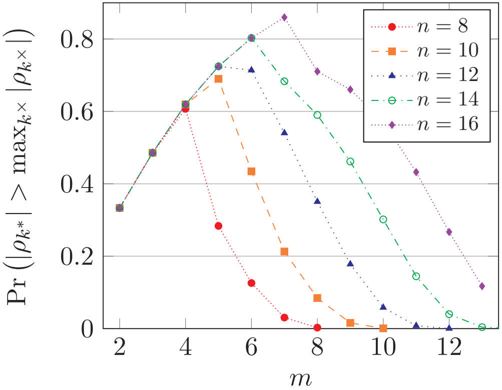
ρ_{k^*} and $\rho_{k^\times}$: Distributions

- Theoretical results and simulations when $n = 8$ and $m = 4$



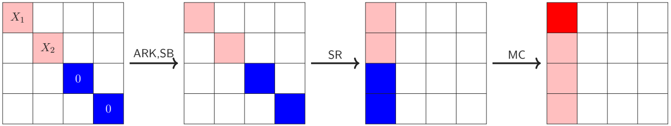
DCA Success Rate

- DCA success (roughly) requires: $\left|\rho_{k^*}\right| > \max_{k^\times} \left|\rho_{k^\times}\right|$.



Attack a NSC Variant: a White-Box AES

- *Byte* encoding protected
- DCA has failed to break it *before this work*
- Our approach: target a output byte of MixColumn in the first round



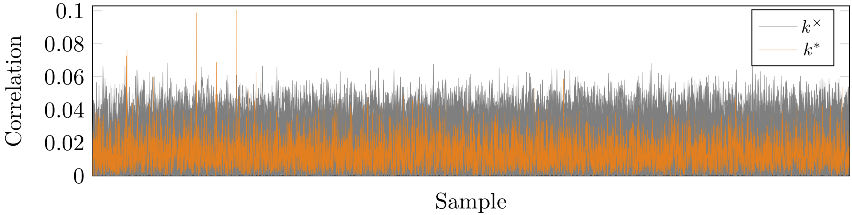
$$\varphi_{k_1||k_2}(x_1||x_2) = 2 \cdot \mathbf{Sbox}(x_1 \oplus k_1) \oplus 3 \cdot \mathbf{Sbox}(x_2 \oplus k_2) \oplus \mathbf{Sbox}(k_3) \oplus \mathbf{Sbox}(k_4)$$

$$\varepsilon' = \varepsilon \circ \oplus_c \ ,$$

$$n = 16, m = 8 \ , |\mathcal{K}| = 2^{16}.$$

Attack a NSC Variant: a White-Box AES

- Attack results: $\sim$ 1800 traces



- Same attack works on the "masked" implementation [LKK18] (intending to resist DCA) as well.

CryptoExperts

Summary

- White-box adversary models the real security treats in many software applications deployed in the real world.
- No provably white-box secure construction is known for standard block ciphers.
- Industrial trending: security through obscurity, which could be fragile in front of motivated and skilled attackers.
- DCA against internal encoding has been analyzed in-depth.
  - it is able to breaker “wider” encodings in “deeper” rounds.
- What can we hope for white-box cryptography?

WhibOx News

- *WhibOx* competition returns
  - expected to start from the beginning of **February 2019**
  - until the end of August 2019
  - `https://whibox-contest.slack.com/`
- The 2nd *WhibOx* workshop will take place in May 18-19, 2019.
  - organized by Chris Brzuska and Pascal Paillier
  - affiliated to Eurocrypt 2019 (Darmstadt, Germany)
  - including talks on all aspects (theory, attacks, design techniques)
  - and a hands-on session dedicated to attack tools and demos

Thank you!