How to Reveal the Secrets of an Obscure White-Box Implementation

Louis Goubin[4] Pascal Paillier[1]
Matthieu Rivain[1] **Junwei Wang**[1,2,3]

[1]CryptoExperts
[2]University of Luxembourg
[3]University of Paris 8
[4]University of Versailles-St-Quentin-en-Yvelines

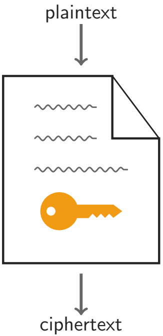
RWC 2018, Zurich

Outline

1. White-Box Cryptography
2. *WhibOx* Contest
3. The Winning Implementation (777)
4. Unveiling the Secrets

Outline

White-Box Cryptography



- Resistant against **key extraction** in the **worst case** [SAC02]
- No *provably secure* construction
- All *practical* schemes in the literature are **heuristic**, and are **vulnerable** to generic attacks [CHES16,BlackHat15]
- Applications: DRM and mobile payment

rapid growth of market

⇓

home-made solutions
(security through obscurity!)

Outline


CHES 2017
Capture the Flag
Challenge
The WhibOx Contest
An ECRYPT White-Box Cryptography Competition

WhibOx Contest - CHES 2017 CTF

- The idea is to invite
  - **designers**: to submit challenges implementing AES-128 in C
  - **breakers**: to recover the hidden keys
- *Not required to* disclose their identity & underlying techniques
- 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

Outline

1. White-Box Cryptography
2. *WhibOx* Contest
3. The Winning Implementation (777)
4. Unveiling the Secrets

The Winning Implementation

- Multi-layer protection
  - **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

```
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);
}
```

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

Outline

1. White-Box Cryptography
2. *WhibOx* Contest
3. The Winning Implementation (777)
4. Unveiling the Secrets

Unveiling the Secrets

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

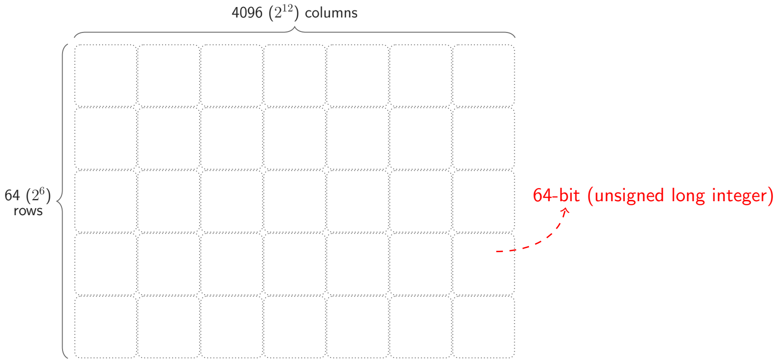
De-Virtualization

```
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 a large number of 64-cycle loops

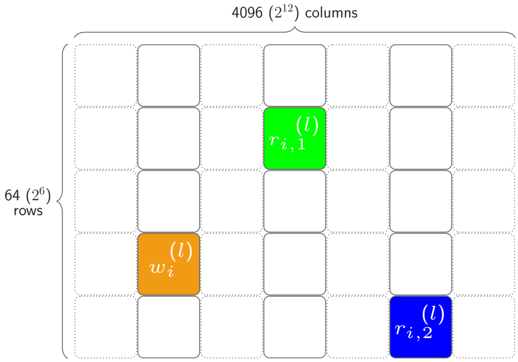
Computation State



global table of 2^{18} elements

$(= 64 \cdot 4096)$

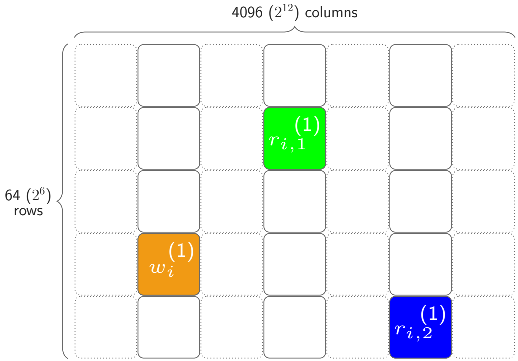
Bitwise Loops



$$l = 1, 2, 3, \cdots, 64$$

$$T[w_1^{(l)}] = T[r_{1,1}^{(l)}] \oplus T[r_{1,2}^{(l)}];$$
$$T[w_2^{(l)}] = T[r_{2,1}^{(l)}] \wedge T[r_{2,2}^{(l)}];$$
$$\vdots$$
$$T[w_i^{(l)}] = T[r_{i,1}^{(l)}] \oplus T[r_{i,2}^{(l)}];$$
$$\vdots$$

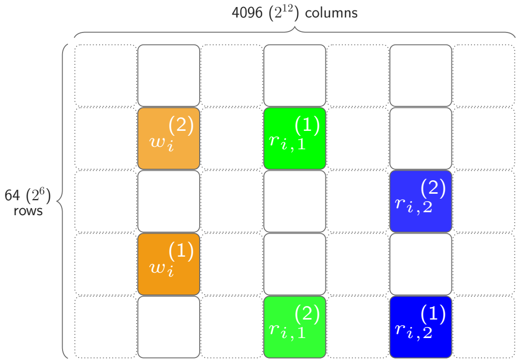
Bitwise Loops



$$l = 1, 2, 3, \cdots, 64$$

$$T[w_i^{(1)}] = T[r_{i,1}^{(1)}] \oplus T[r_{i,2}^{(1)}];$$

Bitwise Loops

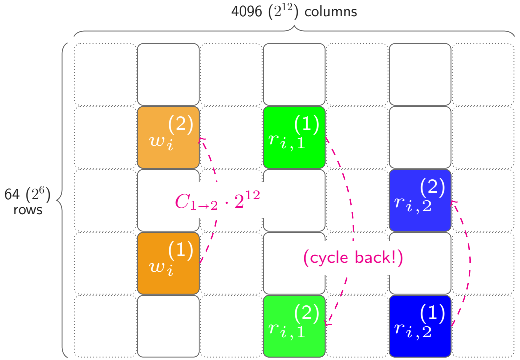


$l = 1, 2, 3, \cdots, 64$

$T[w_i^{(1)}] = T[r_{i,1}^{(1)}] \oplus T[r_{i,2}^{(1)}];$

$T[w_i^{(2)}] = T[r_{i,1}^{(2)}] \oplus T[r_{i,2}^{(2)}];$

Bitwise Loops



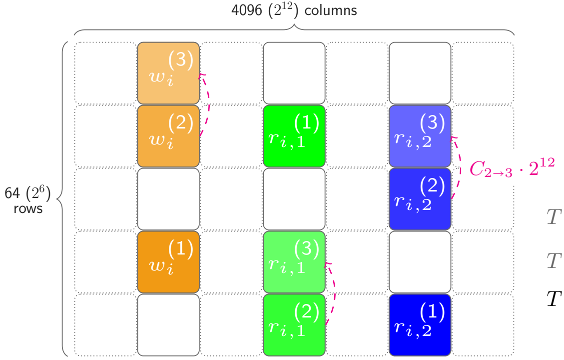
$l = 1, 2, 3, \cdots, 64$

$$T[w_i^{(1)}] = T[r_{i,1}^{(1)}] \oplus T[r_{i,2}^{(1)}];$$

$$T[w_i^{(2)}] = T[r_{i,1}^{(2)}] \oplus T[r_{i,2}^{(2)}];$$

$$w_1^{(2)} - w_i^{(1)} \equiv r_{i,1}^{(2)} - r_{i,1}^{(1)} \equiv r_{i,2}^{(2)} - r_{i,2}^{(1)} \equiv C_{1\to2} \cdot 2^{12} \bmod 2^{18}$$

Bitwise Loops



$$l = 1, 2, \mathbf{3}, \cdots, 64$$

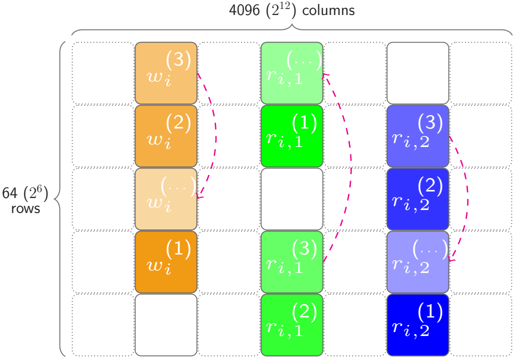
$$T[w_i^{(1)}] = T[r_{i,1}^{(1)}] \oplus T[r_{i,2}^{(1)}];$$

$$T[w_i^{(2)}] = T[r_{i,1}^{(2)}] \oplus T[r_{i,2}^{(2)}];$$

$$T[w_i^{(3)}] = T[r_{i,1}^{(3)}] \oplus T[r_{i,2}^{(3)}];$$

$$w_1^{(3)} - w_i^{(2)} \equiv r_{i,1}^{(3)} - r_{i,1}^{(2)} \equiv r_{i,2}^{(3)} - r_{i,2}^{(2)} \equiv C_{2\to3} \cdot 2^{12} \mod 2^{18}$$

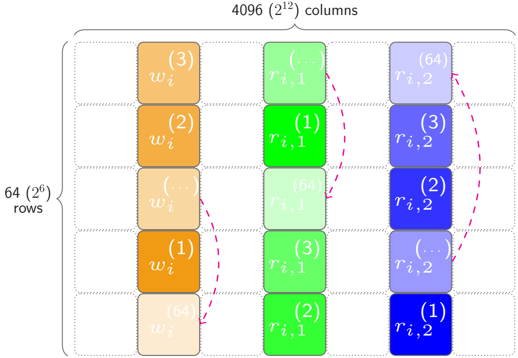
Bitwise Loops



$$l = 1, 2, 3, \cdots, 64$$

$$T[w_i^{(1)}] = T[r_{i,1}^{(1)}] \oplus T[r_{i,2}^{(1)}];$$
$$T[w_i^{(2)}] = T[r_{i,1}^{(2)}] \oplus T[r_{i,2}^{(2)}];$$
$$T[w_i^{(3)}] = T[r_{i,1}^{(3)}] \oplus T[r_{i,2}^{(3)}];$$
$$\vdots$$

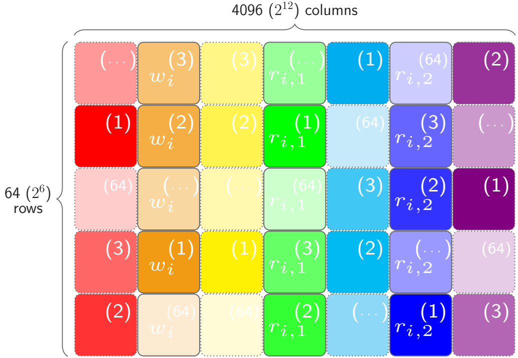
Bitwise Loops



$$l = 1, 2, 3, \cdots, 64$$

$$T[w_i^{(1)}] = T[r_{i,1}^{(1)}] \oplus T[r_{i,2}^{(1)}];$$
$$T[w_i^{(2)}] = T[r_{i,1}^{(2)}] \oplus T[r_{i,2}^{(2)}];$$
$$T[w_i^{(3)}] = T[r_{i,1}^{(3)}] \oplus T[r_{i,2}^{(3)}];$$
$$\vdots$$
$$T[w_i^{(64)}] = T[r_{i,1}^{(64)}] \oplus T[r_{i,2}^{(64)}];$$

Bitwise Loops



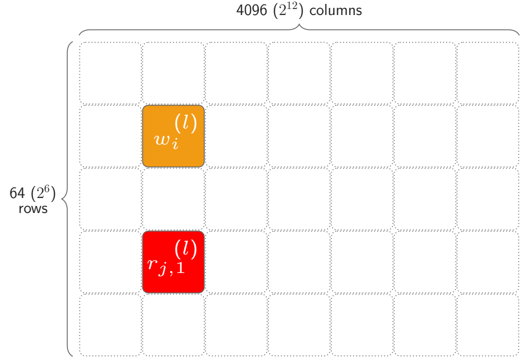
$$l = 1, 2, 3, \cdots, 64$$

$$T[w_1^{(l)}] = T[r_{1,1}^{(l)}] \oplus T[r_{1,2}^{(l)}];$$
$$T[w_2^{(l)}] = T[r_{2,1}^{(l)}] \wedge T[r_{2,2}^{(l)}];$$
$$\vdots$$
$$T[w_i^{(l)}] = T[r_{i,1}^{(l)}] \oplus T[r_{i,2}^{(l)}];$$
$$\vdots$$
$$T[w_j^{(l)}] = T[r_{j,1}^{(l)}] \oplus T[r_{j,2}^{(l)}];$$
$$\vdots$$

$$\forall i, j : w_i^{(l+1)} - w_i^{(l)} \equiv w_j^{(l+1)} - w_j^{(l)} \equiv C_{l \to l+1} \cdot 2^{12} \bmod 2^{18}, \text{ where } 1 \le l \le 63$$

Bitwise Loops

Memory Overlapping

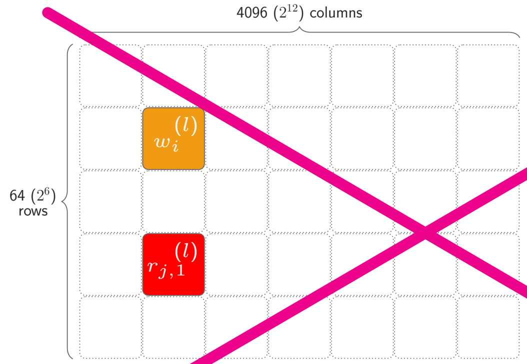


$$l = 1, 2, 3, \cdots, 64$$

$$T[w_1^{(l)}] = T[r_{1,1}^{(l)}] \oplus T[r_{1,2}^{(l)}];$$
$$T[w_2^{(l)}] = T[r_{2,1}^{(l)}] \wedge T[r_{2,2}^{(l)}];$$
$$\vdots$$
$$T[w_i^{(l)}] = T[r_{i,1}^{(l)}] \oplus T[r_{i,2}^{(l)}];$$
$$\vdots$$
$$T[w_j^{(l)}] = T[r_{j,1}^{(l)}] \oplus T[r_{j,2}^{(l)}];$$
$$\vdots$$

Only implementing swap(w_i,$r_{j,1}$)

Bitwise Loops



$$l = 1, 2, 3, \cdots, 64$$

$$T[w_1^{(l)}] = T[r_{1,1}^{(l)}] \oplus T[r_{1,2}^{(l)}];$$

$$T[w_2^{(l)}] = T[r_{2,1}^{(l)}] \wedge T[r_{2,2}^{(l)}];$$

$$\vdots$$

$$T[w_i^{(l)}] = T[r_{i,1}^{(l)}] \oplus T[r_{i,2}^{(l)}];$$

$$\vdots$$

$$T[w_j^{(l)}] = T[r_{j,1}^{(l)}] \oplus T[r_{j,2}^{(l)}];$$

$$\vdots$$

Only implementing swap(w_i,$r_{j,1}$)

Can be removed!

Obtaining Boolean Circuit

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

Single Static Assignment Form

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

Each address is only assigned once!

Circuit Minimization

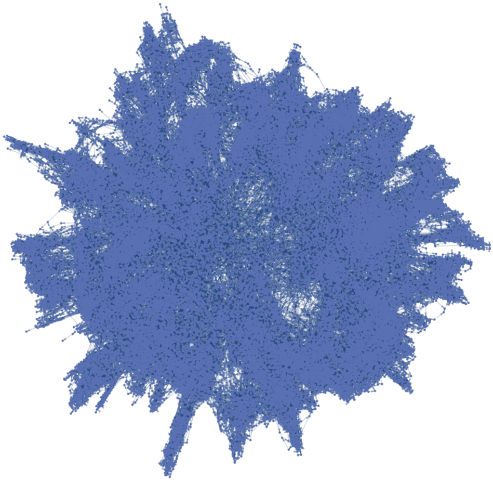
Detect (over many executions) and remove:

- constant: $t_i = 0$ or $t_i = 1$?
- duplicate: $t_i = t_j$? (keep only one copy)
- pseudorandomness:

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

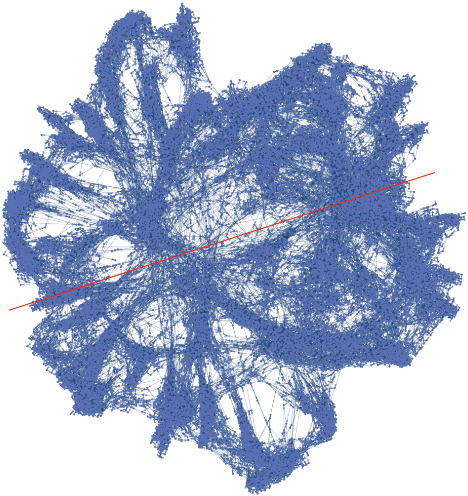
After several rounds, ∼600k ⇒∼280k gates **(53% smaller)**

Data Dependency Analysis



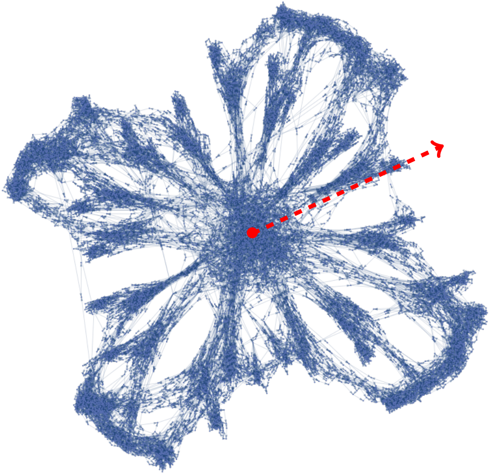
Data dependency graph (first 20% of the circuit)

Data Dependency Analysis



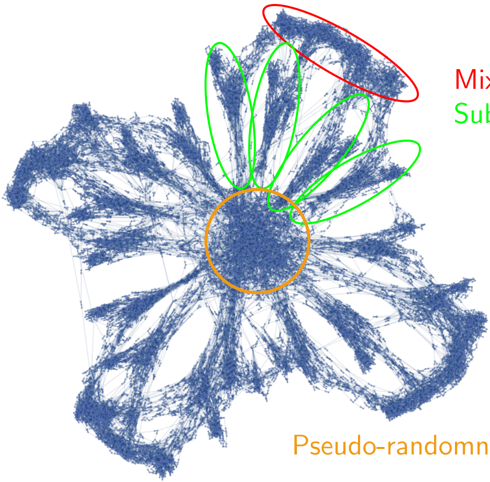
Data dependency graph (first 10% of the circuit)

Data Dependency Analysis



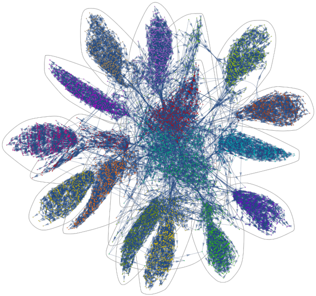
Data dependency graph (first 5% of the circuit)

Data Dependency Analysis



Data dependency graph (first 5% of the circuit)

Data Dependency 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:

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:

$$s_1^{(1)} \quad \cdots \quad s_n^{(1)} \qquad\qquad x^{(1)}$$

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:

$$\begin{matrix} s_1^{(1)} & \cdots & s_n^{(1)} & & x^{(1)} \\ s_1^{(2)} & \cdots & s_n^{(2)} & & x^{(2)} \end{matrix}$$

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:

$$\begin{matrix} s_1^{(1)} & \cdots & s_n^{(1)} & & x^{(1)} \\ s_1^{(2)} & \cdots & s_n^{(2)} & & x^{(2)} \\ \vdots & \ddots & \vdots & & \vdots \end{matrix}$$

Key Recovery

- Hypothesis: linear decoding function

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

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

- Record the s_i's over T executions:

$$\begin{matrix} s_1^{(1)} & \cdots & s_n^{(1)} & & x^{(1)} \\ s_1^{(2)} & \cdots & s_n^{(2)} & & x^{(2)} \\ \vdots & \ddots & \vdots & & \vdots \\ s_1^{(T)} & \cdots & s_n^{(T)} & & x^{(T)} \end{matrix}$$

Key Recovery

- Hypothesis: linear decoding function

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

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

- Record the s_i's over T executions:

$$\begin{matrix} s_1^{(1)} & \cdots & s_n^{(1)} \\ s_1^{(2)} & \cdots & s_n^{(2)} \\ \vdots & \ddots & \vdots \\ s_1^{(T)} & \cdots & s_n^{(T)} \end{matrix} \qquad \begin{matrix} S(x^{(1)} \oplus k)[j] \\ S(x^{(2)} \oplus k)[j] \\ \vdots \\ S(x^{(T)} \oplus k)[j] \end{matrix}$$

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 $k = k^*$

$j = 0$: 0,0,0,0,0,0,1,0,1,0,1,1,1,0,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 1$: 0,0,0,0,0,0,1,0,0,1,1,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 2$: 0,0,0,0,0,0,0,0,1,0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 3$: 0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 4$: 0,0,0,0,0,0,0,1,1,0,0,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 5$: 0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 6$: 0,0,0,0,0,0,1,0,0,0,1,0,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 7$: 0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

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 $k = k^*$

$j = 0$: 0,0,0,0,0,0 [1,0,1,0,1,1,1,0,0,0,1,0,1,0,1] 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 1$: 0,0,0,0,0,0 [1,0,0,1,1,0,1,1,1,1,1,1,1,0,0] 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 2$: 0,0,0,0,0,0 [0,0,1,0,1,0,0,0,1,1,1,0,1,1,1] 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 3$: 0,0,0,0,0,0 [0,0,0,1,1,0,0,0,1,1,1,0,1,1,1] 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 4$: 0,0,0,0,0,0 [0,1,1,0,0,1,0,0,0,0,0,0,1,1,1] 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 5$: 0,0,0,0,0,0 [0,0,0,0,1,0,0,0,0,0,0,0,0,0,1] 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 6$: 0,0,0,0,0,0 [1,0,0,0,1,0,0,1,0,1,0,1,0,1,0] 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 7$: 0,0,0,0,0,0 [0,1,0,0,0,0,1,0,0,1,1,0,0,0,0] 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

(15×8) binary matrix

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

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 $k = k^*$

$j = 0$: 0,0,0,0,0,0 [1,0,1,0,1,1,1,0,0,0,1,0,1,0,1] 0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 1$: 0,0,0,0,0,0 [1,0,0,1,1,0,1,1,1,1,1,1,1,0,0] 0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 2$: 0,0,0,0,0,0 [0,0,1,0,1,0,0,0,1,1,1,0,1,1,1] 0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 3$: 0,0,0,0,0,0 [0,0,0,1,1,0,0,0,1,1,1,0,1,1,1] 0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 4$: 0,0,0,0,0,0 [0,1,1,0,0,1,0,0,0,0,0,0,1,1,1] 0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 5$: 0,0,0,0,0,0 [0,0,0,0,1,0,0,0,0,0,0,0,0,0,1] 0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 6$: 0,0,0,0,0,0 [1,0,0,0,1,0,0,1,0,1,0,1,0,1,0] 0,0,0,0,0,0,0,0,0,0,0,0,0,0
$j = 7$: 0,0,0,0,0,0 [0,1,0,0,0,0,1,0,0,1,1,0,0,0,0] 0,0,0,0,0,0,0,0,0,0,0,0,0,0

(15×8) binary matrix

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

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

CryptoExperts

Summary

- White-box cryptography
  - no realistic solution in the literature
  - increasing industrial demands $\Rightarrow$ home-made solution
- *WhibOx* contest was launched to increase openness and benchmark constructions/attacks
  - everything was eventually broken
  - (could be) only the tip of the iceberg!
- Our attacking techniques
  - smashed the winning design
  - illustrate that resisting against generic attacks is not sufficient
  - could also be generalized to attack impl. with higher-degree decoding functions

White paper: ia.cr/2018/098

Thank you!