# Attacks and Countermeasures for White-box Designs

Alex Biryukov, Aleksei Udovenko

CSC and SnT, University of Luxembourg

December 5, 2018

# Plan

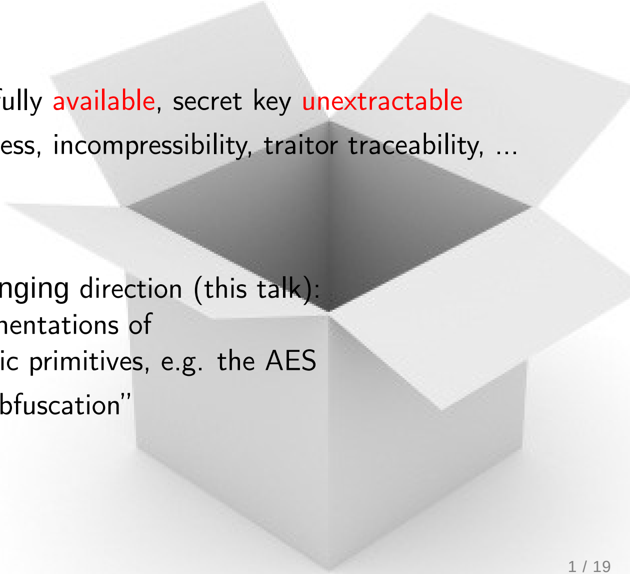
# White-box

- Implementation fully available, secret key unextractable
- Extra: one-wayness, incompressibility, traitor traceability, ...

# White-box

- Implementation fully available, secret key unextractable
- Extra: one-wayness, incompressibility, traitor traceability, ...

- The most challenging direction (this talk):
  white-box implementations of
  existing symmetric primitives, e.g. the AES
- "Cryptographic obfuscation"



# White-box: Industry vs Academia


# White-box: Industry vs Academia


- many applications
- strong need for *practical* white-box
- industry does WB: hidden designs

# White-box: Industry vs Academia


- many applications
- strong need for *practical* white-box
- industry does WB: hidden designs

- theory: approaches using iO/FE, currently *impractical*
- practical WB-AES: few attempts (2002-2017), all broken
- powerful DCA attack (CHES 2016)

# White-Box: Differential Computation Analysis (DCA)

- DCA = Differential Power Analysis (DPA)
  applied to white-box implementations
- Most of the implementations broken automatically

# White-Box: Differential Computation Analysis (DCA)

- DCA = Di erential Power Analysis (DPA)
  applied to white-box implementations
- Most of the implementations broken automatically
- Side-Channel protection: masking schemes

# White-Box: Differential Computation Analysis (DCA)

- DCA = Di□erential Power Analysis (DPA)
  applied to white-box implementations
- Most of the implementations broken automatically
- Side-Channel protection: masking schemes

*this talk*:

Can we apply the masking protection for white-box impl.?

# General Setting

- Boolean circuits
- Obfuscated reference implementation

# General Setting

- Boolean circuits
- Obfuscated reference implementation
- Predictable values: computations from ref. impl., e.g.

$$s = Bit_1(SBox(pt_1 \quad k_1))$$

# General Setting

- Boolean circuits
- Obfuscated reference implementation
- Predictable values: computations from ref. impl., e.g.

$$s = Bit_1(SBox(pt_1 \quad k_1))$$

- Masking: $\exists v_1, \dots, v_t$ nodes (*shares*), $f : \mathbb{F}_2^t \to \mathbb{F}_2$ s.t. for any encryption

$$f(v_1, \dots, v_t) = s$$

# Masking Schemes

- Example: Boolean masking: linear decoder $f = \bigoplus_i v_i$
- Example: FHE: non-linear decoder $f$

# Masking Schemes

- Example: Boolean masking: linear decoder $f = \bigoplus_i v_i$
- Example: FHE: non-linear decoder $f$
- Aim for efficient schemes: relatively small $t$ (number of shares)

# Masking Schemes

- Example: Boolean masking: linear decoder $f = \bigoplus_i v_i$
- Example: FHE: non-linear decoder $f$
- Aim for efficient schemes: relatively small $t$ (number of shares)

$\Rightarrow$ can be secure only if the locations of the shares in the circuit are unknown!

*this talk*: exploring this possibility

# Plan

# Attacks I

Combinatorial attacks:

- (partially) guess locations of the shares
- probabilistic : correlation with predictable values
- exact: time-memory trade-o

# Attacks I

Combinatorial attacks:

- (partially) guess locations of the shares
- probabilistic : correlation with predictable values
- exact: time-memory trade-o

Fault attacks:

- new application: recover locations of the shares
- 1- and 2- share fault injections
- applicability depends on protections

# Attacks II

(Generalized) Di erential Computation Analysis (DCA):

# Attacks II

(Generalized) Di erential Computation Analysis (DCA):

# Attacks II

(Generalized) Di erential Computation Analysis (DCA):

# The Linear Algebra Attack (1)

- consider the Boolean masking (the linear decoder)
- matching with a predictable value s:
  a basic linear algebra problem:

$$M \cdot z = s; \quad M = [\, v_1 \mid \ldots \mid v_n \,]$$

# The Linear Algebra Attack (1)

- consider the Boolean masking (the linear decoder)
- matching with a predictable value $s$:
  a basic linear algebra problem:

$$M \cdot z = s, \quad M = [\, v_1 \mid \cdots \mid v_n \,]$$

- $v_i$ is the vector of values computed in the node $i$ of the circuit
- $z$ is a vector indicating locations of shares among nodes of the circuit
- higher-order masking does not help...

# The Linear Algebra Attack (2)

Generalizations:

- nonlinear decoders, through linearization technique
- approximately linear decoders, through LPN algorithms

# The Linear Algebra Attack (2)

Generalizations:

- nonlinear decoders, through linearization technique
- approximately linear decoders, through LPN algorithms
- semi-linear decoders:
  1. assumes r is computed/shared in the circuit, where
  2. s is a predictable value
  3. r is unpredictable (pseudorandom, uniform)

# The Linear Algebra Attack (2)

Generalizations:

- nonlinear decoders, through linearization technique
- approximately linear decoders, through LPN algorithms
- semi-linear decoders:
  1. assumes $s \oplus r$ is computed/shared in the circuit, where
  2. $s$ is a predictable value
  3. $r$ is unpredictable (pseudorandom, uniform)
  4. choose plaintexts $p_1, \ldots, p_D$ such that:
     $s(p_i) = 0$ for $1 \leq i \leq D - 1$;
     $s(p_i) = 1$ for $i = D$.
  5. $s \oplus r$ will be equal to $(0, 0, \ldots, 0, 1)$ with $\Pr = 1/2$
  6. if $s$ is guessed wrong, such vector is unlikely to be a solution

# Plan

# Our Framework: Two Components

Value Hiding

Structure Hiding

# Our Framework: Two Components

Value Hiding

Structure Hiding

1. DCA side-channel attack
2. (new) linear algebra attack

# Our Framework: Two Components

Value Hiding

1. DCA side-channel attack
2. (new) linear algebra attack

Structure Hiding

1. circuit analysis / simpli cation
2. fault injections
3. pseudorandomness removal
4. etc.

# Our Framework: Two Components

Value Hiding

1. DCA side-channel attack
2. (new) linear algebra attack

Structure Hiding

1. circuit analysis / simpli cation
2. fault injections
3. pseudorandomness removal
4. etc.

(hopefully) easier to solve independently

# Value Hiding

Our solution for value hiding:

1. non-linear masking (vs linear algebra attack)
2. classic linear masking (vs DCA correlation attack)
3. provable security against the linear algebra attack

# Plan

# Algebraic Security (1/2)

Security Model:

1. random bits allowed
   - as in classic masking
   - model unpredictability
   - in WB impl. as pseudorandom

# Algebraic Security (1/2)

Security Model:

1. random bits allowed
   - as in classic masking
   - model unpredictability
   - in WB impl. as pseudorandom
2. Goal:
   any $f \in \mathrm{span}\{v_i\}$ is unpredictable

# Algebraic Security (1/2)

Security Model:

1. random bits allowed
   - as in classic masking
   - model unpredictability
   - in WB impl. as pseudorandom
2. Goal:
   any $f \in \mathrm{span}\{v_i\}$ is unpredictable
3. isolated from obfuscation problems

# Algebraic Security (2/2)

Adversary:

1. chooses plaintext/key pairs

# Algebraic Security (2/2)

Adversary:

1. chooses plaintext/key pairs
2. chooses $f \in \mathrm{span}\{v_i\}$

# Algebraic Security (2/2)

Adversary:

1. chooses plaintext/key pairs
2. chooses $f \in \mathrm{span}\{v_i\}$
3. tries to predict values of this function (i.e. before random bits are sampled)

# Algebraic Security (2/2)

Adversary:

1. chooses plaintext/key pairs
2. chooses $f \in span\{v_i\}$
3. tries to predict values of this function (i.e. before random bits are sampled)
4. succeeds, if only $f$ matches

# Algebraic Security (3/3)

## Proposition

Let $F = \{f(x;\cdot;\cdot) \mid f(x;r_e;r_c) \in span\{v_i\};\ x \in \mathbb{F}_2^N\}.$
Let $\varepsilon = \max_{f \in F} bias(f);\ e = -\log_2(1/2 + \varepsilon)$.
Then for any adversary $A$ choosing $Q$ inputs

$$\mathrm{Adv}[A] \leq min(2^{Q-|r_c|}; 2^{-eQ}).$$

# Algebraic Security (3/3)

## Proposition

Let $F = \{f(x;\cdot;\cdot) \mid f(x;r_e;r_c) \in span\{v_i\}, x \in \mathbb{F}_2^N\}.$
Let $\varepsilon = \max_{f \in F} bias(f), e = -\log_2(1/2 + \varepsilon)$.
Then for any adversary $A$ choosing $Q$ inputs

$$\mathrm{Adv}[A] \leq min(2^{Q-|r_c|}, 2^{-eQ}).$$

## Corollary

Let $k$ be a positive integer. Then for any adversary $A$

$$\mathrm{Adv}[A] \leq 2^{-k} \text{ if } e > 0 \text{ and } |r_c| \geq k \cdot (1 + \frac{1}{e}).$$

# Algebraic Security (3/3)

**Proposition**

Let $F = \{f(x,\cdot,\cdot) \mid f(x,r_e,r_c) \in \mathit{span}\{v_i\},\ x \in \mathbb{F}_2^N\}$.
Let $\varepsilon = \max_{f \in F} \mathit{bias}(f), e = -\log_2(1/2+\varepsilon)$.
Then for any adversary $A$ choosing $Q$ inputs

$$\mathrm{Adv}[A] \leq \min(2^{Q-|r_c|}, 2^{-eQ}).$$

**Corollary**

Let $k$ be a positive integer. Then for any adversary $A$

$$\mathrm{Adv}[A] \leq 2^{-k} \text{ if } e > 0 \text{ and } |r_c| \geq k \cdot (1 + \frac{1}{e}).$$

Information-theoretic security

# Minimalist Quadratic Masking Scheme (MQMS)

Masking scheme:

- set of gadgets
- provably secure composition

function Decode$(a; b; c)$
  return $ab \quad c$

function EvalXOR$((a; b; c); (d; e; f); (r_a; r_b; r_c); (r_d; r_e; r_f))$
  $(a; b; c) \quad$ Refresh$((a; b; c); (r_a; r_b; r_c))$
  $(d; e; f) \quad$ Refresh$((d; e; f); (r_d; r_e; r_f))$
  $x \quad a \quad d$
  $y \quad b \quad e$
  $z \quad c \quad f \quad ae \quad bd$
  return $(x; y; z)$

function EvalAND$((a; b; c); (d; e; f); (r_a; r_b; r_c); (r_d; r_e; r_f))$
  $(a; b; c) \quad$ Refresh$((a; b; c); (r_a; r_b; r_c))$
  $(d; e; f) \quad$ Refresh$((d; e; f); (r_d; r_e; r_f))$
  $m_a \quad bf \quad r_c e$
  $m_d \quad ce \quad r_f b$
  $x \quad ae \quad r_f$
  $y \quad bd \quad r_c$
  $z \quad am_a \quad dm_d \quad r_c r_f \quad cf$
  return $(x; y; z)$

function Refresh$((a; b; c); (r_a; r_b; r_c))$
  $m_a \quad r_a \cdot (b \quad r_c)$
  $m_b \quad r_b \cdot (a \quad r_c)$
  $r_c \quad m_a \quad m_b \quad (r_a \quad r_c)(r_b \quad r_c) \quad r_c$
  $a \quad a \quad r_a$
  $b \quad b \quad r_b$
  $c \quad c \quad r_c$
  return $(a; b; c)$

# Minimalist Quadratic Masking Scheme (MQMS)

Masking scheme:

- set of gadgets
- provably secure composition
- quadratic decoder: $(a; b; c) \; 7\!\! ! \;\; ab \quad c$

```
function Decode(a; b; c)
    return ab    c

function EvalXOR((a; b; c); (d; e; f); (r_a; r_b; r_c); (r_d; r_e; r_f))
    (a; b; c)     Refresh((a; b; c); (r_a; r_b; r_c))
    (d; e; f)     Refresh((d; e; f); (r_d; r_e; r_f))
    x     a     d
    y     b     e
    z     c     f     ae     bd
    return (x; y; z)

function EvalAND((a; b; c); (d; e; f); (r_a; r_b; r_c); (r_d; r_e; r_f))
    (a; b; c)     Refresh((a; b; c); (r_a; r_b; r_c))
    (d; e; f)     Refresh((d; e; f); (r_d; r_e; r_f))
    m_a     bf     r_c e
    m_d     ce     r_f b
    x     ae     r_f
    y     bd     r_c
    z     am_a     dm_d     r_c r_f     cf
    return (x; y; z)

function Refresh((a; b; c); (r_a; r_b; r_c))
    m_a     r_a · (b     r_c)
    m_b     r_b · (a     r_c)
    r_c     m_a     m_b     (r_a     r_c)(r_b     r_c)     r_c
    a     a     r_a
    b     b     r_b
    c     c     r_c
    return (a; b; c)
```

# Minimalist Quadratic Masking Scheme (MQMS)

Masking scheme:

- set of gadgets
- provably secure composition
- quadratic decoder: $(a; b; c)$ 7! $ab \quad c$
- first-order protection

function Decode$(a; b; c)$
    return $ab \quad c$

function EvalXOR$((a; b; c); (d; e; f); (r_a; r_b; r_c); (r_d; r_e; r_f))$
    $(a; b; c)$ Refresh$((a; b; c); (r_a; r_b; r_c))$
    $(d; e; f)$ Refresh$((d; e; f); (r_d; r_e; r_f))$
    $x \quad a \quad d$
    $y \quad b \quad e$
    $z \quad c \quad f \quad ae \quad bd$
    return $(x; y; z)$

function EvalAND$((a; b; c); (d; e; f); (r_a; r_b; r_c); (r_d; r_e; r_f))$
    $(a; b; c)$ Refresh$((a; b; c); (r_a; r_b; r_c))$
    $(d; e; f)$ Refresh$((d; e; f); (r_d; r_e; r_f))$
    $m_a \quad bf \quad r_c e$
    $m_d \quad ce \quad r_f b$
    $x \quad ae \quad r_f$
    $y \quad bd \quad r_c$
    $z \quad am_a \quad dm_d \quad r_c r_f \quad cf$
    return $(x; y; z)$

function Refresh$((a; b; c); (r_a; r_b; r_c))$
    $m_a \quad r_a \cdot (b \quad r_c)$
    $m_b \quad r_b \cdot (a \quad r_c)$
    $r_c \quad m_a \quad m_b \quad (r_a \quad r_c)(r_b \quad r_c) \quad r_c$
    $a \quad a \quad r_a$
    $b \quad b \quad r_b$
    $c \quad c \quad r_c$
    return $(a; b; c)$

# MQMS Security

Security:

1. algorithm to verify that bias ≠ 1=2
2. max. degree on $r$: 4

```
function Decode(a; b; c)
    return ab    c

function EvalXOR((a; b; c); (d; e; f); (r_a; r_b; r_c); (r_d; r_e; r_f))
    (a; b; c)    Refresh((a; b; c); (r_a; r_b; r_c))
    (d; e; f)    Refresh((d; e; f); (r_d; r_e; r_f))
    x    a    d
    y    b    e
    z    c    f    ae    bd
    return (x; y; z)

function EvalAND((a; b; c); (d; e; f); (r_a; r_b; r_c); (r_d; r_e; r_f))
    (a; b; c)    Refresh((a; b; c); (r_a; r_b; r_c))
    (d; e; f)    Refresh((d; e; f); (r_d; r_e; r_f))
    m_a    bf    r_c e
    m_d    ce    r_f b
    x    ae    r_f
    y    bd    r_c
    z    am_a    dm_d    r_c r_f    cf
    return (x; y; z)

function Refresh((a; b; c); (r_a; r_b; r_c))
    m_a    r_a · (b    r_c)
    m_b    r_b · (a    r_c)
    r_c    m_a    m_b    (r_a    r_c)(r_b    r_c)    r_c
    a    a    r_a
    b    b    r_b
    c    c    r_c
    return (a; b; c)
```

# MQMS Security

Security:

1. algorithm to verify that bias 6= 1=2
2. max. degree on $r$: 4

) bias ≤ 7=16

for 80-bit security we need $jr_cj$ ≥ 940

```
function Decode(a; b; c)
    return ab    c

function EvalXOR((a; b; c); (d; e; f); (r_a; r_b; r_c); (r_d; r_e; r_f))
    (a; b; c)    Refresh((a; b; c); (r_a; r_b; r_c))
    (d; e; f)    Refresh((d; e; f); (r_d; r_e; r_f))
    x    a    d
    y    b    e
    z    c    f    ae    bd
    return (x; y; z)

function EvalAND((a; b; c); (d; e; f); (r_a; r_b; r_c); (r_d; r_e; r_f))
    (a; b; c)    Refresh((a; b; c); (r_a; r_b; r_c))
    (d; e; f)    Refresh((d; e; f); (r_d; r_e; r_f))
    m_a    bf    r_c e
    m_d    ce    r_f b
    x    ae    r_f
    y    bd    r_c
    z    am_a    dm_d    r_c r_f    cf
    return (x; y; z)

function Refresh((a; b; c); (r_a; r_b; r_c))
    m_a    r_a · (b    r_c)
    m_b    r_b · (a    r_c)
    r_c    m_a    m_b    (r_a    r_c)(r_b    r_c)    r_c
    a    a    r_a
    b    b    r_b
    c    c    r_c
    return (a; b; c)
```

# Implementation

Proof-of-concept masked AES-128

1. MQMS + 1-st order Boolean masking
2. 31,783 ! 2,588,743 gates expansion (x81)
3. 16 Mb code / 1 Kb RAM / 0.05s per block on a laptop
4. (unoptimized)

github.com/cryptolu/whitebox

# Conclusions

Conclusions:

1. new attack methods ) new constraints on a white-box impl.
2. new results on provable security for white-box model
3. new links with side-channel research

# Conclusions

## Conclusions:

1. new attack methods ) new constraints on a white-box impl.
2. new results on provable security for white-box model
3. new links with side-channel research

## Open problems and future work:

1. structure-hiding component
2. higher-order protection
3. analysis of LPN-based attacks
4. deeper study of the fault attacks
5. optimizations

# The End

ePrint 2018/049

github.com/cryptolu/whitebox

Thank you!