

Attacks and Countermeasures for White-box Designs

Alex Biryukov, Aleksei Udovenko

CSC and SnT, University of Luxembourg

December 5, 2018



Plan

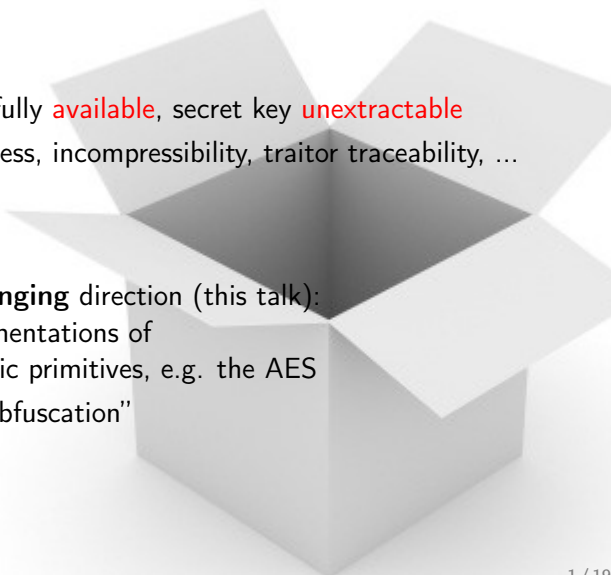
- 1 Introduction
- 2 Attacks on Masked White-box Implementations
- 3 Countermeasures
- 4 Algebraic Security

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

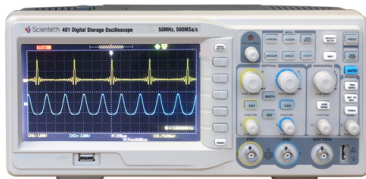


- many applications
- strong need for *practical* white-box
- industry **does** WB:
hidden designs



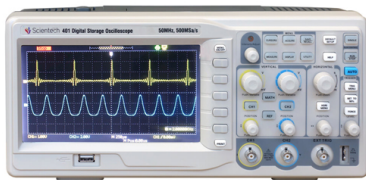
- **theory**: approaches using iO/FE, currently *impractical*
- **practical WB-AES**: few attempts (2002-2017), **all broken**
- powerful DCA attack (CHES 2016)

White-Box: Differential Computation Analysis (DCA)



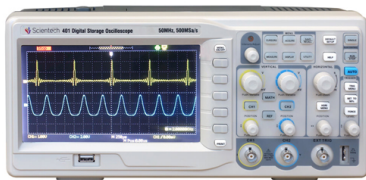
- **DCA = Differential Power Analysis (DPA)**
applied to white-box implementations
- Most of the implementations **broken automatically**

White-Box: Differential Computation Analysis (DCA)



- **DCA = Differential 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 = Differential 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.

$$s = \text{Bit}_1(\text{SBox}(pt_1 \oplus k_1))$$

General Setting

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

$$s = \text{Bit}_1(\text{SBox}(pt_1 \oplus k_1))$$

- **Masking**: $\exists v_1, \dots, v_t$ nodes (*shares*), $f : \mathbb{F}_2^t \rightarrow \mathbb{F}_2$ 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)

⇒ can be secure only if the locations of the shares in the circuit are unknown!

this talk: exploring this possibility

Plan

- 1 Introduction
- 2 Attacks on Masked White-box Implementations**
- 3 Countermeasures
- 4 Algebraic Security

Combinatorial attacks:

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

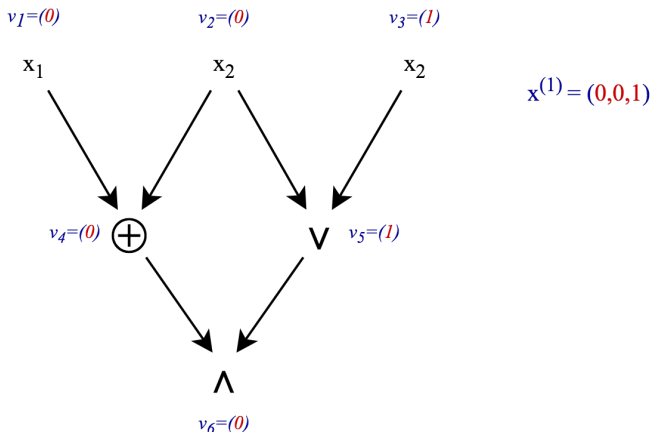
Combinatorial attacks:

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

Fault attacks:

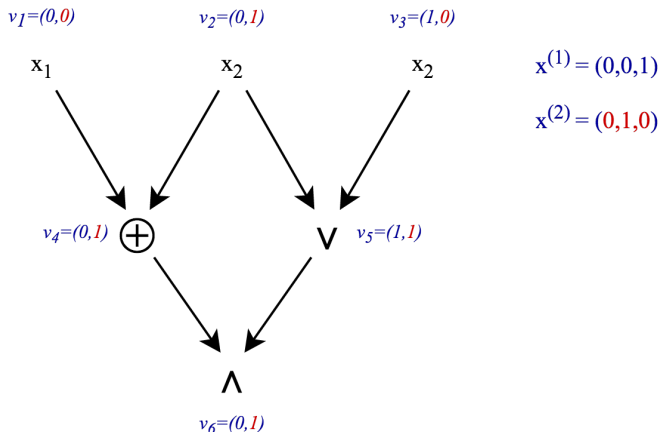
- **new application**: **recover locations** of the shares
- 1- and 2- share fault injections
- applicability depends on protections

(Generalized) Differential Computation Analysis (DCA):

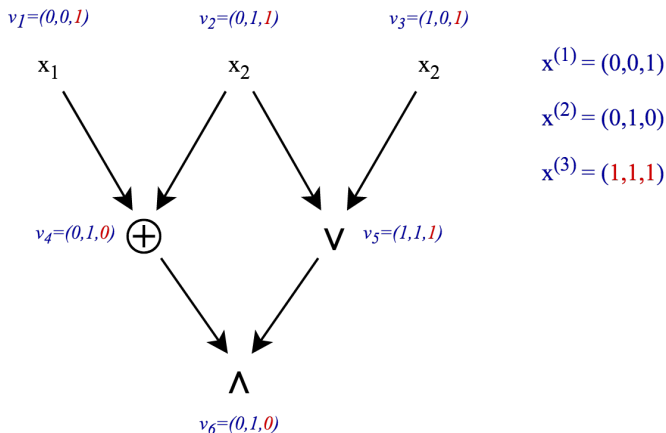


Attacks II

(Generalized) Differential Computation Analysis (DCA):



(Generalized) Differential 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:

$$M \times z = s, \quad M = [v_1 \mid \dots \mid v_n]$$

The Linear Algebra Attack (1)

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

$$M \times z = s, \quad M = [v_1 \mid \dots \mid v_n]$$

- 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 assume $s \cdot r$ is computed/shared in the circuit, where
 - 2 s is a **predictable** value
 - 3 r is **unpredictable** (pseudorandom, \approx uniform)

The Linear Algebra Attack (2)

Generalizations:

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

Plan

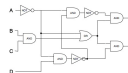
- 1 Introduction
- 2 Attacks on Masked White-box Implementations
- 3 Countermeasures**
- 4 Algebraic Security

Our Framework: Two Components

Value Hiding

```
00101010111010010010101011101001
10101010100101010001001010101011
...
10000010011000000010101010001001
01100001110000010010101011101110
```

Structure Hiding

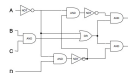


Our Framework: Two Components

Value Hiding

```
00101010111010010010101011101001
10101010100101010001001010101011
...
10000010011000000010101010001001
01100001110000010010101011101110
```

Structure Hiding



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

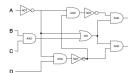
Our Framework: Two Components

Value Hiding

```
001010101110100100101011101001
10101010100101010001001010101011
...
10000010011000000010101010001001
01100001110000010010101011101110
```

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

Structure Hiding



- 1 circuit analysis / simplification
- 2 fault injections
- 3 pseudorandomness removal
- 4 etc.

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

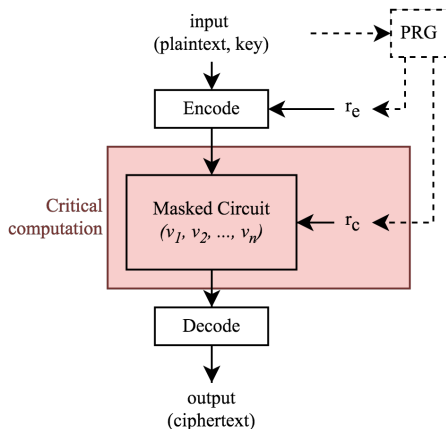
- 1 Introduction
- 2 Attacks on Masked White-box Implementations
- 3 Countermeasures
- 4 Algebraic Security**

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)

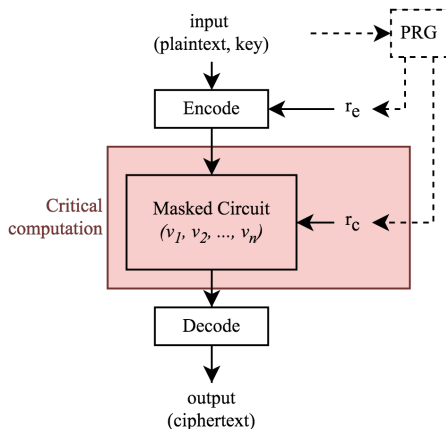
Security Model:

1 random bits allowed

- as in classic masking
- model **unpredictability**
- in WB impl. as **pseudorandom**

2 Goal:

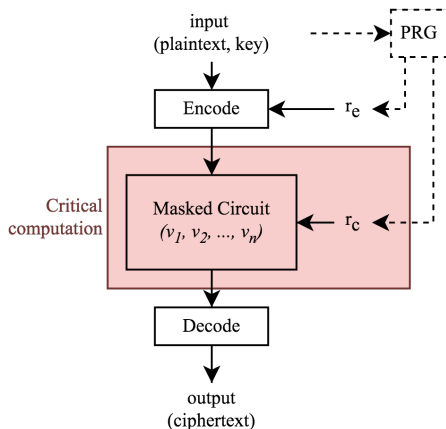
any $f \in \text{span}\{v_i\}$ is **unpredictable**



Algebraic Security (1/2)

Security Model:

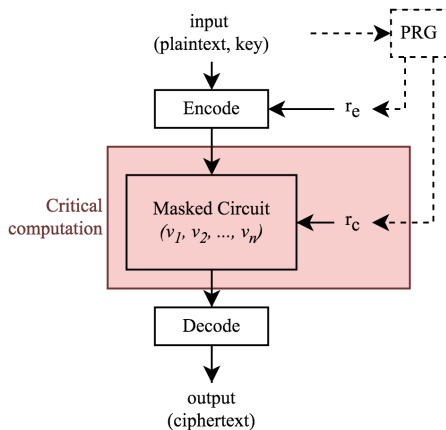
- 1 **random** bits allowed
 - as in classic masking
 - model **unpredictability**
 - in WB impl. as **pseudorandom**
- 2 **Goal:**
any $f \in \text{span}\{v_i\}$ is **unpredictable**
- 3 **isolated** from obfuscation problems



Algebraic Security (2/2)

Adversary:

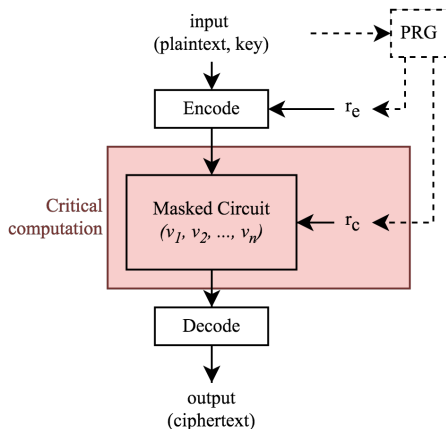
- 1 chooses plaintext/key pairs



Algebraic Security (2/2)

Adversary:

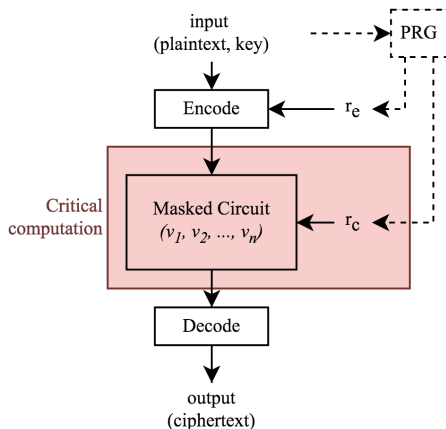
- 1 chooses plaintext/key pairs
- 2 chooses $f \in \text{span}\{v_i\}$



Algebraic Security (2/2)

Adversary:

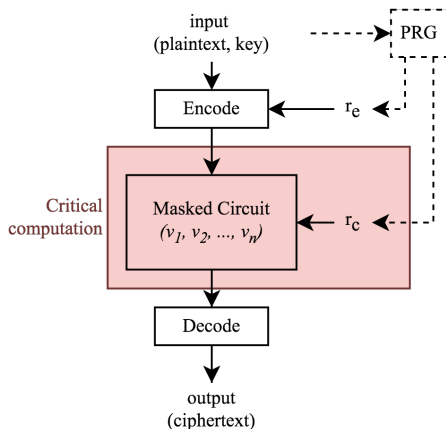
- 1 chooses plaintext/key pairs
- 2 chooses $f \in \text{span}\{v_i\}$
- 3 tries to **predict** values of this function
(i.e. before random bits are sampled)



Algebraic Security (2/2)

Adversary:

- 1 chooses plaintext/key pairs
- 2 chooses $f \in \text{span}\{v_i\}$
- 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 $F = \{f(x, \cdot, \cdot) \mid f(x, r_e, r_c) \in \text{span}\{v_i\}, x \in \mathbb{F}_2^N\}$.

Let $\varepsilon = \max_{f \in F} \text{bias}(f)$, $e = -\log_2(1/2 + \varepsilon)$.

Then for any adversary \mathcal{A} choosing Q inputs

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

Algebraic Security (3/3)

Proposition

Let $F = \{f(x, \cdot, \cdot) \mid f(x, r_e, r_c) \in \text{span}\{v_i\}, x \in \mathbb{F}_2^N\}$.

Let $\varepsilon = \max_{f \in F} \text{bias}(f)$, $e = -\log_2(1/2 + \varepsilon)$.

Then for any adversary \mathcal{A} choosing Q inputs

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

Corollary

Let k be a positive integer. Then for any adversary \mathcal{A}

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

Algebraic Security (3/3)

Proposition

Let $F = \{f(x, \cdot, \cdot) \mid f(x, r_e, r_c) \in \text{span}\{v_i\}, x \in \mathbb{F}_2^N\}$.

Let $\varepsilon = \max_{f \in F} \text{bias}(f)$, $e = -\log_2(1/2 + \varepsilon)$.

Then for any adversary \mathcal{A} choosing Q inputs

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

Corollary

Let k be a positive integer. Then for any adversary \mathcal{A}

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

Information-theoretic security

Minimalist Quadratic Masking Scheme (MQMS)

Masking scheme:

- set of gadgets
- provably secure composition

```
function Decode( $a, b, c$ )  
    return  $ab \oplus c$ 
```

```
function EvalXOR( $(a, b, c), (d, e, f), (r_a, r_b, r_c), (r_d, r_e, r_f)$ )  
     $(a, b, c) \leftarrow \text{Refresh}((a, b, c), (r_a, r_b, r_c))$   
     $(d, e, f) \leftarrow \text{Refresh}((d, e, f), (r_d, r_e, r_f))$   
     $x \leftarrow a \oplus d$   
     $y \leftarrow b \oplus e$   
     $z \leftarrow c \oplus f \oplus ae \oplus bd$   
    return  $(x, y, z)$ 
```

```
function EvalAND( $(a, b, c), (d, e, f), (r_a, r_b, r_c), (r_d, r_e, r_f)$ )  
     $(a, b, c) \leftarrow \text{Refresh}((a, b, c), (r_a, r_b, r_c))$   
     $(d, e, f) \leftarrow \text{Refresh}((d, e, f), (r_d, r_e, r_f))$   
     $m_a \leftarrow bf \oplus r_c e$   
     $m_d \leftarrow ce \oplus r_f b$   
     $x \leftarrow ae \oplus r_f$   
     $y \leftarrow bd \oplus r_c$   
     $z \leftarrow am_a \oplus dm_d \oplus r_c r_f \oplus cf$   
    return  $(x, y, z)$ 
```

```
function Refresh( $(a, b, c), (r_a, r_b, r_c)$ )  
     $m_a \leftarrow r_a \cdot (b \oplus r_c)$   
     $m_b \leftarrow r_b \cdot (a \oplus r_c)$   
     $r_c \leftarrow m_a \oplus m_b \oplus (r_a \oplus r_c)(r_b \oplus r_c) \oplus r_c$   
     $a \leftarrow a \oplus r_a$   
     $b \leftarrow b \oplus r_b$   
     $c \leftarrow c \oplus r_c$   
    return  $(a, b, c)$ 
```

Minimalist Quadratic Masking Scheme (MQMS)

Masking scheme:

- set of gadgets
- provably secure composition
- **quadratic** decoder:
 $(a, b, c) \mapsto ab \oplus c$

```
function Decode(a, b, c)
    return  $ab \oplus c$ 
```

```
function EvalXOR((a, b, c), (d, e, f), (ra, rb, rc), (rd, re, rf))
    (a, b, c) ← Refresh((a, b, c), (ra, rb, rc))
    (d, e, f) ← Refresh((d, e, f), (rd, re, rf))
    x ←  $a \oplus d$ 
    y ←  $b \oplus e$ 
    z ←  $c \oplus f \oplus ae \oplus bd$ 
    return (x, y, z)
```

```
function EvalAND((a, b, c), (d, e, f), (ra, rb, rc), (rd, re, rf))
    (a, b, c) ← Refresh((a, b, c), (ra, rb, rc))
    (d, e, f) ← Refresh((d, e, f), (rd, re, rf))
    ma ←  $bf \oplus r_c e$ 
    md ←  $ce \oplus r_f b$ 
    x ←  $ae \oplus r_f$ 
    y ←  $bd \oplus r_c$ 
    z ←  $am_a \oplus dm_d \oplus r_c r_f \oplus cf$ 
    return (x, y, z)
```

```
function Refresh((a, b, c), (ra, rb, rc))
    ma ←  $r_a \cdot (b \oplus r_c)$ 
    mb ←  $r_b \cdot (a \oplus r_c)$ 
    rc ←  $m_a \oplus m_b \oplus (r_a \oplus r_c)(r_b \oplus r_c) \oplus r_c$ 
    a ←  $a \oplus r_a$ 
    b ←  $b \oplus r_b$ 
    c ←  $c \oplus r_c$ 
    return (a, b, c)
```

Minimalist Quadratic Masking Scheme (MQMS)

Masking scheme:

- set of gadgets
- provably secure composition
- **quadratic** decoder:
 $(a, b, c) \mapsto ab \oplus c$
- first-order protection

```
function Decode(a, b, c)
    return  $ab \oplus c$ 
```

```
function EvalXOR((a, b, c), (d, e, f), (ra, rb, rc), (rd, re, rf))
    (a, b, c) ← Refresh((a, b, c), (ra, rb, rc))
    (d, e, f) ← Refresh((d, e, f), (rd, re, rf))
    x ←  $a \oplus d$ 
    y ←  $b \oplus e$ 
    z ←  $c \oplus f \oplus ae \oplus bd$ 
    return (x, y, z)
```

```
function EvalAND((a, b, c), (d, e, f), (ra, rb, rc), (rd, re, rf))
    (a, b, c) ← Refresh((a, b, c), (ra, rb, rc))
    (d, e, f) ← Refresh((d, e, f), (rd, re, rf))
    ma ←  $bf \oplus r_c e$ 
    md ←  $ce \oplus r_f b$ 
    x ←  $ae \oplus r_f$ 
    y ←  $bd \oplus r_c$ 
    z ←  $am_a \oplus dm_d \oplus r_c r_f \oplus cf$ 
    return (x, y, z)
```

```
function Refresh((a, b, c), (ra, rb, rc))
    ma ←  $r_a \cdot (b \oplus r_c)$ 
    mb ←  $r_b \cdot (a \oplus r_c)$ 
    rc ←  $m_a \oplus m_b \oplus (r_a \oplus r_c)(r_b \oplus r_c) \oplus r_c$ 
    a ←  $a \oplus r_a$ 
    b ←  $b \oplus r_b$ 
    c ←  $c \oplus r_c$ 
    return (a, b, c)
```

Security:

- 1 algorithm to verify that bias $\neq 1/2$
- 2 max. degree on r : 4

function Decode(a, b, c)

return $ab \oplus c$

function EvalXOR($(a, b, c), (d, e, f), (r_a, r_b, r_c), (r_d, r_e, r_f)$)

$(a, b, c) \leftarrow \text{Refresh}((a, b, c), (r_a, r_b, r_c))$

$(d, e, f) \leftarrow \text{Refresh}((d, e, f), (r_d, r_e, r_f))$

$x \leftarrow a \oplus d$

$y \leftarrow b \oplus e$

$z \leftarrow c \oplus f \oplus ae \oplus bd$

return (x, y, z)

function EvalAND($(a, b, c), (d, e, f), (r_a, r_b, r_c), (r_d, r_e, r_f)$)

$(a, b, c) \leftarrow \text{Refresh}((a, b, c), (r_a, r_b, r_c))$

$(d, e, f) \leftarrow \text{Refresh}((d, e, f), (r_d, r_e, r_f))$

$m_a \leftarrow bf \oplus r_c e$

$m_d \leftarrow ce \oplus r_f b$

$x \leftarrow ae \oplus r_f$

$y \leftarrow bd \oplus r_c$

$z \leftarrow am_a \oplus dm_d \oplus r_c r_f \oplus cf$

return (x, y, z)

function Refresh($(a, b, c), (r_a, r_b, r_c)$)

$m_a \leftarrow r_a \cdot (b \oplus r_c)$

$m_b \leftarrow r_b \cdot (a \oplus r_c)$

$r_c \leftarrow m_a \oplus m_b \oplus (r_a \oplus r_c)(r_b \oplus r_c) \oplus r_c$

$a \leftarrow a \oplus r_a$

$b \leftarrow b \oplus r_b$

$c \leftarrow c \oplus r_c$

return (a, b, c)

MQMS Security

Security:

- 1 algorithm to verify that bias $\neq 1/2$
- 2 max. degree on r : 4

\Rightarrow bias $\leq 7/16$

for 80-bit security
we need $|r_c| \geq 940$

```
function Decode(a, b, c)
  return  $ab \oplus c$ 
```

```
function EvalXOR((a, b, c), (d, e, f), (ra, rb, rc), (rd, re, rf))
  (a, b, c)  $\leftarrow$  Refresh((a, b, c), (ra, rb, rc))
  (d, e, f)  $\leftarrow$  Refresh((d, e, f), (rd, re, rf))
  x  $\leftarrow a \oplus d$ 
  y  $\leftarrow b \oplus e$ 
  z  $\leftarrow c \oplus f \oplus ae \oplus bd$ 
  return (x, y, z)
```

```
function EvalAND((a, b, c), (d, e, f), (ra, rb, rc), (rd, re, rf))
  (a, b, c)  $\leftarrow$  Refresh((a, b, c), (ra, rb, rc))
  (d, e, f)  $\leftarrow$  Refresh((d, e, f), (rd, re, rf))
  ma  $\leftarrow bf \oplus r_c e$ 
  md  $\leftarrow ce \oplus r_f b$ 
  x  $\leftarrow ae \oplus r_f$ 
  y  $\leftarrow bd \oplus r_c$ 
  z  $\leftarrow am_a \oplus dm_d \oplus r_c r_f \oplus cf$ 
  return (x, y, z)
```

```
function Refresh((a, b, c), (ra, rb, rc))
  ma  $\leftarrow r_a \cdot (b \oplus r_c)$ 
  mb  $\leftarrow r_b \cdot (a \oplus r_c)$ 
  rc  $\leftarrow m_a \oplus m_b \oplus (r_a \oplus r_c)(r_b \oplus r_c) \oplus r_c$ 
  a  $\leftarrow a \oplus r_a$ 
  b  $\leftarrow b \oplus r_b$ 
  c  $\leftarrow c \oplus r_c$ 
  return (a, b, c)
```

Proof-of-concept masked AES-128

- 1 MQMS + 1-st order Boolean masking
- 2 31,783 \rightarrow 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 \Rightarrow 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 \Rightarrow 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!

