Defeating State-of-the-Art White-Box Countermeasures with Advanced Gray-Box Attacks

Louis Goubin[4] **Matthieu Rivain**[1] **Junwei Wang (王军委)**[1,2,3]

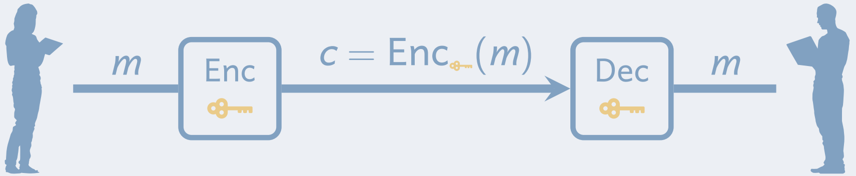
[1]CryptoExperts [2]University of Luxembourg [3]University Paris 8 [4]UVSQ

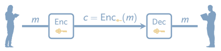
Prerecorded talk for **CHES 2020**, September 2020

Defeating State-of-the-Art White-Box Countermeasures with Advanced Gray-Box Attacks

- Hello everyone, I am Junwei Wang.
- This is a pre-recorded video for CHES 2020.
- The work entitled “defeating state-of-the-art white-box countermeasures with advanced gray-box attacks”, is a collaboration works with Louis Goubin and Matthieu Rivain.
- This work was done when I was a Ph.D. student at University of Luxembourg and University Paris 8.

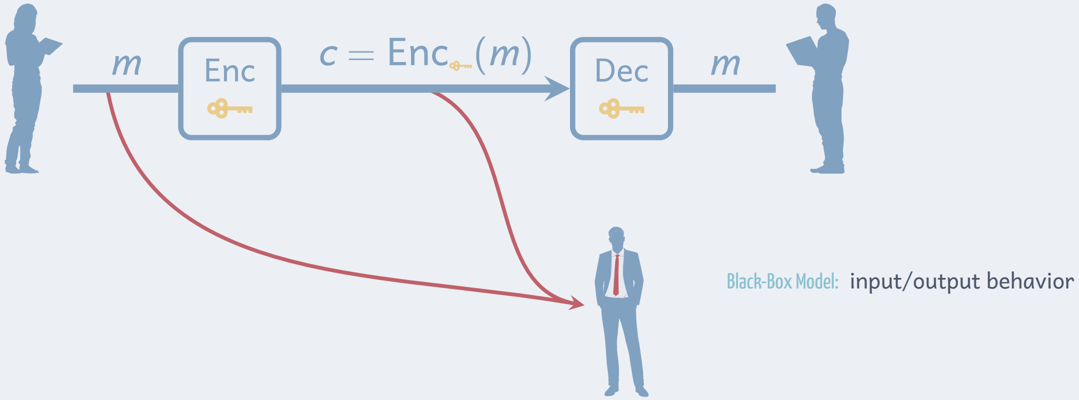
» Security Models: Shades of Gray

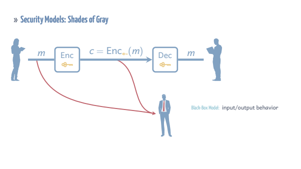




- We consider two parties communicate over an insecure channel with a pre-shared secret key.
- The classical cryptanalysis is done in black-box model in the sense that the adversary only has access to encryption/decryption algorithm as black-box, that is, the adversary can only learn the the input and output behavior of the cipher.
- A cipher has to be implemented in software or hardware to be useful.
- When a cipher is implemented in hardware, then adversary is further able to access to the physical information of the execution of the cipher, such as the power consumption. This security model is called gray-box model.
- In the extreme case, the adversary can fully control an implementation of some cipher and its execution environment. In this model, the capabilities of the adversary are greatly enhanced. We call this security model a white-box model.

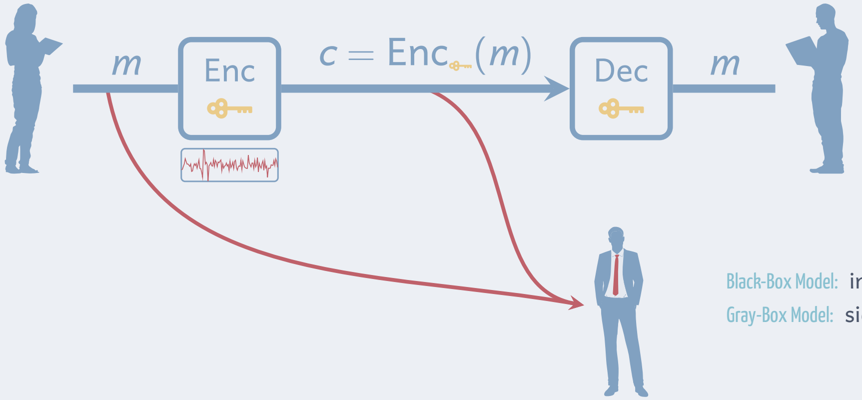
» Security Models: Shades of Gray





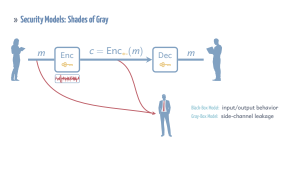
- We consider two parties communicate over an insecure channel with a pre-shared secret key.
- The classical cryptanalysis is done in black-box model in the sense that the adversary only has access to encryption/decryption algorithm as black-box, that is, the adversary can only learn the the input and output behavior of the cipher.
- A cipher has to be implemented in software or hardware to be useful.
- When a cipher is implemented in hardware, then adversary is further able to access to the physical information of the execution of the cipher, such as the power consumption. This security model is called gray-box model.
- In the extreme case, the adversary can fully control an implementation of some cipher and its execution environment. In this model, the capabilities of the adversary are greatly enhanced. We call this security model a white-box model.

» Security Models: Shades of Gray



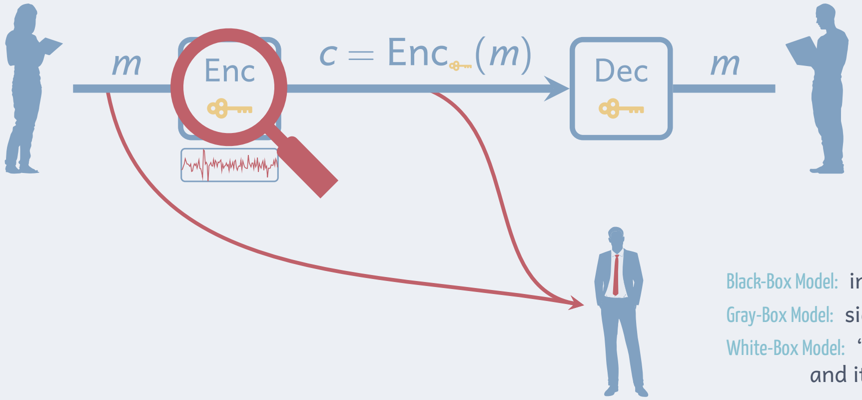
Black-Box Model: input/output behavior

Gray-Box Model: side-channel leakage



- We consider two parties communicate over an insecure channel with a pre-shared secret key.
- The classical cryptanalysis is done in black-box model in the sense that the adversary only has access to encryption/decryption algorithm as black-box, that is, the adversary can only learn the the input and output behavior of the cipher.
- A cipher has to be implemented in software or hardware to be useful.
- When a cipher is implemented in hardware, then adversary is further able to access to the physical information of the execution of the cipher, such as the power consumption. This security model is called gray-box model.
- In the extreme case, the adversary can fully control an implementation of some cipher and its execution environment. In this model, the capabilities of the adversary are greatly enhanced. We call this security model a white-box model.

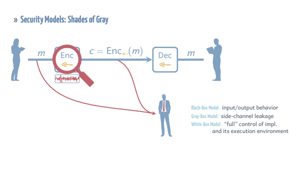
» Security Models: Shades of Gray



Black-Box Model: input/output behavior

Gray-Box Model: side-channel leakage

White-Box Model: "full" control of impl. and its execution environment

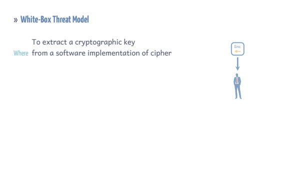


- We consider two parties communicate over an insecure channel with a pre-shared secret key.
- The classical cryptanalysis is done in black-box model in the sense that the adversary only has access to encryption/decryption algorithm as black-box, that is, the adversary can only learn the the input and output behavior of the cipher.
- A cipher has to be implemented in software or hardware to be useful.
- When a cipher is implemented in hardware, then adversary is further able to access to the physical information of the execution of the cipher, such as the power consumption. This security model is called gray-box model.
- In the extreme case, the adversary can fully control an implementation of some cipher and its execution environment. In this model, the capabilities of the adversary are greatly enhanced. We call this security model a white-box model.

» White-Box Threat Model

To extract a cryptographic key

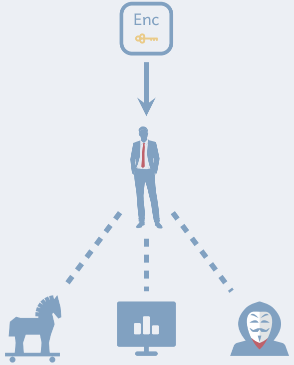
Where from a software implementation of cipher

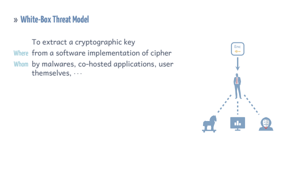


- In the white-box model, the adversary tries to extract the underlying secret key from a software implementation.
- The adversary could represent malwares, malicious applications in the same host and even the application user themselves.
- The adversary could try all possible ways, for instance,
  - she could perform static/dynamic analysis of the code
  - she could spy on the memory while executing the code,
  - she can even modify the code, inject faults and employ the erroneous executions
  - she can easily reset the external randomness

» White-Box Threat Model

	To extract a cryptographic key
Where	from a software implementation of cipher
Whom	by malwares, co-hosted applications, user themselves, · · ·





- In the white-box model, the adversary tries to extract the underlying secret key from a software implementation.
- The adversary could represent malwares, malicious applications in the same host and even the application user themselves.
- The adversary could try all possible ways, for instance,
  - she could perform static/dynamic analysis of the code
  - she could spy on the memory while executing the code,
  - she can even modify the code, inject faults and employ the erroneous executions
  - she can easily reset the external randomness

» White-Box Threat Model

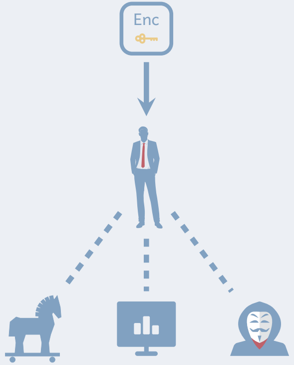
To extract a cryptographic key

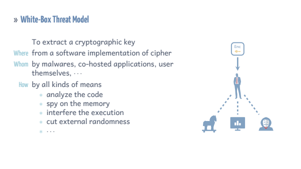
Where from a software implementation of cipher

Whom by malwares, co-hosted applications, user themselves, · · ·

How by all kinds of means

- analyze the code
- spy on the memory
- interfere the execution
- cut external randomness
- · · ·

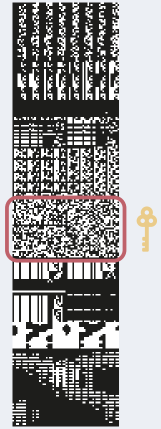




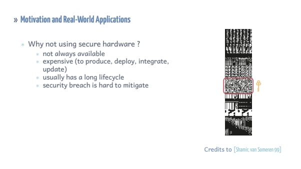
- In the white-box model, the adversary tries to extract the underlying secret key from a software implementation.
- The adversary could represent malwares, malicious applications in the same host and even the application user themselves.
- The adversary could try all possible ways, for instance,
  - she could perform static/dynamic analysis of the code
  - she could spy on the memory while executing the code,
  - she can even modify the code, inject faults and employ the erroneous executions
  - she can easily reset the external randomness

» Motivation and Real-World Applications

- Why not using secure hardware ?
  - not always available
  - expensive (to produce, deploy, integrate, update)
  - usually has a long lifecycle
  - security breach is hard to mitigate



Credits to [Shamir, van Someren 99]

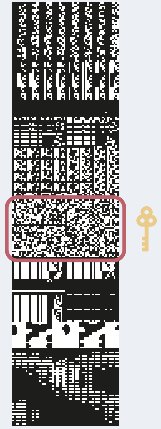


- Traditionally, secure hardware is used to protect secret keys in ciphers.
- However, such a hardware are not always available.
- Besides, it is costly to maintain the long lifecycle of a secure hardware.
- Most importantly, it is not easy to mitigate the security breach of hardware
- In many real-world application, cryptographic algorithm has to be deployed in pure software, such context including
  - Digital content distribution, mobile payment, digital contract signing and blockchain technologies.
- As illustrated by Shamir and van Someren, if the key is not protected, it is trivial to extract the key from the memory since the key looks random.
- Hence, in these contexts, white-box cryptography is an essential component to maintain the security of the system.

» Motivation and Real-World Applications

- Why not using secure hardware ?
  - not always available
  - expensive (to produce, deploy, integrate, update)
  - usually has a long lifecycle
  - security breach is hard to mitigate

- Applications
  - Digital Content Distribution
  - Mobile Payment
  - Digital Contract Signing
  - Blockchains and cryptocurrencies



Credits to [Shamir, van Someren 99]

» Motivation and Real-World Applications
Why not using secure hardware ? not always available; expensive (to produce, deploy, integrate, update); usually has a long lifecycle; security breach is hard to mitigate
Applications: Digital Content Distribution; Mobile Payment; Digital Contract Signing; Blockchains and cryptocurrencies
Credits to [Shamir, van Someren 99]

- Traditionally, secure hardware is used to protect secret keys in ciphers.
- However, such a hardware are not always available.
- Besides, it is costly to maintain the long lifecycle of a secure hardware.
- Most importantly, it is not easy to mitigate the security breach of hardware

- In many real-world application, cryptographic algorithm has to be deployed in pure software, such context including
  - Digital content distribution, mobile payment, digital contract signing and blockchain technologies.
- As illustrated by Shamir and van Someren, if the key is not protected, it is trivial to extract the key from the memory since the key looks random.
- Hence, in these contexts, white-box cryptography is an essential component to maintain the security of the system.

» Security through Obscurity

- All public white-box designs broken
- No provably secure solution

- Growing demand in industry
- Huge application potential

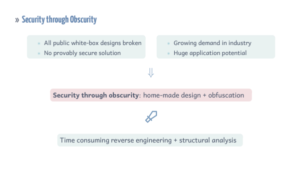
⇓

Security through obscurity: home-made design + obfuscation

Time consuming reverse engineering + structural analysis

2020-09-03

Defeating State-of-the-Art White-Box Countermeasures with Advanced Gray-Box Attacks
└White-Box Cryptography
└Security through Obscurity

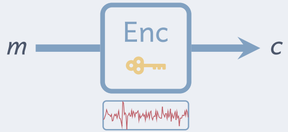


- Unfortunately, all existing public white-box designs in the literature are broken by structural analysis and we don't have any provably secure white-box for any standard cipher.
- However, the demand of white-box crypto from the industry keep growing and there is huge application potential for white-box crypto.
- This forces industry to achieve security through obscurity, namely, the use home-made design, the put a layer of obfuscation to protect their design.
- The industry hopes this would discourage the attacker by time-consuming reverse engineering and new structural analysis

» Differential Computation Analysis (DCA) [BHMT16]

Differential power analysis (DPA) techniques on computational leakages.

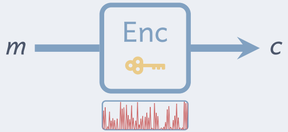
gray-box model



side-channel leakages (noisy)

e.g. power / EM / time / ···

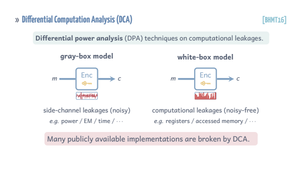
white-box model



computational leakages (noisy-free)

e.g. registers / accessed memory / ···

Many publicly available implementations are broken by DCA.



- At CHES 2016, Bos et al suggested to apply differential power analysis techniques to break white-box implementations.
- Classical DPA in gray-box model works with noisy side-channel leakage such as power consumption, electromagnetic radiation, and execution time.
- In white-box model, the attack is called differential computation analysis, since the attack works with the noisy-free computational leakages, which could be any collected running information, such as values in the registers and memory.
- DCA is a generic attack because it does not to know any implementation details.
- Surprisingly, DCA authors have shown that it is able to break many publicly available white-box implementations.
- DCA has becomes a main threat of the security thorough obscurity paradigm.

» WhibOx Competitions

- Organized as CHES CTF events

> *The competition gives an opportunity for researchers and practitioners to confront their (secretly designed) white-box implementations to state-of-the-art attackers*
> —- WhibOx 2017

- Designer: to submit the C source codes of AES-128 with secret key
- Attacker: to reveal the hidden key
- No need to disclose identity or underlying techniques

» WhibOx Competitions
Organized as CHES CTF events
The competition gives an opportunity for researchers and practitioners to confront their (secretly designed) white-box implementations to state-of-the-art attackers
—- WhibOx 2017
Designer: to submit the C source codes of AES-128 with secret key
Attacker: to reveal the hidden key
No need to disclose identity or underlying techniques

- In this context, two editions of WhibOx competitions are organized as CHES CTF events.
- As quoted from WhibOx 2017, the competition gives an opportunity for researchers and practitioners to confront their (secretly designed) white-box implementations to state-of-the-art attackers.
- The competitions invite designers to submit the C source codes of AES-128 with secretly chosen key
- and invite attacker to reveal the hidden keys
- The participants do not have to disclose their identities or designing / attacking techniques.

» WhibOx Competitions (cont.)

- WhibOx 2017
  - 94 submissions were **all broken** by 877 individual breaks
  - most (86%) of them were alive for < 1 day
  - mostly broken by DCA [BT20]

» WhibOx Competitions (cont.)
WhibOx 2017
94 submissions were all broken by 877 individual breaks
most (86%) of them were alive for < 1 day
mostly broken by DCA [BT20]

- The first competition took place in 2017.
- It attracted 94 submission which were all broken with nearly 900 individual breakes
- Most of them were broken in one day and it is reported that mostly of them can be broken simply by DCA attacks.
- The new edition took place in 2019.
- New rules are introduced to encourage designers to submit “smaller” and “faster” implementations
- Finally, 27 implementations were submitted and only 3 of them stayed alive until the end of the competition.
- They were broken soon after the competition with the techniques presented in this talk.

» WhibOx Competitions (cont.)

- WhibOx 2017
  - 94 submissions were **all broken** by 877 individual breaks
  - most (86%) of them were alive for < 1 day
  - mostly broken by DCA [BT20]
- WhibOx 2019
  - new rules encourage designers to submit “smaller” and “faster” implementations
  - 27 submissions with 124 individual breaks
  - 3 implementations survived, but broken after the competition in this article

» WhibOx Competitions (cont.)
WhibOx 2017
94 submissions were all broken by 877 individual breaks
most (86%) of them were alive for < 1 day
mostly broken by DCA [BT20]
WhibOx 2019
new rules encourage designers to submit “smaller” and “faster” implementations
27 submissions with 124 individual breaks
3 implementations survived, but broken after the competition in this article

- The first competition took place in 2017.
- It attracted 94 submission which were all broken with nearly 900 individual breakes
- Most of them were broken in one day and it is reported that mostly of them can be broken simply by DCA attacks.
- The new edition took place in 2019.
- New rules are introduced to encourage designers to submit “smaller” and “faster” implementations
- Finally, 27 implementations were submitted and only 3 of them stayed alive until the end of the competition.
- They were broken soon after the competition with the techniques presented in this talk.

» Outline

» Outline
Advanced Gray-Box Countermeasures and Attacks
Data-Dependency Analysis
Conclusion

- Since DCA is DPA techniques used in white-box context, it is natural to adopt DPA countermeasures to protect white-box implementation.
- In this presentation, I will first talk about the advanced gray-box countermeasures that are used in practical white-box implementation, as well as the three winning implementations of WhibOx 2019.
- In the same time, I will present different attacking paths against these countermeasures and analyze the performance in term of computation complexity.
- Then I will introduce a new data-dependency based attack that substantially improves the attacking complexity.
- Finally, I will conclude this talk.

Advanced Gray-Box Countermeasures and Attacks

- Linear Masking, Higher-Order DCA, and Linear Decoding Analysis
- Algebraic Security and Non-Linear Masking
- Shuffling

Advanced Gray-Box Countermeasures and Attacks
- Linear Masking, Higher-Order DCA, and Linear Decoding Analysis
- Algebraic Security and Non-Linear Masking
- Shuffling

- We start with the linear masking countermeasures and two attacks against it
- Then we talk about algebraic security and non-linear masking countermeasures
- Finally, we talk about the role of shuffling countermeasure played in a white-box implementation.

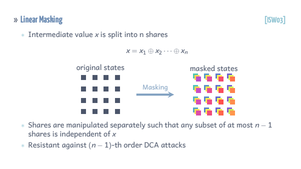
» Linear Masking

[ISW03]

- Intermediate value x is split into n shares

$$x = x_1 \oplus x_2 \cdots \oplus x_n$$

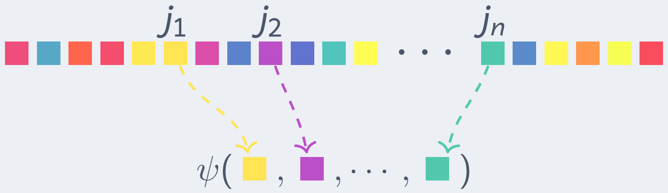
- Shares are manipulated separately such that any subset of at most $n-1$ shares is independent of x
- Resistant against $(n-1)$-th order DCA attacks



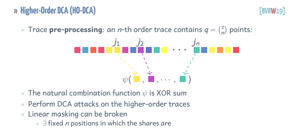
- Linear masking is a widely deploy gray-box countermeasure.
- The idea is to split any key-dependent sensitive intermediate variables in the implementation into n shares.
- Then the n shares are manipulated in a such way that any $n-1$ shares is independent of x.
- Apparently, linear masking of n shares is able to resist against $(n-1)$-th order DCA attacks.

» Higher-Order DCA (HO-DCA) [BVRW19]

- Trace **pre-processing**: an n-th order trace contains $q = \binom{t}{n}$ points:



- The natural combination function ψ is XOR sum
- Perform DCA attacks on the higher-order traces
- Linear masking can be broken
  - $\exists$ fixed n positions in which the shares are

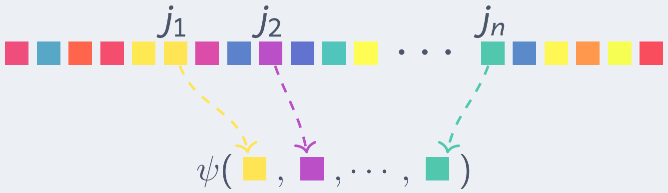


- The higher-order DCA attacks first pre-process the computation traces and obtain higher-order traces.
- Then apply standard DCA on the higher-order traces.
- A higher-order trace contains samples combining any n tuples in the original trace, hence it has $\binom{t}{n}$ points where t is the original trace size and n is the attacking order.
- The nature combination function to attack linear masking is XOR sum since it simply reveals the sensitive value.
- This sample corresponds to the original n linear shares.
- In an obscure white-box implementation, we don't know where the n shares are, hence t could be large, and higher-order DCA could be impractical.
- For example if the linear masking order $n = 5$, and the attacking trace window is 1000, there would be $\binom{1000}{5}$ shares in the higher-order trace.

» Higher-Order DCA (HO-DCA)

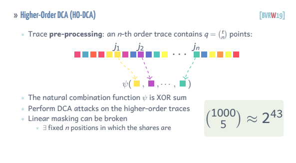
[BVRW19]

- Trace **pre-processing**: an n-th order trace contains $q = \binom{t}{n}$ points:



- The natural combination function ψ is XOR sum
- Perform DCA attacks on the higher-order traces
- Linear masking can be broken
  - $\exists$ fixed n positions in which the shares are

$$\binom{1000}{5} \approx 2^{43}$$



- The higher-order DCA attacks first pre-process the computation traces and obtain higher-order traces.
- Then apply standard DCA on the higher-order traces.
- A higher-order trace contains samples combining any n tuples in the original trace, hence it has $\binom{t}{n}$ points where t is the original trace size and n is the attacking order.
- The nature combination function to attack linear masking is XOR sum since it simply reveals the sensitive value.
- This sample corresponds to the original n linear shares.
- In an obscure white-box implementation, we don't know where the n shares are, hence t could be large, and higher-order DCA could be impractical.
- For example if the linear masking order $n = 5$, and the attacking trace window is 1000, there would be $\binom{1000}{5}$ shares in the higher-order trace.

» Linear Decoding Analysis (LDA)

[GPRW20]

- Assumption: there exists a linear (affine) decoding function

$$D(v_1, v_2, \cdots, v_t) = a_0 \oplus \left(\bigoplus_{1 \le i \le t} a_i \cdot v_i \right) = \varphi_k(x)$$

for some sensitive variable φ_k and some fixed coefficients $a_0, a_1, \cdots, a_t$.

» Linear Decoding Analysis (LDA) [GPRW20]
Assumption: there exists a linear (affine) decoding function
$D(v_1, v_2, \cdots, v_t) = a_0 \oplus \left(\bigoplus_{1 \le i \le t} a_i \cdot v_i \right) = \varphi_k(x)$
for some sensitive variable φ_k and some fixed coefficients $a_0, a_1, \cdots, a_t$.

- Linear decoding analysis is an attack whose complexity is independent with the linear masking order
- The attack assumes that there exists a linear decoding function that could reveal the key-dependent sensitive variable.
- The LDA adversary first collects many computation traces and make predictions of the attacked variables according to some key guess and try to solve this linear system.

» Linear Decoding Analysis (LDA) [GPRW20]

- Assumption: there exists a linear (affine) decoding function

$$D(v_1, v_2, \cdots, v_t) = a_0 \oplus \left(\bigoplus_{1 \le i \le t} a_i \cdot v_i \right) = \varphi_k(x)$$

  for some sensitive variable φ_k and some fixed coefficients $a_0, a_1, \cdots, a_t$.
- Record the v_i's over N executions:

$$v_1^{(1)} \quad \cdots \quad v_t^{(1)} \qquad\qquad x^{(1)}$$

» Linear Decoding Analysis (LDA) [GPRW20]
Assumption: there exists a linear (affine) decoding function
$D(v_1, v_2, \cdots, v_t) = a_0 \oplus \left(\bigoplus_{1 \le i \le t} a_i \cdot v_i \right) = \varphi_k(x)$
for some sensitive variable φ_k and some fixed coefficients $a_0, a_1, \cdots, a_t$.
Record the v_i's over N executions:
$v_1^{(1)} \cdots v_t^{(1)} \quad x^{(1)}$

- Linear decoding analysis is an attack whose complexity is independent with the linear masking order
- The attack assumes that there exists a linear decoding function that could reveal the key-dependent sensitive variable.
- The LDA adversary first collects many computation traces and make predictions of the attacked variables according to some key guess and try to solve this linear system.

» Linear Decoding Analysis (LDA) [GPRW20]

- Assumption: there exists a linear (affine) decoding function

$$D(v_1, v_2, \cdots, v_t) = a_0 \oplus \left(\bigoplus_{1 \le i \le t} a_i \cdot v_i \right) = \varphi_k(x)$$

for some sensitive variable φ_k and some fixed coefficients $a_0, a_1, \cdots, a_t$.

- Record the v_i's over N executions:

$$\begin{array}{ccccc} v_1^{(1)} & \cdots & v_t^{(1)} & \quad & x^{(1)} \\ v_1^{(2)} & \cdots & v_t^{(2)} & & x^{(2)} \\ \vdots & \ddots & \vdots & & \vdots \\ v_1^{(N)} & \cdots & v_t^{(N)} & & x^{(N)} \end{array}$$

- Linear decoding analysis is an attack whose complexity is independent with the linear masking order
- The attack assumes that there exists a linear decoding function that could reveal the key-dependent sensitive variable.
- The LDA adversary first collects many computation traces and make predictions of the attacked variables according to some key guess and try to solve this linear system.

» Linear Decoding Analysis (LDA) [GPRW20]

- Assumption: there exists a linear (affine) decoding function

$$D(v_1, v_2, \cdots, v_t) = a_0 \oplus \left(\bigoplus_{1 \le i \le t} a_i \cdot v_i \right) = \varphi_k(x)$$

  for some sensitive variable φ_k and some fixed coefficients $a_0, a_1, \cdots, a_t$.
- Record the v_i's over N executions:

$$\begin{array}{ccccc} v_1^{(1)} & \cdots & v_t^{(1)} & \quad & \varphi_k(x^{(1)}) \\ v_1^{(2)} & \cdots & v_t^{(2)} & & \varphi_k(x^{(2)}) \\ \vdots & \ddots & \vdots & & \vdots \\ v_1^{(N)} & \cdots & v_t^{(N)} & & \varphi_k(x^{(N)}) \end{array}$$

- Linear decoding analysis is an attack whose complexity is independent with the linear masking order
- The attack assumes that there exists a linear decoding function that could reveal the key-dependent sensitive variable.
- The LDA adversary first collects many computation traces and make predictions of the attacked variables according to some key guess and try to solve this linear system.

» Linear Decoding Analysis (LDA) [GPRW20]

- Assumption: there exists a linear (affine) decoding function

$$D(v_1, v_2, \cdots, v_t) = a_0 \oplus \left(\bigoplus_{1 \le i \le t} a_i \cdot v_i \right) = \varphi_k(x)$$

for some sensitive variable φ_k and some fixed coefficients $a_0, a_1, \cdots, a_t$.

- Record the v_i's over N executions:

$$\begin{bmatrix} 1 & v_1^{(1)} & \cdots & v_t^{(1)} \\ 1 & v_1^{(2)} & \cdots & v_t^{(2)} \\ 1 & \vdots & \ddots & \vdots \\ 1 & v_1^{(N)} & \cdots & v_t^{(N)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \varphi_k(x^{(1)}) \\ \varphi_k(x^{(2)}) \\ \vdots \\ \varphi_k(x^{(N)}) \end{bmatrix}$$

- Linear decoding analysis is an attack whose complexity is independent with the linear masking order
- The attack assumes that there exists a linear decoding function that could reveal the key-dependent sensitive variable.
- The LDA adversary first collects many computation traces and make predictions of the attacked variables according to some key guess and try to solve this linear system.

» Linear Decoding Analysis (LDA) (cont.) [GPRW20]

- Record the v_i's over N executions:

$$\begin{bmatrix} 1 & v_1^{(1)} & \cdots & v_t^{(1)} \\ 1 & v_1^{(2)} & \cdots & v_t^{(2)} \\ 1 & \vdots & \ddots & \vdots \\ 1 & v_1^{(N)} & \cdots & v_t^{(N)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} \varphi_k(x^{(1)}) \\ \varphi_k(x^{(2)}) \\ \vdots \\ \varphi_k(x^{(N)}) \end{bmatrix}$$

- Linear masking is vulnerable to LDA
  - system solvable for k^*
  - but not for incorrect key guess $k^\times$
- Trace Complexity $t + \mathcal{O}(1)$
- Computation complexity $\mathcal{O}(t^{2.8} \cdot |\mathcal{K}|)$

$$1000^{2.8} \approx 2^{28}$$

- Apparently, this system is solvable for the correct key guess, but not for the incorrect key guesses as long as the number of traces N is slightly large than window size t.
- Hence, the trace complexity of LDA attack is $t + \mathcal{O}(1)$
- And the computation complexity is the complexity to solve *a* linear system times the size of the key search space.
- For example, when the attacking window size $t = 1000$, this complexity to solve one linear system is $1000^{2.8}$, which is independent with the linear masking order.

Advanced Gray-Box Countermeasures and Attacks

* Linear Masking, Higher-Order DCA, and Linear Decoding Analysis
* Algebraic Security and Non-Linear Masking
* Shuffling

Advanced Gray-Box Countermeasures and Attacks
* Linear Masking, Higher-Order DCA, and Linear Decoding Analysis
* Algebraic Security and Non-Linear Masking
* Shuffling

Next, we talk about algebraic security and non-linear masking countermeasure.

» Algebraic Security and Non-Linear Masking [BU18]

* Introduced by Biryukov and Udovenko at Asiacrypt 2018
* To capture LDA like algebraic attack

A *d*-th degree algebraically-secure non-linear masking ensures that any function of up to *d* degree to the intermediate variables should not compute a "predictable" variable.

» Algebraic Security and Non-Linear Masking [BU18]
Introduced by Biryukov and Udovenko at Asiacrypt 2018
To capture LDA like algebraic attack
A *d*-th degree algebraically-secure non-linear masking ensures that any function of up to *d* degree to the intermediate variables should not compute a "predictable" variable.

* Algebraic security is proposed by Biryukov and Udovenko at Asiacrypt 2018
* A *d*-th degree algebraically-secure non-linear masking ensures that any function of up to *d* degree to the intermediate variables should not compute a "predictable" variable.

» First-Degree Secure Non-Linear Masking [BU18]

- Quadratic decoding function

$$(a, b, c) \mapsto ab \oplus c$$

- Secure gadgets for bit XOR, bit AND, and refresh
- Provably secure composition
- But vulnerable to DCA attack

$$\mathrm{Cor}(ab \oplus c,\ c) = \frac{1}{2}$$

- They suggest using a combination of linear masking and non-linear masking to thwart both DCA (probing security) and LDA (algebraic security).

- They also design a first degree secure non-linear masking scheme which resist against LDA attack.
- In their scheme, each sensitive variable is encoded by three variables a, b, c and with a quadratic decoding function $ab \oplus c$.
- They designed secure gadgets for bitwise operations and proved their composability
- Notice this scheme is vulnerable to DCA attack since the correlation score between the sensitive variable and c is $\frac{1}{2}$.
- They suggest using a combination of linear masking and non-linear masking to thwart both DCA (probing security) and LDA (algebraic security).

» Combination of Linear Masking and Non-linear Masking

We suggest three possible natural combinations:

1. apply linear masking on top of non-linear masking

$$x = (a_1 \oplus a_2 \oplus \cdots \oplus a_n)(b_1 \oplus b_2 \oplus \cdots \oplus b_n) \oplus (c_1 \oplus c_2 \oplus \cdots \oplus c_n)$$

2. apply non-linear masking on top of linear masking

$$x = (a_1 b_1 \oplus c_1) \oplus (a_2 b_2 \oplus c_2) \oplus \cdots \oplus (a_n b_n \oplus c_n) \ .$$

3. merge the two maskings into a new encoding

$$x = ab \oplus c_1 \oplus c_2 \oplus \cdots \oplus c_n \ .$$

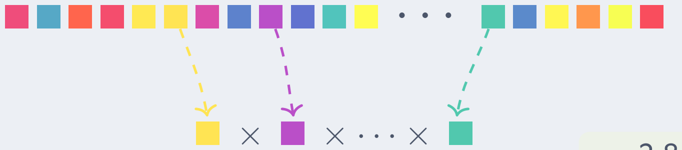
» Combination of Linear Masking and Non-linear Masking
We suggest three possible natural combinations:
1. apply linear masking on top of non-linear masking
$x = (a_1 \oplus a_2 \oplus \cdots \oplus a_n)(b_1 \oplus b_2 \oplus \cdots \oplus b_n) \oplus (c_1 \oplus c_2 \oplus \cdots \oplus c_n)$
2. apply non-linear masking on top of linear masking
$x = (a_1 b_1 \oplus c_1) \oplus (a_2 b_2 \oplus c_2) \oplus \cdots \oplus (a_n b_n \oplus c_n)$.
3. merge the two maskings into a new encoding
$x = ab \oplus c_1 \oplus c_2 \oplus \cdots \oplus c_n$.

- Although, it was suggested by Biryukov and Udovenko to combine linear masking and non-linear masking, but they didn't show how to do it.
- In the presentation, we suggest three nature ways to combine these two countermeasures.
- The first combination consist in applying linear masking on top of non-linear masking, that is the sensitive variable firstly non-linearly shared, then each non-linear share further linearly shared
- the second ...
- the 3rd..., two interpretations
- For first two combinations, the combined masking gadgets can be simply derived from the original gadgets of both schemes.
- For the third combination, new gadgets have to be developped, although it is not the purpose of this paper.

» Higher-Degree Decoding Analysis (HDDA)

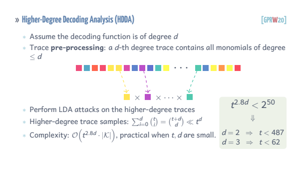
[GPRW20]

- Assume the decoding function is of degree d
- Trace **pre-processing**: *a d-th degree trace contains all monomials of degree $\leq d$*



- Perform LDA attacks on the higher-degree traces
- Higher-degree trace samples: $\sum_{i=0}^{d} \binom{t}{i} = \binom{t+d}{d} \ll t^d$
- Complexity: $\mathcal{O}\left(t^{2.8d} \cdot |\mathcal{K}|\right)$, practical when t, d are small.

$$t^{2.8d} < 2^{50}$$
$$\Downarrow$$
$$d = 2 \;\Rightarrow\; t < 487$$
$$d = 3 \;\Rightarrow\; t < 62$$



- A d-th degree decoding analysis assumes that there exists a d-th degree decoding function that reveals the sensitive variables.
- Similar to higher-order DCA, higher-degree decoding analysis also first pre-processes the original computation traces, then apply a LDA analysis.
- A d-th degree trace contains of all monomial of degree not greater d, hence it contains t^d samples.
- The complexity of HDDA hence is $\mathcal{O}(()t^{2.8d})$ times the size of key space.
- Note that this complexity can be only be practical when both the attack window size t and the decoding function degree d are small.
- For instance, if we want to bound $t^{2.8d} \leq 2^{50}$, if $d = 2$, then $t < 487$ and if $t = 128$ implies $d \leq 2$.

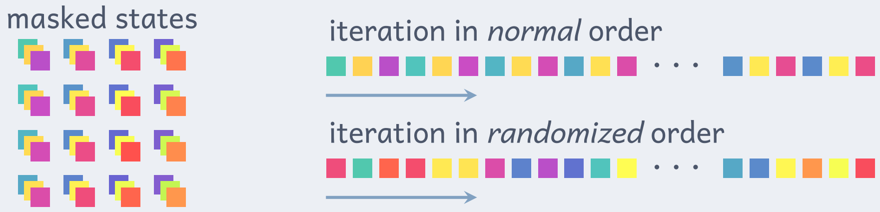
Advanced Gray-Box Countermeasures and Attacks

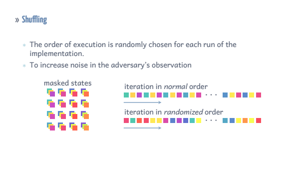
* Linear Masking, Higher-Order DCA, and Linear Decoding Analysis
* Algebraic Security and Non-Linear Masking
* Shuffling

Advanced Gray-Box Countermeasures and Attacks
* Linear Masking, Higher-Order DCA, and Linear Decoding Analysis
* Algebraic Security and Non-Linear Masking
* Shuffling

» Shuffling

- The order of execution is randomly chosen for each run of the implementation.
- To increase noise in the adversary's observation



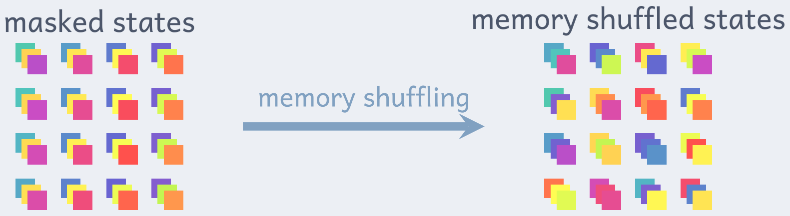


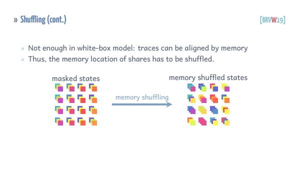
- Shuffling is another commonly used gray-box countermeasure
- The idea is to randomly choose the execution order in each run of the implementation
- thus, the noise in the adversary's observation will be increase

» Shuffling (cont.)

[BRVW19]

* Not enough in white-box model: traces can be aligned by memory
* Thus, the memory location of shares has to be shuffled.





- It has been shown that only shuffling operation order is insufficient in the white-box model, since the adversary could realign the trace sample according their memory address.
- Hence, we have to shuffle the usage of memory as well

» HO-DCA and Integrated HO-DCA against Masking and Shuffling

	shuffling degree λ	
	correlation decrease	attack slowdown
HODCA	λ	λ^2

» HO-DCA and Integrated HO-DCA against Masking and Shuffling

	shuffling degree λ	
	correlation decrease	attack slowdown
HODCA	λ	λ^2

- $\nexists$ n fixed locations for all shares
- Shuffling degree is λ
  - correlation score decreased by a factor of λ
  - attack slow down by a factor of λ^2
- Integrate values from all λ slots
  - correlation score decreased by a factor of $\sqrt{\lambda}$
  - attack slow down by a factor of λ

» HO-DCA and Integrated HO-DCA against Masking and Shuffling

	shuffling degree λ	
	correlation decrease	attack slowdown
HODCA	λ	λ^2
Integrated HODCA	$\sqrt{\lambda}$	λ

» HO-DCA and Integrated HO-DCA against Masking and Shuffling

	shuffling degree λ	
	correlation decrease	attack slowdown
HODCA	λ	λ^2
Integrated HODCA	$\sqrt{\lambda}$	λ

- $\nexists$ n fixed locations for all shares
- Shuffling degree is λ
  - correlation score decreased by a factor of λ
  - attack slow down by a factor of λ^2
- Integrate values from all λ slots
  - correlation score decreased by a factor of $\sqrt{\lambda}$
  - attack slow down by a factor of λ

Data-Dependency Analysis

- Data-Dependency Graph
- Data-Dependency Analysis against Masking Combinations

Data-Dependency Analysis
- Data-Dependency Graph
- Data-Dependency Analysis against Masking Combinations

- When masking and shuffling both applied, if the attacking windows and linear masking order is big, the HO-DCA attack would be impractical.
- Next part, we try to improve DCA by exploting data-dependency of the implementation

Data-Dependency Analysis

* Data-Dependency Graph
* Data-Dependency Analysis against Masking Combinations

Data-Dependency Analysis
* Data-Dependency Graph
* Data-Dependency Analysis against Masking Combinations

» Data Dependency Graph

- White-box adversary also observes data-flow.
- Data-dependency graph (DDG) can visually reveal the structure of the implementation.

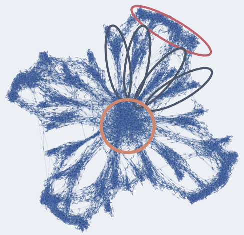


Illustration from [GPRW20]

- The white-box adversary can observe any internal states of a white-box implementation, as well as its data flow.
- From the data flow, an adversary could build the data-dependency graph of an white-box implementation.
- While breaking the winning implementation of WhibOx 2017, we gradually reveal the structure of the AES first round by plotting data-dependency graph and manage to locate the first round sbox output encoded in a window of about 50 variables.

Data-Dependency Analysis

- Data-Dependency Graph
- Data-Dependency Analysis against Masking Combinations

Data-Dependency Analysis
- Data-Dependency Graph
- Data-Dependency Analysis against Masking Combinations

- However, the data dependency graph dose not always visually disclose implementation details.
- Nevertheless, to attack an obscure white-box implementation, it is still minimize the attacking trace window by exploiting the data-dependency in an automatic ways.
- Hereafter, we show that how data-dependency analysis can be used to break linear masking and non-linear masking combinations.

» Linear Masking Gadget for AND

[ISW03]

$$(x_1,\ x_2, \cdots,\ x_n),\ (y_1,\ y_2, \cdots,\ y_n) \ \mapsto\ (z_1,\ z_2, \cdots,\ z_n) \quad \text{s.t.} \bigoplus_i x_i \cdot \bigoplus_i y_i = \bigoplus_i z_i\ .$$

- Our analysis is inspired by an observation in linear masking gadget for AND operation.
- An secure linear masking gadget takes the linear shares of two variables and obtain the linear shares of their product.
- The linear share gadgets can be interpreted as sum of three matrices and then sum the values in the same row.
- If you look at a share of x: x_1, it multiplies with all share of y
- This is true for the second shares x_2 and third share x_3
- Each x_i is multiplied with all shares of y: $(y_j)_j$, vice versa.
- True for any order n

» Linear Masking Gadget for AND [ISW03]

$$(x_1, x_2, \cdots, x_n), (y_1, y_2, \cdots, y_n) \mapsto (z_1, z_2, \cdots, z_n) \quad \text{s.t.} \bigoplus_i x_i \cdot \bigoplus_i y_i = \bigoplus_i z_i .$$

$$\begin{bmatrix} x_1y_1 & 0 & 0 \\ x_1y_2 & x_2y_2 & 0 \\ x_1y_3 & x_2y_3 & x_3y_3 \end{bmatrix} \oplus \begin{bmatrix} 0 & x_2y_1 & x_3y_1 \\ 0 & 0 & x_3y_2 \\ 0 & 0 & 0 \end{bmatrix}^T \oplus \begin{bmatrix} 0 & r_{1,2} & r_{1,3} \\ r_{1,2} & 0 & r_{2,3} \\ r_{1,3} & r_{2,3} & 0 \end{bmatrix} \xrightarrow{\text{sum rows}} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

- Our analysis is inspired by an observation in linear masking gadget for AND operation.
- An secure linear masking gadget takes the linear shares of two variables and obtain the linear shares of their product.
- The linear share gadgets can be interpreted as sum of three matrices and then sum the values in the same row.
- If you look at a share of x: x_1, it multiplies with all share of y
- This is true for the second shares x_2 and third share x_3
- Each x_i is multiplied with all shares of y: $(y_j)_j$, vice versa.
- True for any order n

» Linear Masking Gadget for AND

[ISW03]

$$(x_1,\ x_2, \cdots,\ x_n),\ (y_1,\ y_2, \cdots,\ y_n) \mapsto (z_1,\ z_2, \cdots,\ z_n) \quad \text{s.t.} \bigoplus_i x_i \cdot \bigoplus_i y_i = \bigoplus_i z_i\ .$$

$$\begin{bmatrix} x_1 y_1 & 0 & 0 \\ x_1 y_2 & x_2 y_2 & 0 \\ x_1 y_3 & x_2 y_3 & x_3 y_3 \end{bmatrix} \oplus \begin{bmatrix} 0 & x_2 y_1 & x_3 y_1 \\ 0 & 0 & x_3 y_2 \\ 0 & 0 & 0 \end{bmatrix}^T \oplus \begin{bmatrix} 0 & r_{1,2} & r_{1,3} \\ r_{1,2} & 0 & r_{2,3} \\ r_{1,3} & r_{2,3} & 0 \end{bmatrix} \xrightarrow{\text{sum rows}} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

- Our analysis is inspired by an observation in linear masking gadget for AND operation.
- An secure linear masking gadget takes the linear shares of two variables and obtain the linear shares of their product.
- The linear share gadgets can be interpreted as sum of three matrices and then sum the values in the same row.
- If you look at a share of x: x_1, it multiplies with all share of y
- This is true for the second shares x_2 and third share x_3
- Each x_i is multiplied with all shares of y: $(y_j)_j$, vice versa.
- True for any order n

» Linear Masking Gadget for AND [ISW03]

$$(x_1,\ x_2,\cdots,\ x_n),\ (y_1,\ y_2,\cdots,\ y_n)\ \mapsto\ (z_1,\ z_2,\cdots,\ z_n)\quad \text{s.t.} \bigoplus_i x_i \cdot \bigoplus_i y_i = \bigoplus_i z_i\ .$$

$$\begin{bmatrix} x_1y_1 & 0 & 0 \\ x_1y_2 & x_2y_2 & 0 \\ x_1y_3 & x_2y_3 & x_3y_3 \end{bmatrix} \oplus \begin{bmatrix} 0 & x_2y_1 & x_3y_1 \\ 0 & 0 & x_3y_2 \\ 0 & 0 & 0 \end{bmatrix}^T \oplus \begin{bmatrix} 0 & r_{1,2} & r_{1,3} \\ r_{1,2} & 0 & r_{2,3} \\ r_{1,3} & r_{2,3} & 0 \end{bmatrix} \xrightarrow{\text{sum rows}} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

» Linear Masking Gadget for AND [ISW03]

- Our analysis is inspired by an observation in linear masking gadget for AND operation.
- An secure linear masking gadget takes the linear shares of two variables and obtain the linear shares of their product.
- The linear share gadgets can be interpreted as sum of three matrices and then sum the values in the same row.
- If you look at a share of x: x_1, it multiplies with all share of y
- This is true for the second shares x_2 and third share x_3
- Each x_i is multiplied with all shares of y: $(y_j)_j$, vice versa.
- True for any order n

» Linear Masking Gadget for AND [ISW03]

$$(x_1,\ x_2, \cdots,\ x_n),\ (y_1,\ y_2, \cdots,\ y_n) \ \mapsto \ (z_1,\ z_2, \cdots,\ z_n) \quad \text{s.t.} \bigoplus_i x_i \cdot \bigoplus_i y_i = \bigoplus_i z_i\,.$$

$$\begin{bmatrix} x_1y_1 & 0 & 0 \\ x_1y_2 & x_2y_2 & 0 \\ x_1y_3 & x_2y_3 & x_3y_3 \end{bmatrix} \oplus \begin{bmatrix} 0 & x_2y_1 & x_3y_1 \\ 0 & 0 & x_3y_2 \\ 0 & 0 & 0 \end{bmatrix}^T \oplus \begin{bmatrix} 0 & r_{1,2} & r_{1,3} \\ r_{1,2} & 0 & r_{2,3} \\ r_{1,3} & r_{2,3} & 0 \end{bmatrix} \xrightarrow{\text{sum rows}} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

- Our analysis is inspired by an observation in linear masking gadget for AND operation.
- An secure linear masking gadget takes the linear shares of two variables and obtain the linear shares of their product.
- The linear share gadgets can be interpreted as sum of three matrices and then sum the values in the same row.
- If you look at a share of x: x_1, it multiplies with all share of y
- This is true for the second shares x_2 and third share x_3
- Each x_i is multiplied with all shares of y: $(y_j)_j$, vice versa.
- True for any order n

» Linear Masking Gadget for AND

[ISW03]

$$(x_1,\ x_2, \cdots,\ x_n),\ (y_1,\ y_2, \cdots,\ y_n) \ \mapsto\ (z_1,\ z_2, \cdots,\ z_n) \quad \text{s.t.} \bigoplus_i x_i \cdot \bigoplus_i y_i = \bigoplus_i z_i\,.$$

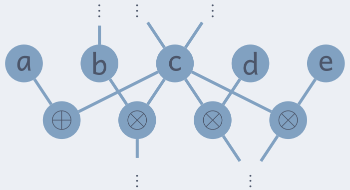
$$\begin{bmatrix} x_1y_1 & 0 & 0 \\ x_1y_2 & x_2y_2 & 0 \\ x_1y_3 & x_2y_3 & x_3y_3 \end{bmatrix} \oplus \begin{bmatrix} 0 & x_2y_1 & x_3y_1 \\ 0 & 0 & x_3y_2 \\ 0 & 0 & 0 \end{bmatrix}^T \oplus \begin{bmatrix} 0 & r_{1,2} & r_{1,3} \\ r_{1,2} & 0 & r_{2,3} \\ r_{1,3} & r_{2,3} & 0 \end{bmatrix} \xrightarrow{\text{sum rows}} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

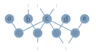
Each x_i is multiplied with all shares of y: $(y_j)_j$, vice versa.

- Our analysis is inspired by an observation in linear masking gadget for AND operation.
- An secure linear masking gadget takes the linear shares of two variables and obtain the linear shares of their product.
- The linear share gadgets can be interpreted as sum of three matrices and then sum the values in the same row.
- If you look at a share of x: x_1, it multiplies with all share of y
- This is true for the second shares x_2 and third share x_3
- Each x_i is multiplied with all shares of y: $(y_j)_j$, vice versa.
- True for any order n

» Data-Dependency Analysis against Masking Combinations

[GRW20]



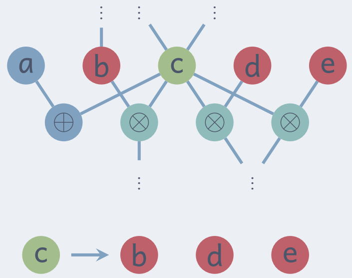


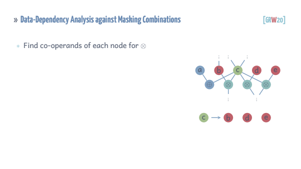
- Now consider a circuit-based white-box implementation
- For each gate, we compute its co-operand for AND operation
- For instance, gate *b*, *d*, *e* are the co=operation of gate *c* for AND operation
- As we shown above, if *c* happens to be a share of some variable, then *b*, *d*, *e* could be the linear shares of some sensitive intermediate variable
- Our data-dependency analysis then collect data-dependency traces which contains the sum of the co-operands of each gate for AND operation
- Then we perform standard DCA attack on data-dependency traces
- The attack would succeed if some biased value recovered in the data dependency traces
- Note that if there are noisy values in the co-operand set, we could simply compute the sum of all the subsets of the co-operands set.
- This would substantially improve the computation complexity by avoiding

» Data-Dependency Analysis against Masking Combinations

[GRW20]

* Find co-operands of each node for ⊗

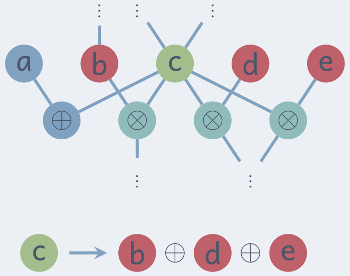


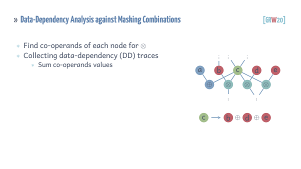


- Now consider a circuit-based white-box implementation
- For each gate, we compute its co-operand for AND operation
- For instance, gate b, d, e are the co=operation of gate c for AND operation
- As we shown above, if c happens to be a share of some variable, then b, d, e could be the linear shares of some sensitive intermediate variable
- Our data-dependency analysis then collect data-dependency traces which contains the sum of the co-operands of each gate for AND operation
- Then we perform standard DCA attack on data-dependency traces
- The attack would succeed if some biased value recovered in the data dependency traces
- Note that if there are noisy values in the co-operand set, we could simply compute the sum of all the subsets of the co-operands set.
- This would substantially improve the computation complexity by avoiding

» Data-Dependency Analysis against Masking Combinations [GRW20]

- Find co-operands of each node for ⊗
- Collecting data-dependency (DD) traces
  - Sum co-operands values

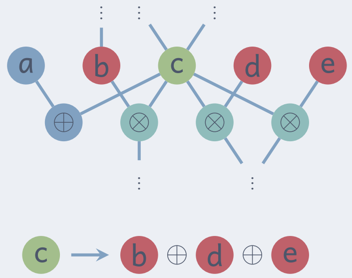


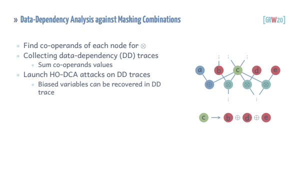


- Now consider a circuit-based white-box implementation
- For each gate, we compute its co-operand for AND operation
- For instance, gate *b*, *d*, *e* are the co=operation of gate *c* for AND operation
- As we shown above, if *c* happens to be a share of some variable, then *b*, *d*, *e* could be the linear shares of some sensitive intermediate variable
- Our data-dependency analysis then collect data-dependency traces which contains the sum of the co-operands of each gate for AND operation
- Then we perform standard DCA attack on data-dependency traces
- The attack would succeed if some biased value recovered in the data dependency traces
- Note that if there are noisy values in the co-operand set, we could simply compute the sum of all the subsets of the co-operands set.
- This would substantially improve the computation complexity by avoiding

» Data-Dependency Analysis against Masking Combinations [GRW20]

- Find co-operands of each node for ⊗
- Collecting data-dependency (DD) traces
  - Sum co-operands values
- Launch HO-DCA attacks on DD traces
  - Biased variables can be recovered in DD trace

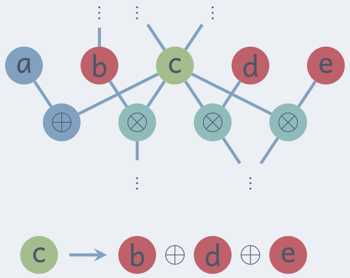


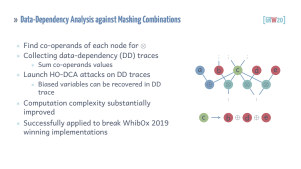


- Now consider a circuit-based white-box implementation
- For each gate, we compute its co-operand for AND operation
- For instance, gate *b*, *d*, *e* are the co=operation of gate *c* for AND operation
- As we shown above, if *c* happens to be a share of some variable, then *b*, *d*, *e* could be the linear shares of some sensitive intermediate variable
- Our data-dependency analysis then collect data-dependency traces which contains the sum of the co-operands of each gate for AND operation
- Then we perform standard DCA attack on data-dependency traces
- The attack would succeed if some biased value recovered in the data dependency traces
- Note that if there are noisy values in the co-operand set, we could simply compute the sum of all the subsets of the co-operands set.
- This would substantially improve the computation complexity by avoiding

» Data-Dependency Analysis against Masking Combinations [GRW20]

- Find co-operands of each node for ⊗
- Collecting data-dependency (DD) traces
  - Sum co-operands values
- Launch HO-DCA attacks on DD traces
  - Biased variables can be recovered in DD trace
- Computation complexity substantially improved
- Successfully applied to break WhibOx 2019 winning implementations





- Now consider a circuit-based white-box implementation
- For each gate, we compute its co-operand for AND operation
- For instance, gate *b*, *d*, *e* are the co=operation of gate *c* for AND operation
- As we shown above, if *c* happens to be a share of some variable, then *b*, *d*, *e* could be the linear shares of some sensitive intermediate variable
- Our data-dependency analysis then collect data-dependency traces which contains the sum of the co-operands of each gate for AND operation
- Then we perform standard DCA attack on data-dependency traces
- The attack would succeed if some biased value recovered in the data dependency traces
- Note that if there are noisy values in the co-operand set, we could simply compute the sum of all the subsets of the co-operands set.
- This would substantially improve the computation complexity by avoiding

» Attack Comparison

	linear masking		linear + NL masking	
	#trace	computation	#trace	computation
		without shuffling		
LDA/HDDA	$t + \mathcal{O}(1)$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{2.8})$	$\mathcal{O}(t^2)$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{5.6})$
HODCA	c	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^n)$	$4\,c$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^n)$
DD-DCA	c	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t)$	$4\,c$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t)$

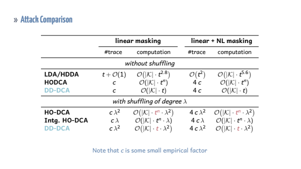
Note that c is some small empirical factor

- Now, we have revisited all advanced gray-box countermeasures and attacks.
- Let's give a comparison of different attacks against different countermeasure combinations.
- We first consider when shuffling is absent,
  - The LDA against linear masking requires t traces and its computation complexity is $\mathcal{O}(|\mathcal{K}| \cdot t^{2.8})$
  - the HDDA against linear masking and non-linear masking combination require $\mathcal{O}(t^2)$ traces and $\mathcal{O}(|\mathcal{K}| \cdot t^{5.6})$ computations.
- When shuffling is applied
  - Algebraic attack does not work any work
  - HO-DCA requires $c \cdot \lambda^2$ traces and the complexity grows exponential with the linear masking order
  - When all shuffling slots are integrated, the attack is λ times faster

» Attack Comparison

	linear masking		linear + NL masking	
	#trace	computation	#trace	computation
without shuffling				
LDA/HDDA	$t+\mathcal{O}(1)$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{2.8})$	$\mathcal{O}(t^2)$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{5.6})$
HODCA	c	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{n})$	$4\,c$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{n})$
DD-DCA	c	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t)$	$4\,c$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t)$
with shuffling of degree λ				
HO-DCA	$c\,\lambda^2$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{n} \cdot \lambda^2)$	$4\,c\,\lambda^2$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{n} \cdot \lambda^2)$
Intg. HO-DCA	$c\,\lambda$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{n} \cdot \lambda)$	$4\,c\,\lambda$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{n} \cdot \lambda)$
DD-DCA	$c\,\lambda^2$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t \cdot \lambda^2)$	$4\,c\,\lambda^2$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t \cdot \lambda^2)$

Note that c is some small empirical factor

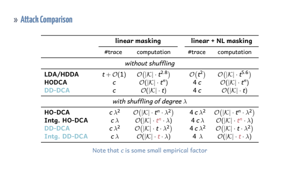


- Now, we have revisited all advanced gray-box countermeasures and attacks.
- Let's give a comparison of different attacks against different countermeasure combinations.
- We first consider when shuffling is absent,
  - The LDA against linear masking requires t traces and its computation complexity is $\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{2.8})$
  - the HDDA against linear masking and non-linear masking combination require $\mathcal{O}(t^2)$ traces and $\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{5.6})$ computations.
- When shuffling is applied
  - Algebraic attack does not work any work
  - HO-DCA requires $c \cdot \lambda^2$ traces and the complexity grows exponential with the linear masking order
  - When all shuffling slots are integrated, the attack is λ times faster

» Attack Comparison

	linear masking		linear + NL masking	
	#trace	computation	#trace	computation
without shuffling				
LDA/HDDA	$t + \mathcal{O}(1)$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{2.8})$	$\mathcal{O}(t^2)$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{5.6})$
HODCA	c	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^n)$	$4\,c$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^n)$
DD-DCA	c	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t)$	$4\,c$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t)$
with shuffling of degree λ				
HO-DCA	$c\,\lambda^2$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^n \cdot \lambda^2)$	$4\,c\,\lambda^2$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^n \cdot \lambda^2)$
Intg. HO-DCA	$c\,\lambda$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^n \cdot \lambda)$	$4\,c\,\lambda$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^n \cdot \lambda)$
DD-DCA	$c\,\lambda^2$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t \cdot \lambda^2)$	$4\,c\,\lambda^2$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t \cdot \lambda^2)$
Intg. DD-DCA	$c\,\lambda$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t \cdot \lambda)$	$4\ \lambda$	$\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t \cdot \lambda)$

Note that c is some small empirical factor



- Now, we have revisited all advanced gray-box countermeasures and attacks.
- Let's give a comparison of different attacks against different countermeasure combinations.
- We first consider when shuffling is absent,
  - The LDA against linear masking requires t traces and its computation complexity is $\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{2.8})$
  - the HDDA against linear masking and non-linear masking combination require $\mathcal{O}(t^2)$ traces and $\mathcal{O}(\lvert\mathcal{K}\rvert \cdot t^{5.6})$ computations.
- When shuffling is applied
  - Algebraic attack does not work any work
  - HO-DCA requires $c \cdot \lambda^2$ traces and the complexity grows exponential with the linear masking order
  - When all shuffling slots are integrated, the attack is λ times faster

Conclusion

Conclusion

» Conclusion

- Revisited state-of-the-art countermeasures employed in practice
  - Linear masking, non-linear masking, shuffling and how to combine them
- Quantified different (advanced) gray-box attack performance against different countermeasures
  - (Higher-order) DCA, (higher-degree) Decoding Analysis, · · ·
- Proposed new attacks based on data-dependency with substantial computation complexity improvement
- Broken three WhibOx 2019 winning challenges

paper ia.cr/2020/413

attack **CryptoExperts** / breaking-winning-challenges-of-whibox2019

└Conclusion

» Conclusion

- Revisited state-of-the-art countermeasures employed in practice
  - Linear masking, non-linear masking, shuffling and how to combine them
- Quantified different (advanced) gray-box attack performance against different countermeasures
  - (Higher-order) DCA, (higher-degree) Decoding Analysis, · · ·
- Proposed new attacks based on data-dependency with substantial computation complexity improvement
- Broken three WhibOx 2019 winning challenges

paper ia.cr/2020/413

attack CryptoExperts / breaking-winning-challenges-of-whibox2019