Tutorial on white-box cryptography

Matthieu Rivain

Aleksei Udovenko

CHES 2022 Tutorial
Leuven, 18 Sep. 2022

Overview

- Introduction to white-box cryptography
  - *Presentation ~1h*
- Generating and attacking white-box implementations
  - *Practical tutorial ~2h*

CryptoExperts
WE INNOVATE TO SECURE YOUR BUSINESS

UNIVERSITÉ DU LUXEMBOURG

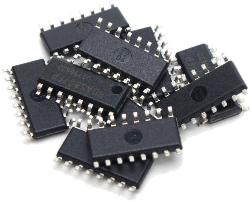
Overview of the presentation

- White-box crypto context
- White-box crypto in theory
  - definitions & security notions
- White-box crypto in practice
  - early designs & breaks
  - gray-box attacks & countermeasures
  - WhibOx competitions

White-box crypto context

How to protect a cryptographic key?

How to protect a cryptographic key?



Well, put it in a **smartcard** of course!

... or any piece of **secure hardware**

But...

- Secure hardware is **expensive** (production, integration, infrastructures...)
- Long lifecycle, limited updates
- Bugs, security flaws might occur
  - *e.g.* ROCA vulnerability (October 2017)


COMPLETELY BROKEN —

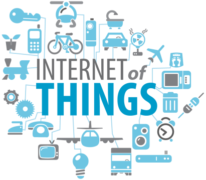
Millions of high-security crypto keys crippled by newly discovered flaw

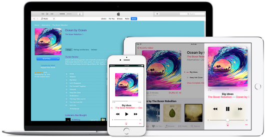
Factorization weakness lets attackers impersonate key holders and decrypt their data.

DAN GOODIN (US) - 16/10/2017, 16:30

Security in pure software

- Advantages: cheaper, faster time-to-market, easier to update
- Huge need for many contexts
  - Mobile apps (SE/TEE not always available)
  - IoT (cheap hardware)
  - Content protection, DRM
  - OS / firmwares



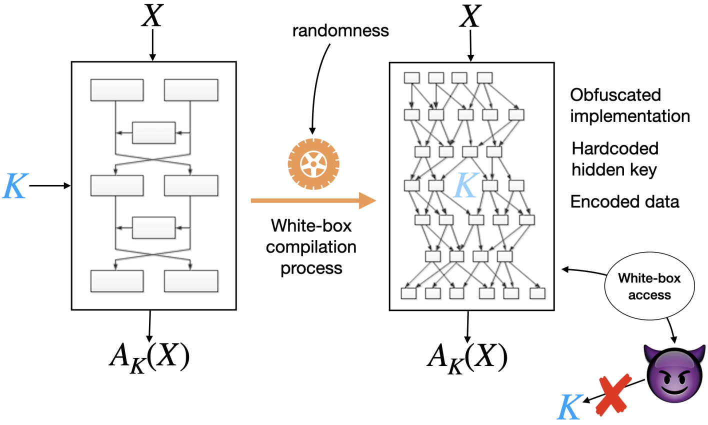


Protecting keys in software?

- Potential threats:
  - malwares
  - co-hosted applications
  - users themselves
- White-box adversary model
  - full control of the execution environment
  - analyse the code
  - access the memory
  - tamper with execution

White-box cryptography

General idea: hide the secret key in an obfuscated cryptographic implementation



White-box crypto in theory

What is a program?

- A word in a formal language $P \in \mathcal{L}$

$$\text{execute}: \begin{array}{rcl} \mathcal{L} \times \{0,1\}^* & \to & \{0,1\}^* \\ (P, input) & \mapsto & output \end{array}$$

(Universal Turing Machine)

- $|P|$: size of $P \in \mathcal{L}$
- time(P): $\#$ operations for execute$(P, \cdot)$

What is a program?

- $P \equiv f$ *(P implements f)*

$$\forall x \;:\; \text{execute}(P, x) = f(x)$$

- $P_1 \equiv P_2$ *(functional equivalence)*

$$\forall x \;:\; \text{execute}(P_1, x) = \text{execute}(P_2, x)$$

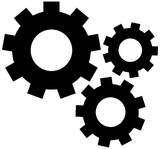
- Straight-line programs
  - no conditional statements, no loops
  - $|P| = \text{time}(P)$

What is an obfuscator?

- An algorithm:

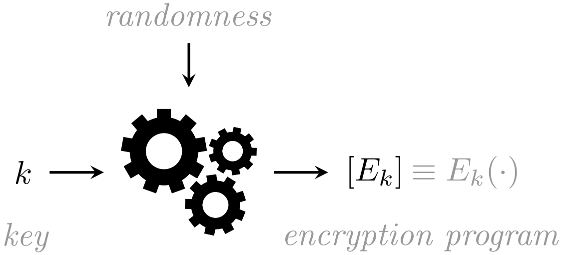
$$\textit{randomness}$$

$$\downarrow$$

$P \longrightarrow$  $\longrightarrow O(P) \equiv P$

- Size and execution time increase (hopefully not too much)

What is a white-box compiler?

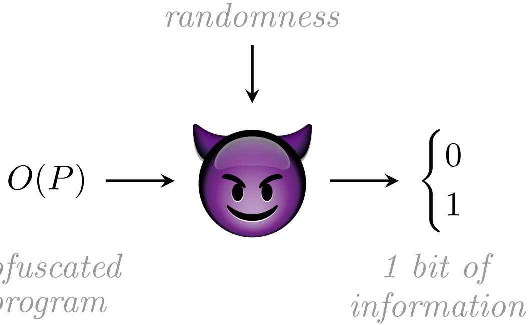


- Specific to an encryption function E
- Can be constructed from an obfuscator

$$k \to P \equiv E_k(\cdot) \xrightarrow{O} [E_k]$$

What is an adversary?

- An algorithm:

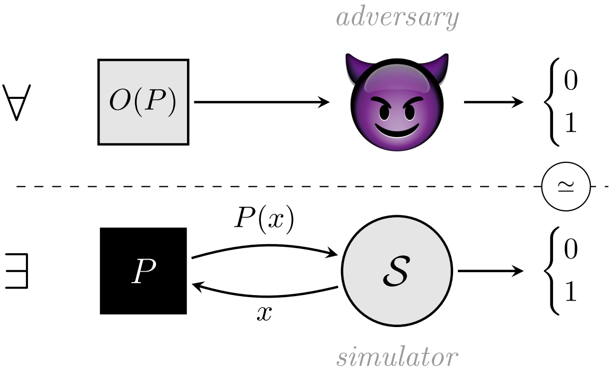


- Wlg: $\nexists$ 1-bit ♉ $\Rightarrow$ $\nexists$ multi-bit ♉

[BGI+01] *On the (Im)possibility of Obfuscating Programs* (CRYPTO 2001)

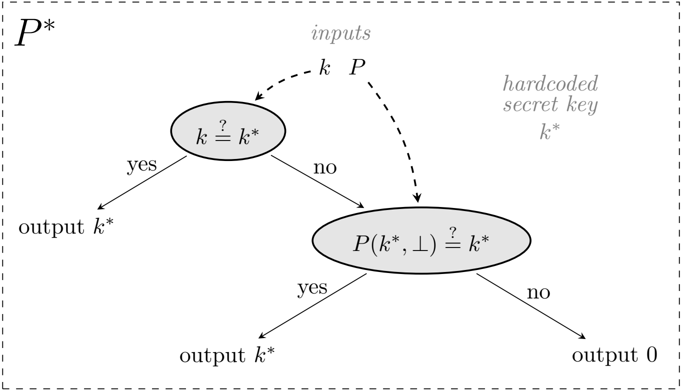
- Virtual Black Box (VBB) security notion
- Impossibility result: VBB cannot be achieved for all programs (counterexample)
- Indistinguishability Obfuscation (IO)

VBB security notion



- $O(P)$ reveals nothing more than the I/O behavior of P

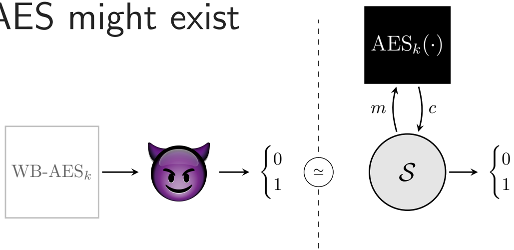
Impossibility result



- $P^*(0, P^*) = k^*$
- BB access to P^* reveals nothing

The good news

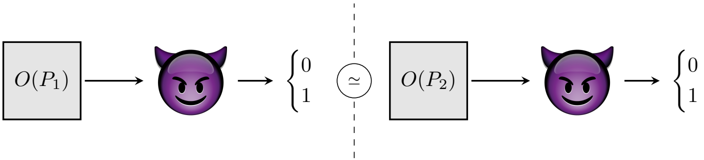
- The impossibility result does not apply to a given encryption algorithm
- VBB AES might exist



- The bad news: seems very hard to achieve

Indistinguishability Obfuscation (IO)

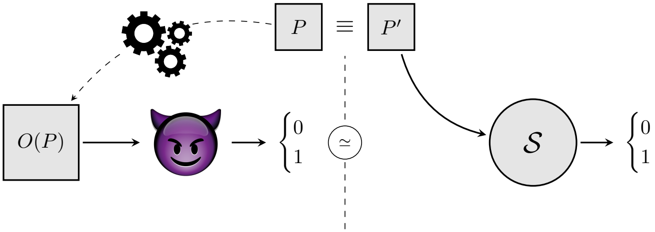
- Notion restricted to straight-line programs
- For any (P_1, P_2) st $P_1 \equiv P_2$ and $|P_1| = |P_2|$



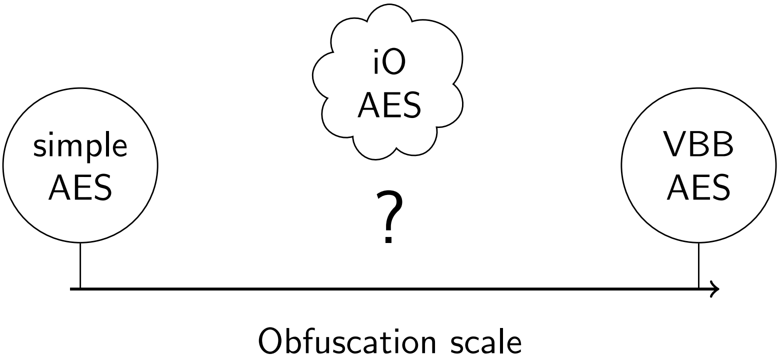
- *i.e.* $O(P_1)$ and $O(P_2)$ are indistinguishable

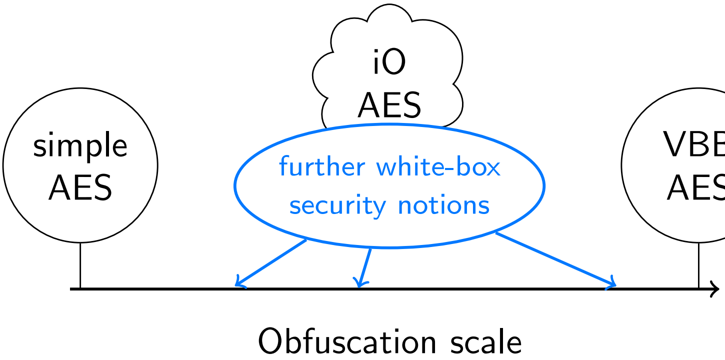
Why is IO meaningful?

- IO $\Leftrightarrow$ Best Possible Obfuscation
- For any P':



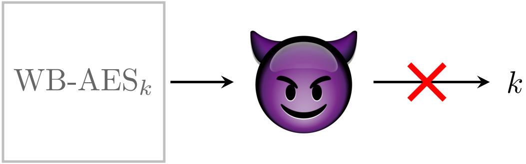
- $O(P)$ doesn't reveal anything more than the *best obfuscated* program P'


simple
AES
iO
AES
?
VBB
AES
Obfuscation scale


iO
AES
simple
AES
further white-box
security notions
VBB
AES
Obfuscation scale

White-box security notions

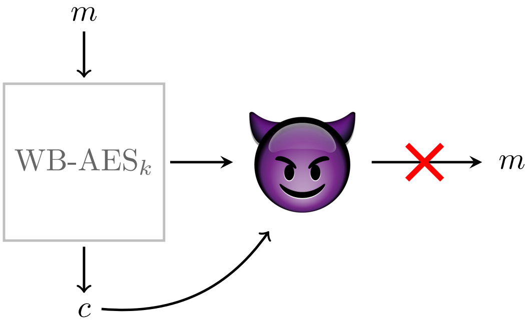
- **Unbreakability**: resistance to key extraction



- Basic requirement but insufficient in practice
- Other security notions
  - [SWP09] *Towards Security Notions for White-Box Cryptography* (ISC 2009)
  - [DLPR13] *White-Box Security Notions for Symmetric Encryption Schemes* (SAC 2013)

One-wayness

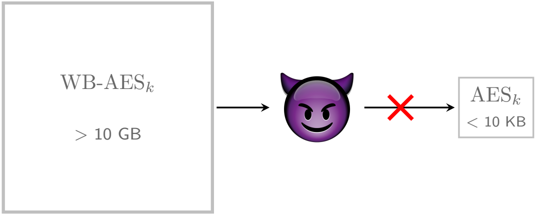
- **One-wayness**: hardness of inversion



- Turns AES into a public-key cryptosystem
- PK crypto with light-weight private operations

Incompressibility

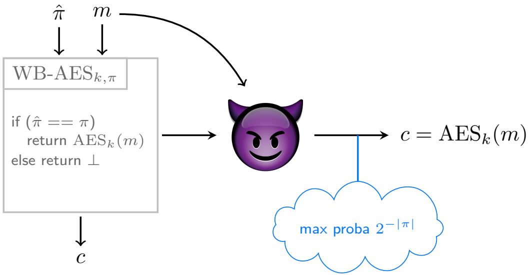
- **Incompressibility**: hardness of compression



- Makes the implementation less convenient to share at a large scale

Password

- **Password**: WB implem locked by password



- User password / application-dependent secret (a.k.a binding)

Some relations

- If the underlying encryption scheme is secure:

$$\begin{tikzcd} & \mathsf{INC} \arrow[d, Rightarrow] & \\ \mathsf{OW} \arrow[r, Rightarrow] & \mathsf{UBK} & \mathsf{PWD} \arrow[l, Rightarrow] \end{tikzcd}$$

Some relations

- If the underlying encryption scheme is secure:

$$\begin{array}{ccccccccc} & & & & \text{INC} & & & & \\ & & & & \Downarrow & & & & \\ \text{VBB} & \Rightarrow & \text{OW} & \Rightarrow & \text{UBK} & \Leftarrow & \text{PWD} & \Leftarrow & \text{VBB} \end{array}$$

Some relations

- If the underlying encryption scheme is secure:

$$\begin{array}{ccccccccc} & & & & \mathrm{VBB} & & & & \\ & & & & \not\Downarrow & & & & \\ & & & & \mathrm{INC} & & & & \\ & & & & \Downarrow & & & & \\ \mathrm{VBB} & \Rightarrow & \mathrm{OW} & \Rightarrow & \mathrm{UBK} & \Leftarrow & \mathrm{PWD} & \Leftarrow & \mathrm{VBB} \end{array}$$

Some relations

- If the underlying encryption scheme is secure:

$$
\begin{array}{ccccccccc}
 & & & & \text{VBB} & & & & \\
 & & & & \not\Downarrow & & & & \\
 & & & & \text{INC} & & & & \\
 & & & & \Downarrow & & & & \\
\text{VBB} & \Rightarrow & \text{OW} & \Rightarrow & \text{UBK} & \Leftarrow & \text{PWD} & \Leftarrow & \text{VBB}
\end{array}
$$

- No UBK construction known for AES

  $\Rightarrow$ no OW/INC/PWB/VBB construction either

Further white-box notions

- [DLPR13] *White-Box Security Notions for Symmetric Encryption Schemes* (SAC 2013)
  - Perturbation-Value Hiding (PVH) $\Rightarrow$ traceability
- [AABM20] *On the Security Goals of White-Box Cryptography* (CHES 2020)
  - Authenticated encryption
  - Hardware binding, application binding
- [ABFJM21] *Security Reductions for White-Box Key-Storage in Mobile Payments* (ASIACRYPT 2021)
  - Key derivation
  - Payment application

White-box crypto in practice

Original white-box AES

- [CEJV02] *White-Box Cryptography and an AES Implementation* (SAC 2002)
- First step: network of look-up tables
- Each round split in 4 *sub-rounds*

$$(x_0, x_5, x_{10}, x_{15}) \mapsto \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \otimes \begin{pmatrix} S(x_0 \oplus k_0) \\ S(x_5 \oplus k_5) \\ S(x_{10} \oplus k_{10}) \\ S(x_{15} \oplus k_{15}) \end{pmatrix}$$

Original white-box AES

- Computed as

$$T_0[x_0] \oplus T_5[x_5] \oplus T_{10}[x_{10}] \oplus T_{15}[x_{15}]$$

- Tables T_i : 8 bits → 32 bits

$$\begin{aligned}
T_0[x] &= S(x \oplus k_0) \times (\mathsf{02\ 01\ 01\ 03})^T \\
T_5[x] &= S(x \oplus k_5) \times (\mathsf{03\ 02\ 01\ 01})^T \\
T_{10}[x] &= S(x \oplus k_{10}) \times (\mathsf{01\ 03\ 02\ 01})^T \\
T_{15}[x] &= S(x \oplus k_{15}) \times (\mathsf{01\ 01\ 03\ 02})^T
\end{aligned}$$

- XOR table: 8 bits → 4 bits

$$T_{\mathsf{xor}}[x_0 \| x_1] = x_0 \oplus x_1$$

Original white-box AES

- Second step: randomize look-up tables
- Each table T is replaced by

$$T' = g \circ T \circ f^{-1}$$

where f, g are **random encodings**

- For two *connected* tables T, R

$$\begin{aligned} T' &= g \circ T \circ f^{-1} \\ R' &= h \circ R \circ g^{-1} \end{aligned} \quad \Rightarrow \quad R' \circ T' = h \circ (R \circ T) \circ f^{-1}$$

Original white-box AES

- Intuition: encoded tables bring no information
- True for a single (bijective) table $g \circ T \circ f^{-1}$
- Not for the large picture

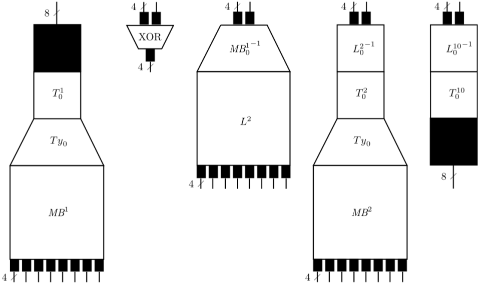
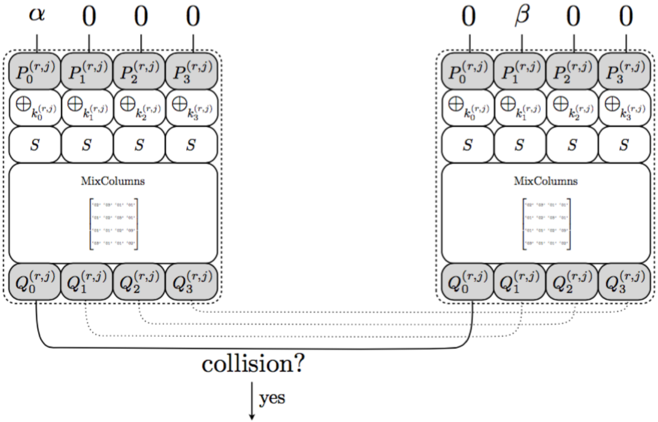


Illustration: J. Muir *"A Tutorial on White-box AES"* (ePrint 2013)

Many breaks

- First break: BGE attack
  - [BGE04] *Cryptanalysis of a White Box AES Implementation* (SAC 2004)
- Generic attack on WB SPN ciphers
  - [MGH08] *Cryptanalysis of a Generic Class of White-Box Implementations* (SAC 2008)
- Collision attack & improved BGE attack
  - [LRD+13] *Two Attacks on a White-Box AES Implementation* (SAC 2013)

Example: collision attack



$$02 \cdot S_0(\alpha) \oplus 03 \cdot S_1(0) = 02 \cdot S_0(0) \oplus 03 \cdot S_1(\beta)$$

where $S_0(x) = S(P_0(x) \oplus k_0)$ and $S_1(x) = S(P_1(x) \oplus k_1)$

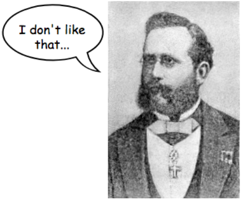
Illustration: Y. De Mulder (presentation SAC 2013)

Patches and variants

- Perturbed WB-AES using MV crypto [BCD06] (ePrint 2006)
  $\Rightarrow$ broken [DWP10] (INDOCRYPT 2010)
- WB-AES based on wide linear encodings [XL09] (CSA 2009)
  $\Rightarrow$ broken [DRP12] (SAC 2012)
- WB-AES based on dual AES ciphers [Kar10] (ICISC 2010)
  $\Rightarrow$ broken [LRD+13] (SAC 2013)
- Same situation with DES

Secret design paradigm

- Industrial need
  - home-made solutions
  - mix of several obfuscation techniques
  - secret designs

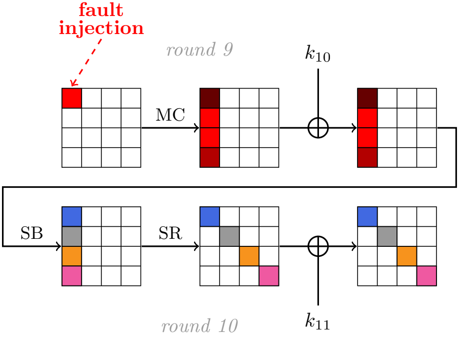


Auguste Kerckhoffs

- Security evaluations by ITSEF labs
- Development of generic “gray-box” attacks
  - Fault attacks, DCA
  - Avoid costly reverse engineering effort

Fault attacks

- Easy fault injection in the white-box context
- Plenty of efficient FA techniques (on *e.g.* AES)

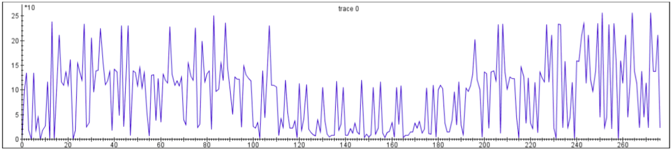


- Original white-box AES vulnerable to this attack

Differential Computation Analysis

- Suggested by NXP / Riscure
  - Presentation at BalckHat 2015
  - Best paper award CHES 2016

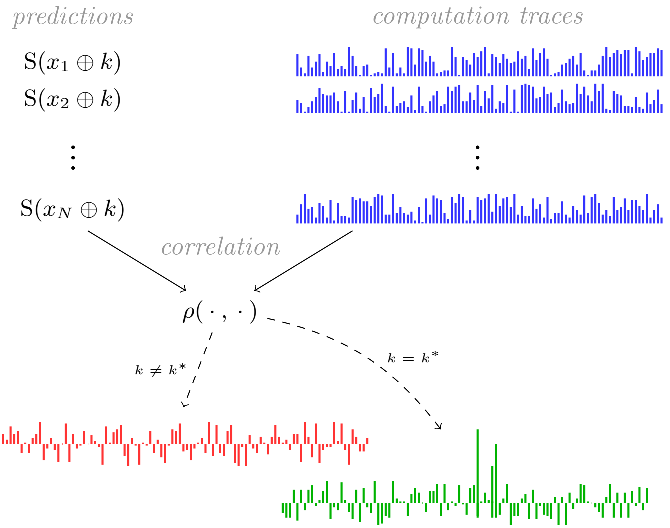
- Record data-dependent information at execution ⇒ *computation trace*



Trace: J. Bos (presentation CHES 2016)

- Apply DPA techniques to computation traces

Differential Computation Analysis

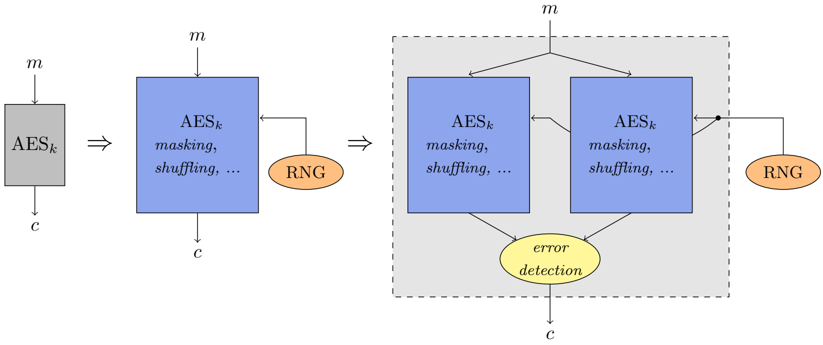


DCA in presence of encodings

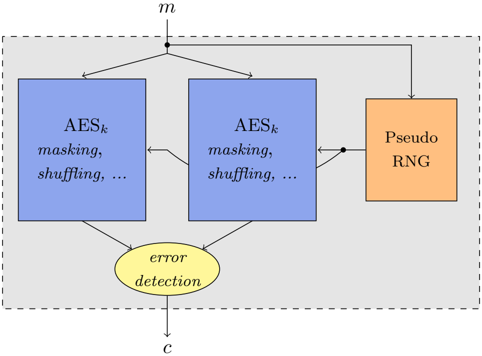
- DCA can break the original white-box AES
  - [BHMT16] *Differential Computation Analysis* (CHES 2016)
- Why?
  - [ABMT18] *On the Ineffectiveness of Internal Encodings* (ACNS 2018)
  - [RW09] *Analysis and Improvement of Differential Computation Attacks against Internally-Encoded White-Box Implementations* (CHES 2019)

Countermeasures?

- Natural approach: use known SCA/FA countermeasures

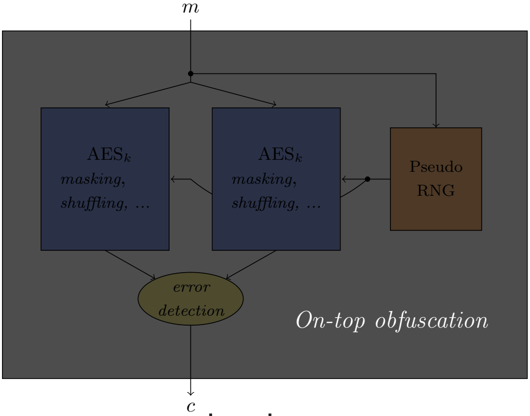


Countermeasures?



- Pseudo-randomness from m
- PRNG should be somehow secret

Countermeasures?



- Countermeasures hard to remove
- Pseudo-randomness / redundancy hard to detect

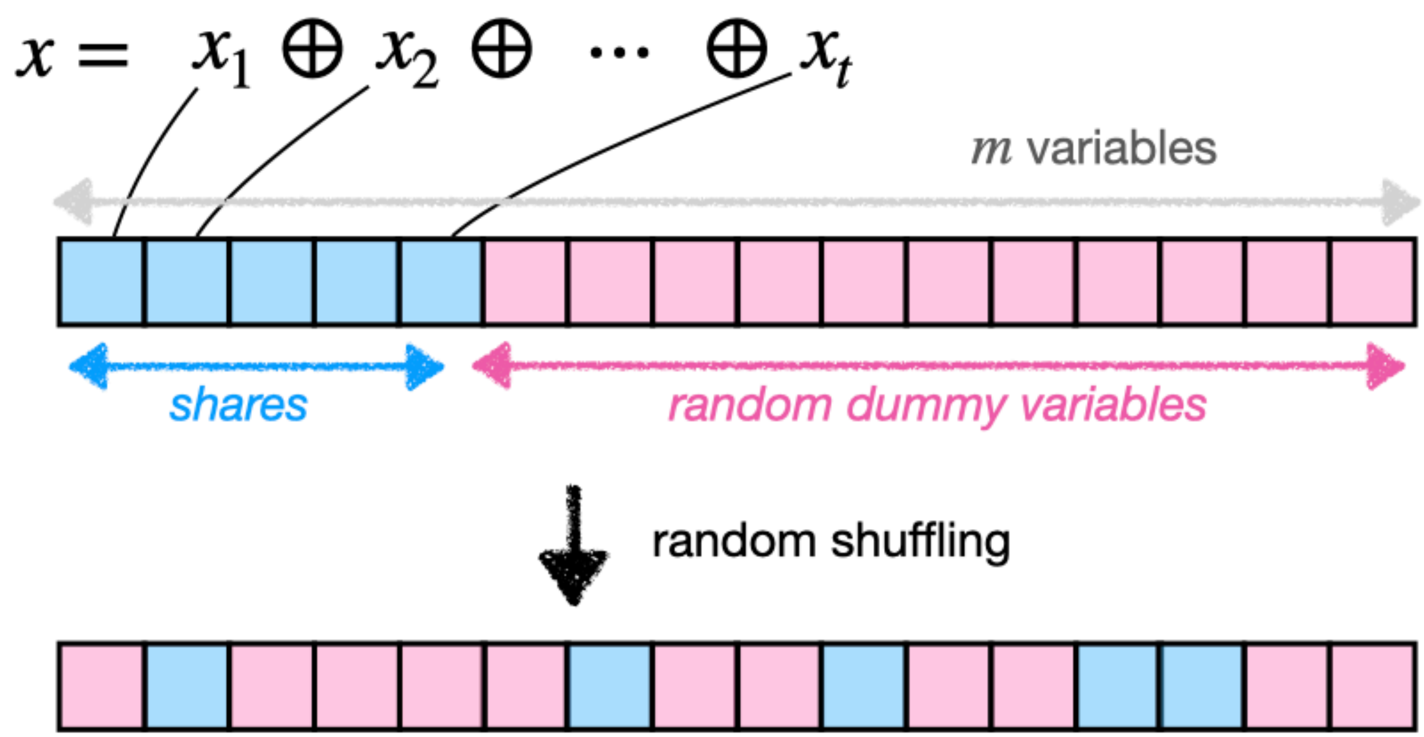
New paradigm: gray-box attacks and countermeasures

Coming next...

- **Case study 1**: masking and shuffling
- WhibOx contest
- **Case study 2**: WhibOx 2017 winner
- Linear Decoding Analysis
- **Case study 3**: WhibOx 2019 winners
- Data Dependency Analysis

Case study 1: masking and shuffling

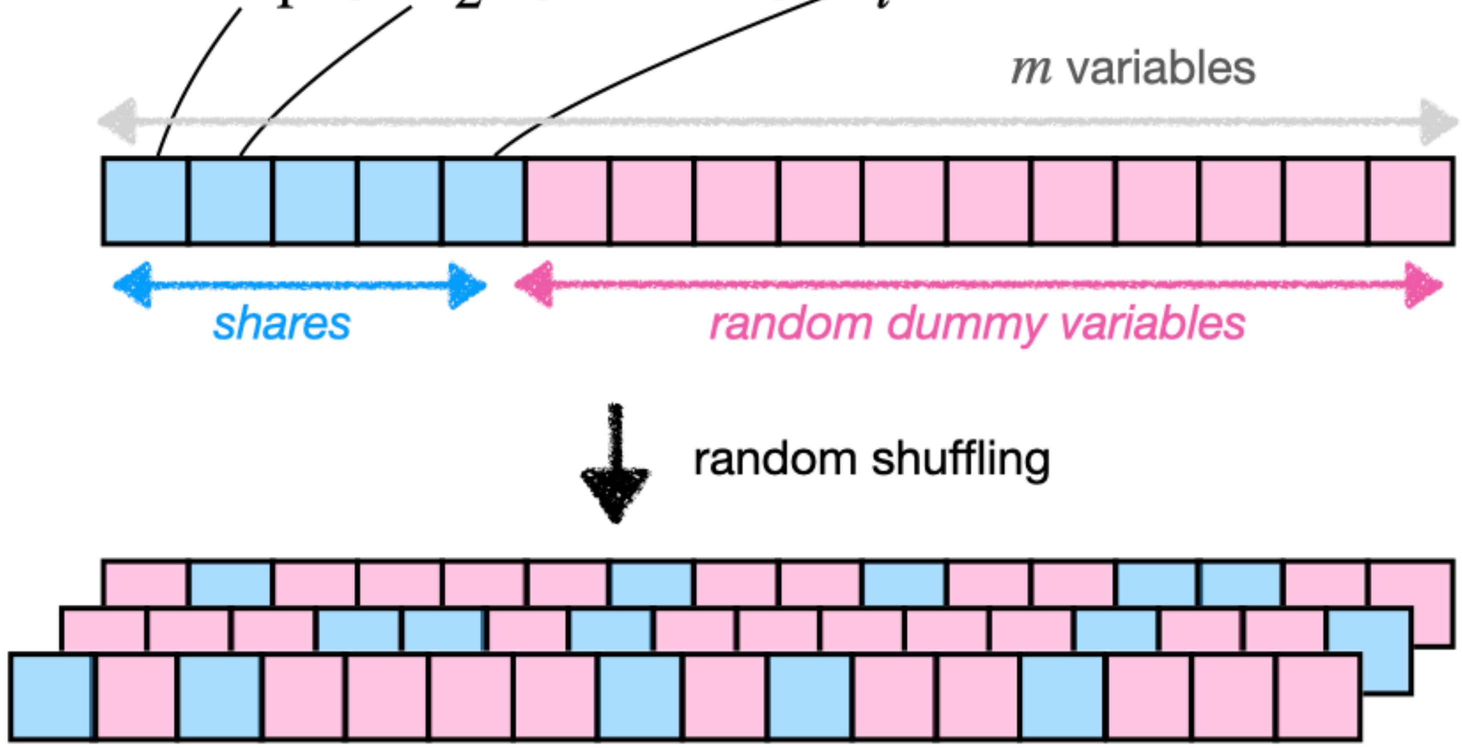
- [BRVW19] *Higher-Order DCA against Standard Side-Channel Countermeasures* (COSADE 2019)



Case study 1: masking and shuffling

- [BRVW19] *Higher-Order DCA against Standard Side-Channel Countermeasures* (COSADE 2019)

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



x ?

Optimal attack complexity

$$\geq \binom{m}{t}$$

Case study 1: masking and shuffling

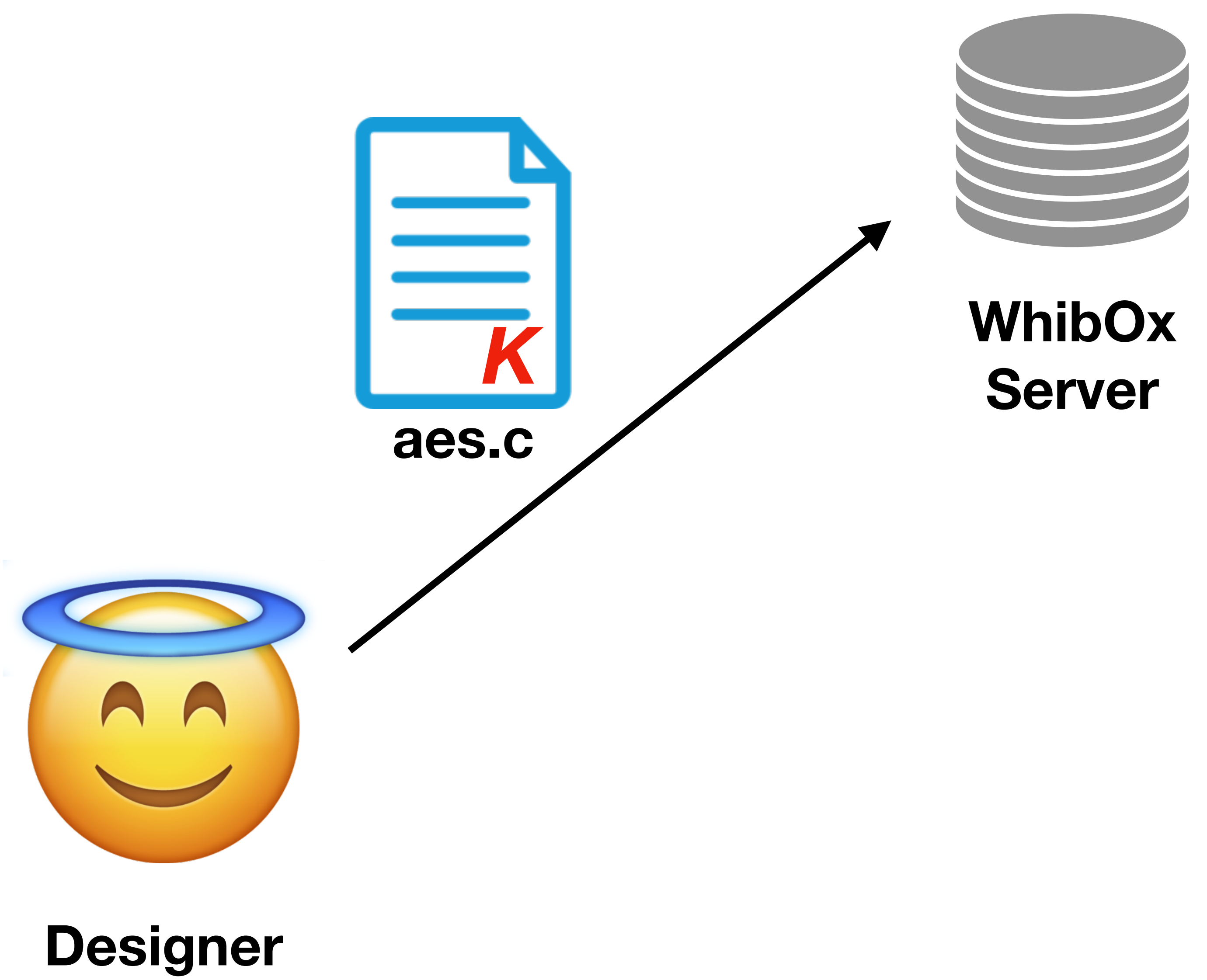
- We obtain exponential security 👍🏻
- But against a limited adversary
    - Passive attack
    - No reverse engineering
- The adversary can do more in the WB model 😈
    - Detect / deactivate shuffling
    - Exploit data dependency
    - Inject faults

WhibOx contets

Goal: confront designers and attackers of practical white-box crypto

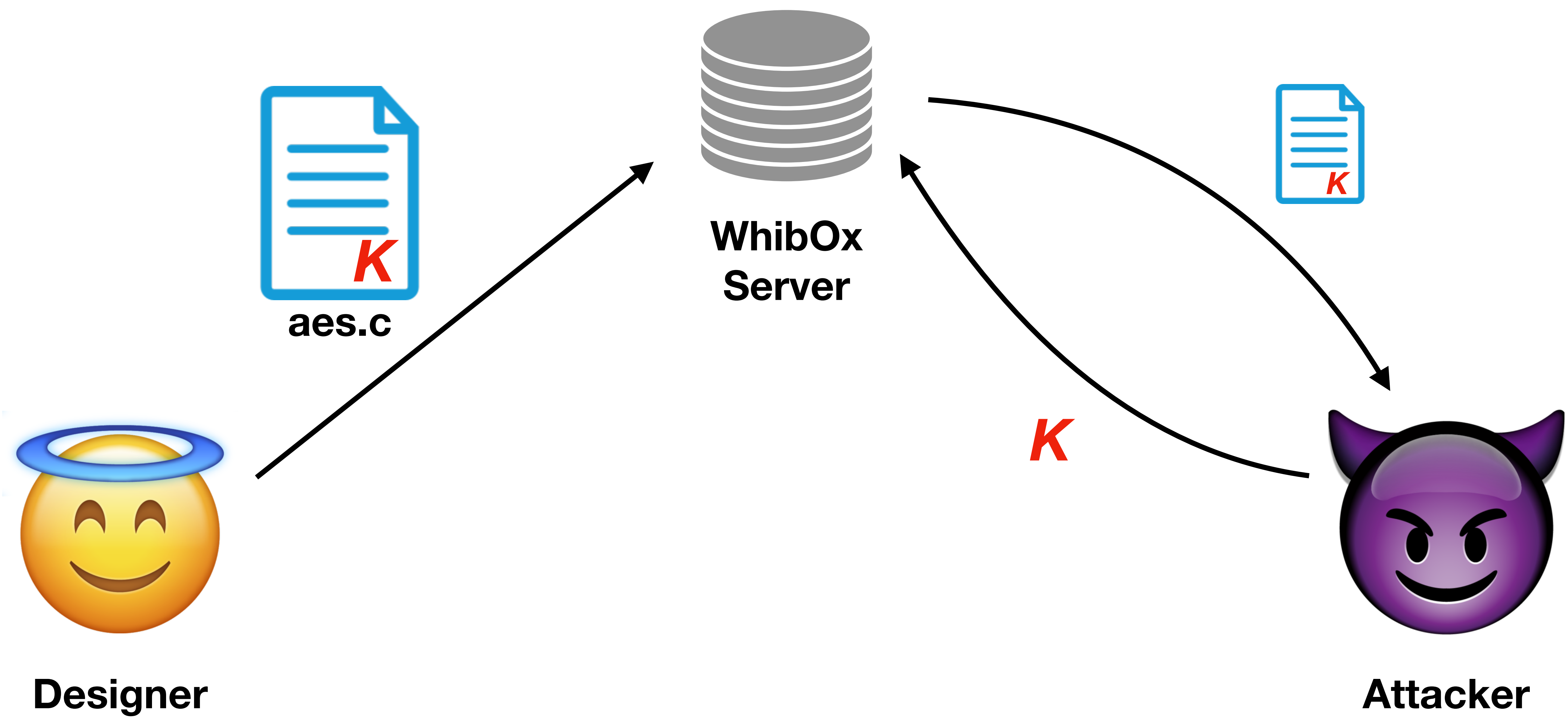
WhibOx contets

Goal: confront designers and attackers of practical white-box crypto



WhibOx contets

Goal: confront designers and attackers of practical white-box crypto



WhibOx contets

- 1st edition (2017)
    - White-box AES (< 20MB / runs < 1s)
    - 94 submitted implementations / 877 breaks
    - Everything broken / winner survived **29 days**

WhibOx contets

- 1st edition (2017)
    - White-box AES (< 20MB / runs < 1s)
    - 94 submitted implementations / 877 breaks
    - Everything broken / winner survived **29 days**
- 2nd edition (2019)
    - White-box AES (< 20MB / runs < 1s)
    - 27 submitted implementations / 124 breaks
    - **3 survivors** (broken few days after deadline)

WhibOx contets

- 1st edition (2017)
    - White-box AES (< 20MB / runs < 1s)
    - 94 submitted implementations / 877 breaks
    - Everything broken / winner survived **29 days**
- 2nd edition (2019)
    - White-box AES (< 20MB / runs < 1s)
    - 27 submitted implementations / 124 breaks
    - **3 survivors** (broken few days after deadline)
- 3rd edition (2021)
    - ECDSA (< 20MB excl. GMP / runs < 3s)
    - 97 submitted implementations / 898 breaks
    - Everything broken is **less than 48h**

WhibOx contets

- 1st edition (2017)
  - White-box AES (< 20MB / runs < 1s)
  - 94 submitted implementations / 877 breaks
  - Everything broken / winner survived **29 days**
- 2nd edition (2019)
  - White-box AES (< 20MB / runs < 1s)
  - 27 submitted implementations / 124 breaks
  - **3 survivors** (broken few days after deadline)
- 3rd edition (2021)
  - ECDSA (< 20MB excl. GMP / runs < 3s)
  - 97 submitted implementations / 898 breaks
  - Everything broken is **less than 48h**
- https://whibox.io/contests/

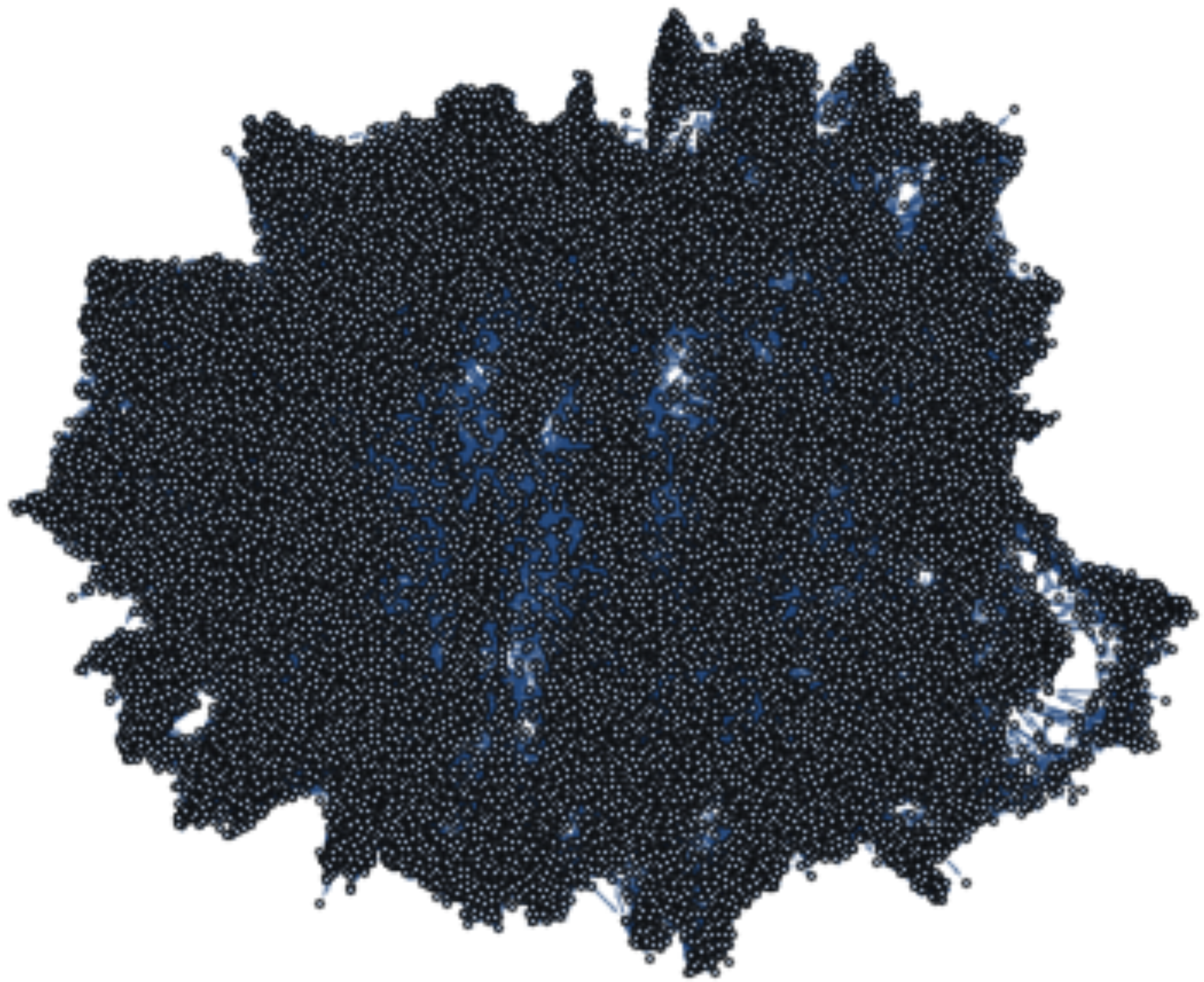
Case study 2: WhibOx 2017 winner

- Winner: challenge #777 (a.k.a. adoring_poitras)
    - From Alex Biryukov, Aleksei Udovenko
    - Boolean level masking, bitslicing, error detection, dummy operations, virtualisation, obfuscation

Case study 2: WhibOx 2017 winner

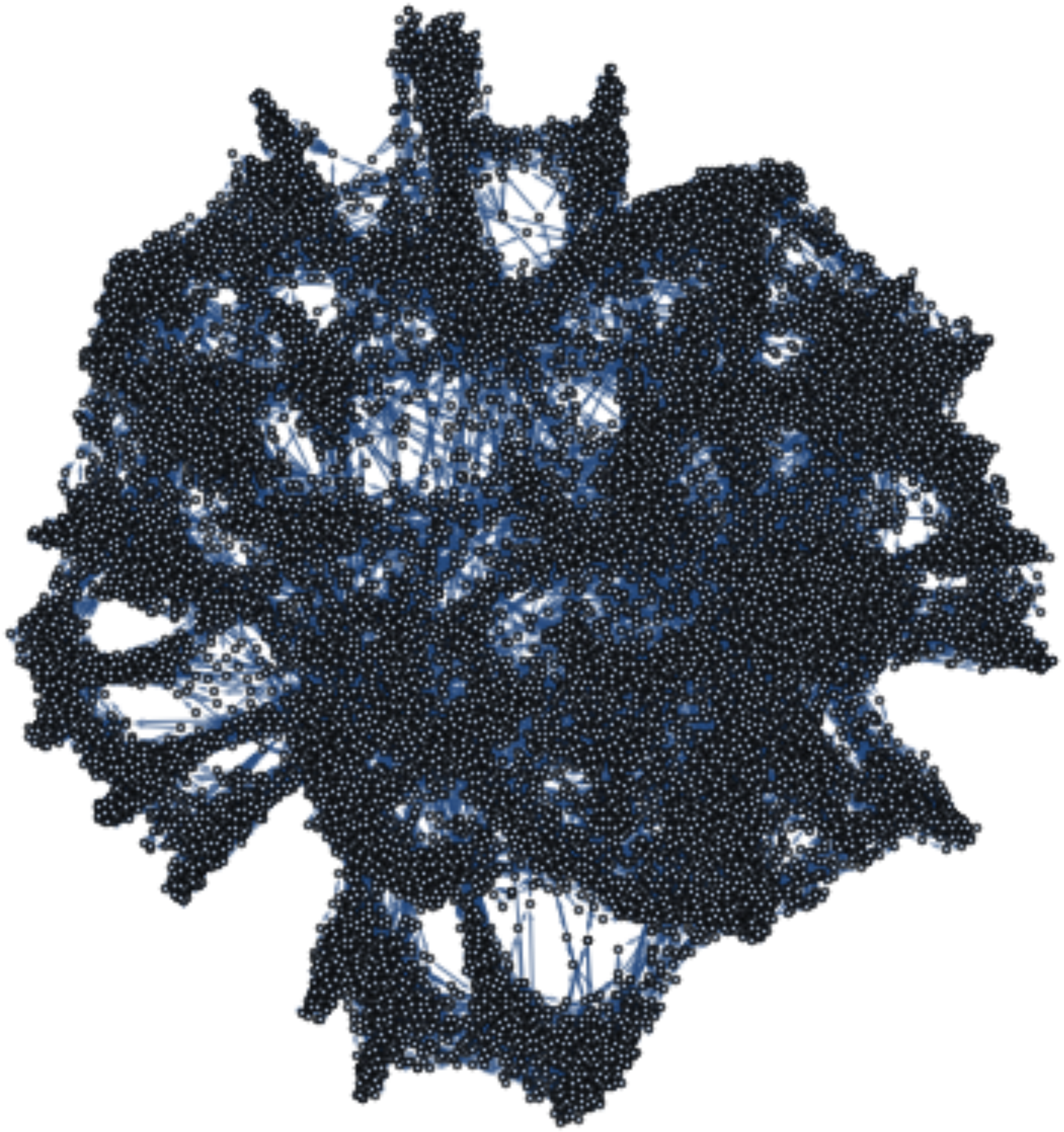
- Winner: challenge #777 (a.k.a. adoring_poitras)
  - From Alex Biryukov, Aleksei Udovenko
  - Boolean level masking, bitslicing, error detection, dummy operations, virtualisation, obfuscation
- Break from Louis Goubin, Pascal Paillier, Matthieu Rivain, Junwei Wang
  - [GPRW18] *How to Reveal the Secrets of an Obscure White-Box Implementation* (ePrint 2018, JCEN 2020)
  - Human reverse engineering ⇒ SSA-format program (circuit)
  - Circuit minimisation (detect dummy / constant / duplicate variables & pseudo-randomness)
    - 600 K gates ⇒ 280 K gates

Data dependency analysis



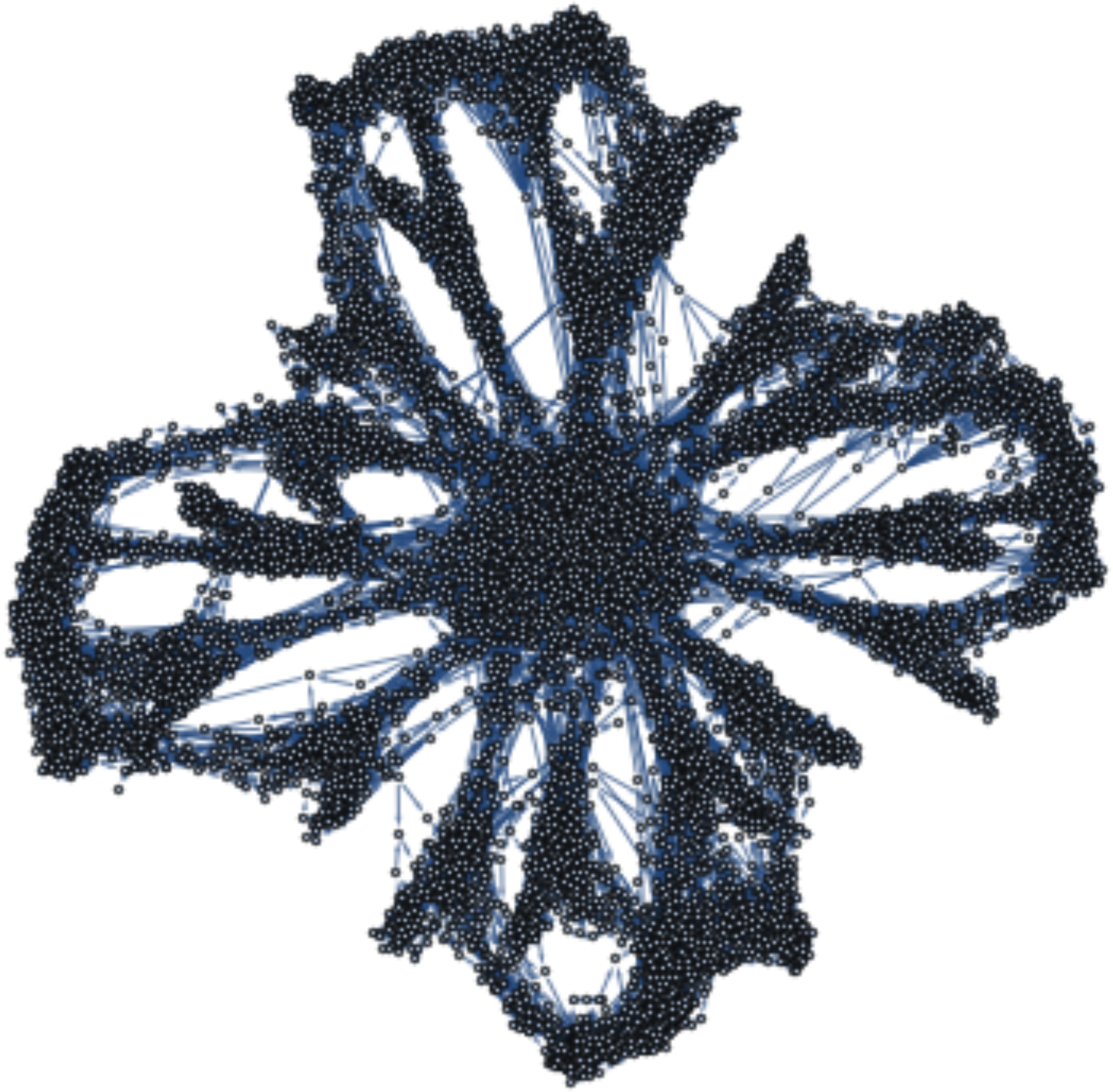
Data dependency graph (20% of the circuit)

Data dependency analysis



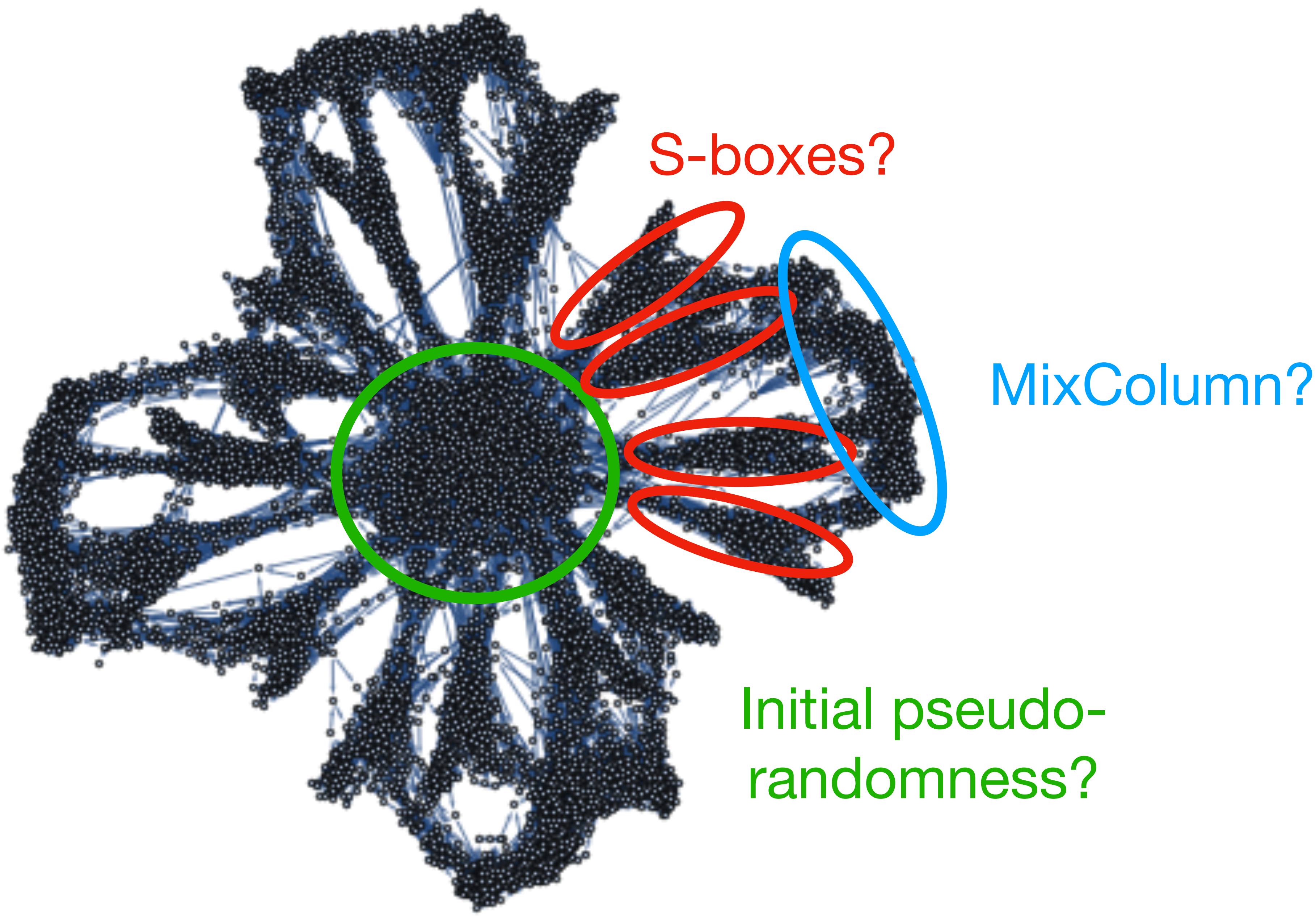
Data dependency graph (10% of the circuit)

Data dependency analysis



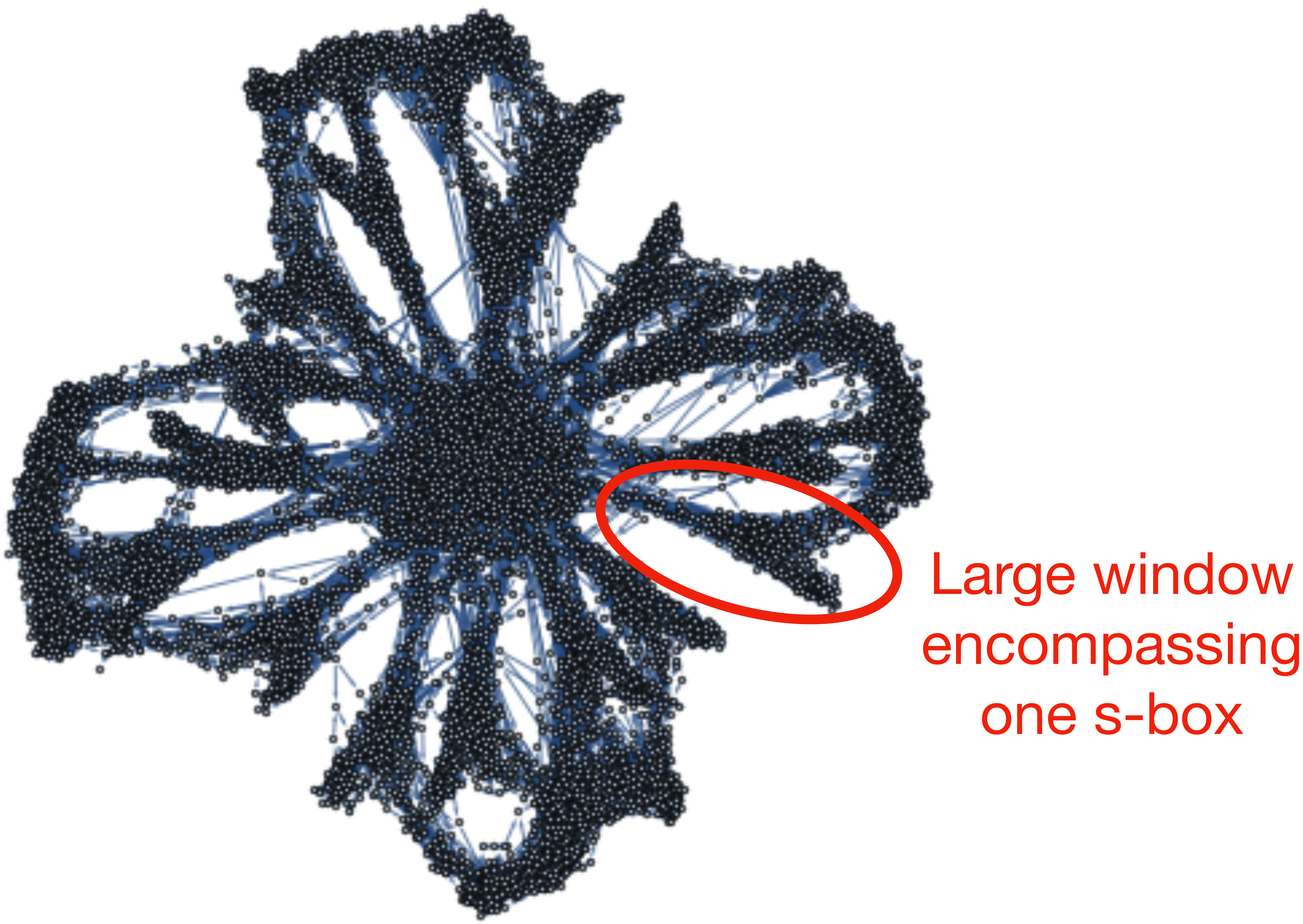
Data dependency graph (5% of the circuit)

Data dependency analysis

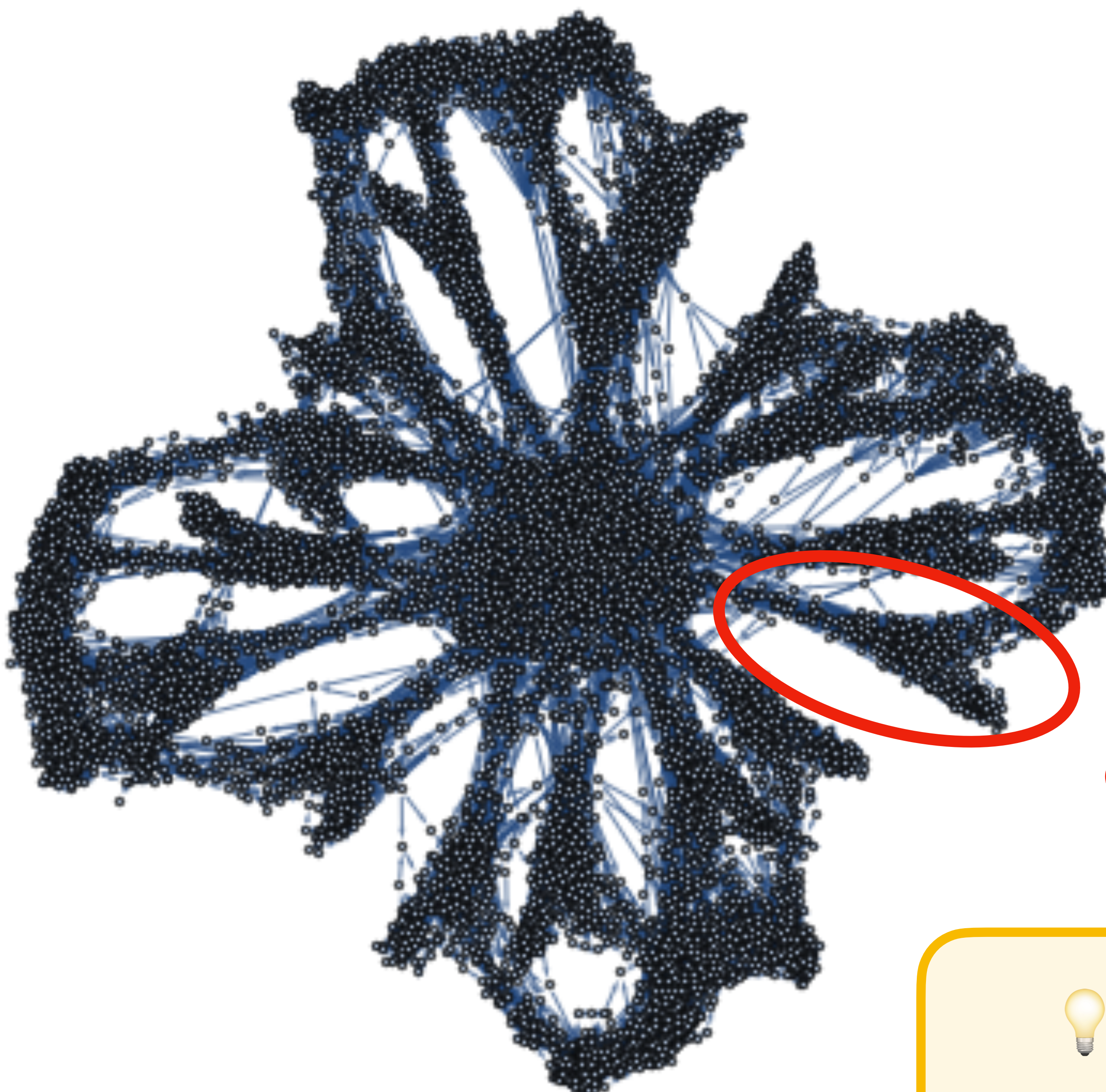


Data dependency graph (5% of the circuit)

Data dependency analysis



Data dependency analysis



Large window encompassing one s-box

💡 Assumption: contains variables encoding $S(x \oplus k)$

Linear Decoding Analysis

- Let $s_1, \ldots, s_m$ the variables in the window
- Record them for n executions

$$\begin{bmatrix} s_1^{(1)} & s_2^{(1)} & \cdots & s_m^{(1)} \\ s_1^{(2)} & s_2^{(2)} & \cdots & s_m^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ s_1^{(n)} & s_2^{(n)} & \cdots & s_m^{(n)} \end{bmatrix}$$

Linear Decoding Analysis

- Let $s_1, \ldots, s_m$ the variables in the window
- Record them for n executions

$$\begin{bmatrix} s_1^{(1)} & s_2^{(1)} & \cdots & s_m^{(1)} \\ s_1^{(2)} & s_2^{(2)} & \cdots & s_m^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ s_1^{(n)} & s_2^{(n)} & \cdots & s_m^{(n)} \end{bmatrix} \times \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} S_j(x^{(1)} \oplus k) \\ S_j(x^{(2)} \oplus k) \\ \vdots \\ S_j(x^{(n)} \oplus k) \end{bmatrix}$$

- 💡 by assumption, we get a linear system

Linear Decoding Analysis

- Let $s_1, \ldots, s_m$ the variables in the window
- Record them for n executions

$$\begin{bmatrix} s_1^{(1)} & s_2^{(1)} & \cdots & s_m^{(1)} \\ s_1^{(2)} & s_2^{(2)} & \cdots & s_m^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ s_1^{(n)} & s_2^{(n)} & \cdots & s_m^{(n)} \end{bmatrix} \times \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} S_j(x^{(1)} \oplus k) \\ S_j(x^{(2)} \oplus k) \\ \vdots \\ S_j(x^{(n)} \oplus k) \end{bmatrix}$$

Plaintext byte

- 💡 by assumption, we get a linear system

Linear Decoding Analysis

- Let $s_1, \ldots, s_m$ the variables in the window
- Record them for n executions

One output bit of the s-box

$$\begin{bmatrix} s_1^{(1)} & s_2^{(1)} & \cdots & s_m^{(1)} \\ s_1^{(2)} & s_2^{(2)} & \cdots & s_m^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ s_1^{(n)} & s_2^{(n)} & \cdots & s_m^{(n)} \end{bmatrix} \times \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} S_j(x^{(1)} \oplus k) \\ S_j(x^{(2)} \oplus k) \\ \vdots \\ S_j(x^{(n)} \oplus k) \end{bmatrix}$$

Plaintext byte

- 💡 by assumption, we get a linear system

Linear Decoding Analysis

- Let $s_1, \ldots, s_m$ the variables in the window
- Record them for n executions

One output bit of the s-box

$$\begin{bmatrix} s_1^{(1)} & s_2^{(1)} & \cdots & s_m^{(1)} \\ s_1^{(2)} & s_2^{(2)} & \cdots & s_m^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ s_1^{(n)} & s_2^{(n)} & \cdots & s_m^{(n)} \end{bmatrix} \times \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} S_j(x^{(1)} \oplus k) \\ S_j(x^{(2)} \oplus k) \\ \vdots \\ S_j(x^{(n)} \oplus k) \end{bmatrix}$$

Unknown key byte

Unknown coefficients

Plaintext byte

- 💡 by assumption, we get a linear system

LDA: generalisation

- LDA defeats WB implems based on additive sharing
- Generalisation to encoding of higher degrees

$$\overrightarrow{v} = (1 \,|\, \vec{s} \,|\, \vec{s} \otimes \vec{s} \,|\, \vec{s} \otimes \vec{s} \otimes \vec{s} \,|\, \ldots)$$

LDA: generalisation

- LDA defeats WB implems based on additive sharing
- Generalisation to encoding of higher degrees

$$\overrightarrow{v} = (1 \mid \vec{s} \mid \underbrace{\vec{s} \otimes \vec{s}}_{\text{degree-2 monomials}} \mid \underbrace{\vec{s} \otimes \vec{s} \otimes \vec{s}}_{\text{degree-3 monomials}} \mid \underbrace{\ldots}_{\text{etc.}})$$

LDA: generalisation

- LDA defeats WB implems based on additive sharing
- Generalisation to encoding of higher degrees

$$\overrightarrow{v} = (1 \,|\, \vec{s} \,|\, \vec{s} \otimes \vec{s} \,|\, \vec{s} \otimes \vec{s} \otimes \vec{s} \,|\, \ldots)$$

degree-2 monomials

degree-3 monomials

etc.

- Larger system

$$\begin{bmatrix} v_1^{(1)} & v_2^{(1)} & \cdots & v_{m'}^{(1)} \\ v_1^{(2)} & v_2^{(2)} & \cdots & v_{m'}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ v_1^{(n)} & v_2^{(n)} & \cdots & v_{m'}^{(n)} \end{bmatrix} \times \begin{bmatrix} c_1 \\ \vdots \\ c_2 \\ \vdots \\ \vdots \\ \vdots \\ c_{m'} \end{bmatrix} = \begin{bmatrix} f(x^{(1)}, k) \\ f(x^{(2)}, k) \\ \vdots \\ f(x^{(n)}, k) \end{bmatrix}$$

LDA: generalisation

- LDA defeats WB implems based on additive sharing
- Generalisation to encoding of higher degrees

$$\overrightarrow{v} = (1 \mid \vec{s} \mid \underbrace{\vec{s} \otimes \vec{s}}_{\text{degree-2 monomials}} \mid \underbrace{\vec{s} \otimes \vec{s} \otimes \vec{s}}_{\text{degree-3 monomials}} \mid \underbrace{\ldots}_{\text{etc.}})$$

- Larger system

$$\begin{bmatrix} v_1^{(1)} & v_2^{(1)} & \cdots & v_{m'}^{(1)} \\ v_1^{(2)} & v_2^{(2)} & \cdots & v_{m'}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ v_1^{(n)} & v_2^{(n)} & \cdots & v_{m'}^{(n)} \end{bmatrix} \times \begin{bmatrix} c_1 \\ \vdots \\ c_2 \\ \vdots \\ \vdots \\ c_{m'} \end{bmatrix} = \begin{bmatrix} f(x^{(1)}, k) \\ f(x^{(2)}, k) \\ \vdots \\ f(x^{(n)}, k) \end{bmatrix}$$

Complexity:

Inverting a $m^d \times m^d$ matrix

$\Rightarrow \mathcal{O}(m^{2.8d})$

LDA: mitigation

- Non-linear masking

$$x = x_1 \cdot x_2 \oplus x_3$$

  - [BU18] *Attacks and Countermeasures for White-box Designs* (ASIACRYPT 2018)
  - [SEL21] *A White-Box Masking Scheme Resisting Computational and Algebraic Attacks* (CHES 202)

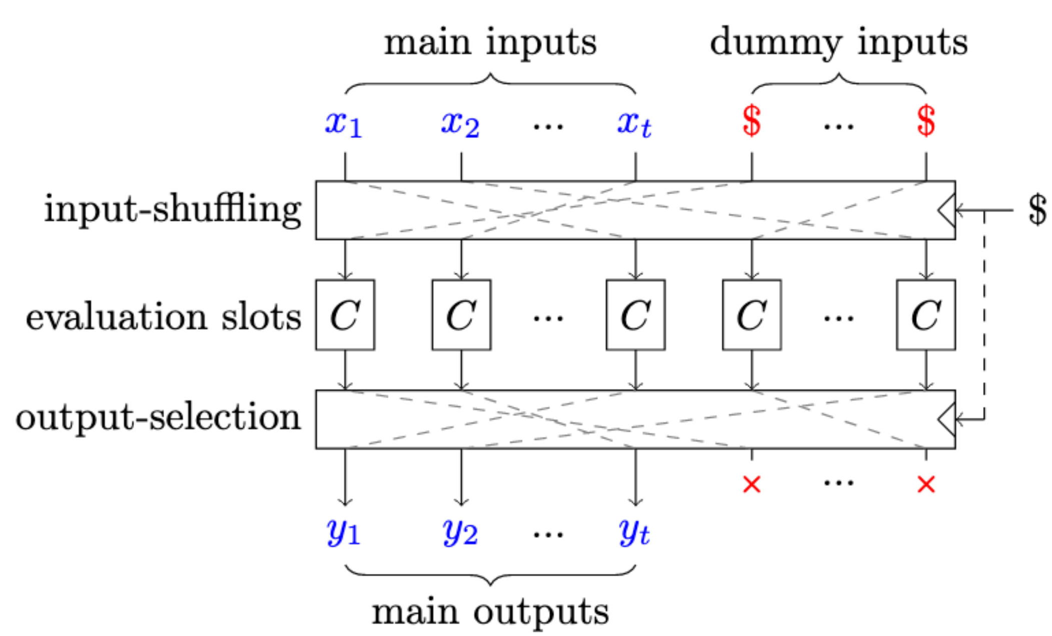
LDA: mitigation

- Non-linear masking

$$x = x_1 \cdot x_2 \oplus x_3$$

  - [BU18] *Attacks and Countermeasures for White-box Designs* (ASIACRYPT 2018)
  - [SEL21] *A White-Box Masking Scheme Resisting Computational and Algebraic Attacks* (CHES 202)

- Dummy shuffling



  - [BU21] *Dummy Shuffling against Algebraic Attacks in White-box Implementations* (EUROCRYPT 2021)

Case study 3: WhibOx 2019 winners

- Winners: challenges #100, #111, #115
  - From Alex Biryukov, Aleksei Udovenko
  - Linear (high-order) masking, non-linear masking, shuffling, obfuscation, virtualisation
- Breaks from Louis Goubin, Matthieu Rivain, Junwei Wang / Arnolds Kikusts, Artur Pchelkin
  - [GRW20] *Defeating State-of-the-Art White-Box Countermeasures with Advanced Gray-Box Attacks* (CHES 2020)
  - Human reverse engineering ⇒ SSA-format program

Higher-order DCA

- From a trace / window $s_1, \ldots, s_m$ compute

$$s_{i_1} \oplus s_{i_2} \oplus \cdots \oplus s_{t_t} \quad \forall \ \ 1 \leq i_1 < i_2 < \cdots < i_t \leq n$$

$\Rightarrow$ t-th order trace

Higher-order DCA

- From a trace / window $s_1, \ldots, s_m$ compute

$$s_{i_1} \oplus s_{i_2} \oplus \cdots \oplus s_{t_t} \quad \forall \;\; 1 \leq i_1 < i_2 < \cdots < i_t \leq n$$

$\Rightarrow$ t-th order trace

- Against t-order masking + non-linear masking

$$x = a \cdot b \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_t$$

$$\Rightarrow \text{Cor}(x, x_1 \oplus \cdots \oplus x_t) = \frac{1}{2}$$

Higher-order DCA

- From a trace / window $s_1, \ldots, s_m$ compute

$$s_{i_1} \oplus s_{i_2} \oplus \cdots \oplus s_{i_t} \quad \forall \;\; 1 \le i_1 < i_2 < \cdots < i_t \le n$$

$\Rightarrow$ t-th order trace

- Against t-order masking + non-linear masking

$$x = a \cdot b \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_t$$

$$\Rightarrow \text{Cor}(x, x_1 \oplus \cdots \oplus x_t) = \frac{1}{2}$$

- Against t-order masking + non-linear masking + λ-shuffling

$$\text{HO-DCA} \Rightarrow \text{Cor} = \frac{1}{2\lambda}$$

$$\text{Integrated HO-DCA} \Rightarrow \text{Cor} = \frac{1}{2\sqrt{\lambda}}$$

Higher-order DCA

- From a trace / window $s_1, \ldots, s_m$ compute

$$s_{i_1} \oplus s_{i_2} \oplus \cdots \oplus s_{i_t} \quad \forall \;\; 1 \leq i_1 < i_2 < \cdots < i_t \leq n$$

$\Rightarrow$ t-th order trace

- Against t-order masking + non-linear masking

$$x = a \cdot b \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_t$$

$$\Rightarrow \text{Cor}(x, x_1 \oplus \cdots \oplus x_t) = \frac{1}{2}$$

- Against t-order masking + non-linear masking + λ-shuffling

$$\text{HO-DCA} \Rightarrow \text{Cor} = \frac{1}{2\lambda}$$

$$\text{Integrated HO-DCA} \Rightarrow \text{Cor} = \frac{1}{2\sqrt{\lambda}}$$

impact of non-linear masking

impact of shuffling

Higher-order DCA

- From a trace / window $s_1, \ldots, s_m$ compute

$$s_{i_1} \oplus s_{i_2} \oplus \cdots \oplus s_{i_t} \quad \forall \;\; 1 \leq i_1 < i_2 < \cdots < i_t \leq n$$

$\Rightarrow$ t-th order trace

impact of non-linear masking

- Against t-order masking + non-linear masking

$$x = a \cdot b \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_t$$

$$\Rightarrow \text{Cor}(x, x_1 \oplus \cdots \oplus x_t) = \frac{1}{2}$$

impact of shuffling

- Against t-order masking + non-linear masking + λ-shuffling

$$\text{HO-DCA} \Rightarrow \text{Cor} = \frac{1}{2\lambda}$$

$$\text{Integrated HO-DCA} \Rightarrow \text{Cor} = \frac{1}{2\sqrt{\lambda}}$$

$\Rightarrow$ **Number of traces $\mathcal{O}(\lambda)$**

Higher-order DCA

- From a trace / window $s_1, \dots, s_m$ compute

$$s_{i_1} \oplus s_{i_2} \oplus \cdots \oplus s_{i_t} \quad \forall \;\; 1 \leq i_1 < i_2 < \cdots < i_t \leq n$$

$\Rightarrow$ t-th order trace

impact of non-linear masking

- Against t-order masking + non-linear masking

$$x = a \cdot b \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_t$$

$$\Rightarrow \text{Cor}(x, x_1 \oplus \cdots \oplus x_t) = \frac{1}{2}$$

impact of shuffling

- Against t-order masking + non-linear masking + λ-shuffling

$$\text{HO-DCA} \Rightarrow \text{Cor} = \frac{1}{2\lambda}$$

$$\text{Integrated HO-DCA} \Rightarrow \text{Cor} = \frac{1}{2\sqrt{\lambda}}$$

$\Rightarrow$ **Number of traces $\mathcal{O}(\lambda)$**

- Size of t-th order trace: $\binom{m}{t}$

Higher-order DCA

- From a trace / window $s_1, \ldots, s_m$ compute

$$s_{i_1} \oplus s_{i_2} \oplus \cdots \oplus s_{i_t} \quad \forall \;\; 1 \leq i_1 < i_2 < \cdots < i_t \leq n$$

$\Rightarrow$ t-th order trace

- Against t-order masking + non-linear masking

$$x = a \cdot b \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_t$$

$$\Rightarrow \text{Cor}(x, x_1 \oplus \cdots \oplus x_t) = \frac{1}{2}$$

- Against t-order masking + non-linear masking + λ-shuffling

$$\text{HO-DCA} \Rightarrow \text{Cor} = \frac{1}{2\lambda}$$

$$\text{Integrated HO-DCA} \Rightarrow \text{Cor} = \frac{1}{2\lambda/\lambda}$$

impact of non-linear masking

impact of shuffling

$\Rightarrow$ **Number of traces $\mathcal{O}(\lambda)$**

- Size of t-th order trace: $\binom{m}{t}$

window size

masking order

$\Rightarrow$ **Size $\mathcal{O}(m^t)$**

How to reduce the window size?

- 💡 idea: exploit the locality of a masking gadget

How to reduce the window size?

- 💡 idea: exploit the locality of a masking gadget
- Multiplication gadget $(x_1, \ldots, x_t) \otimes (y_1, \ldots, y_t)$

$$\begin{pmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_t \\ x_2y_1 & x_2y_2 & \cdots & x_2y_t \\ \vdots & \vdots & \ddots & \vdots \\ x_ty_1 & x_ty_2 & \cdots & x_ty_t \end{pmatrix} + \text{randomness} \rightarrow \sum \rightarrow (z_1, \ldots, z_t)$$

How to reduce the window size?

- 💡 idea: exploit the locality of a masking gadget
- Multiplication gadget $(x_1, \ldots, x_t) \otimes (y_1, \ldots, y_t)$

$$\begin{pmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_t \\ x_2y_1 & x_2y_2 & \cdots & x_2y_t \\ \vdots & \vdots & \ddots & \vdots \\ x_ty_1 & x_ty_2 & \cdots & x_ty_t \end{pmatrix} + \text{randomness} \rightarrow \sum \rightarrow (z_1, \ldots, z_t)$$

- Set of co-operands of any $x_i \Rightarrow$ all the shares $y_1, \ldots, y_n$

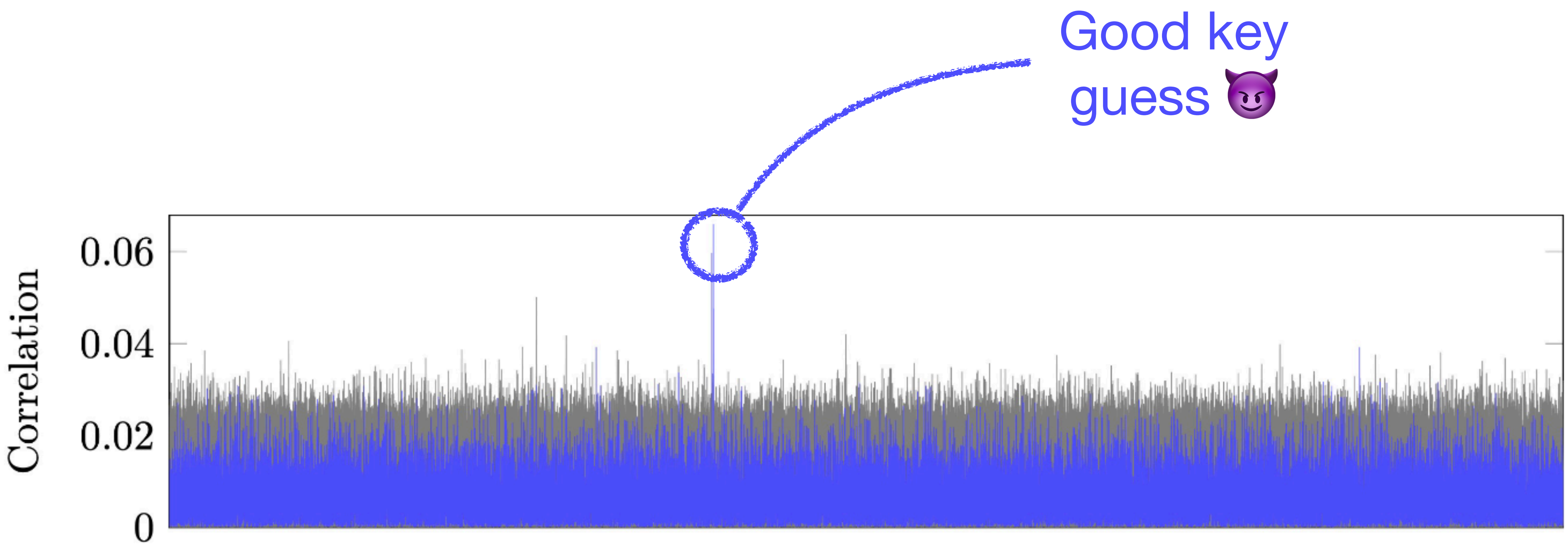
How to reduce the window size?

- 💡 idea: exploit the locality of a masking gadget
- Multiplication gadget $(x_1, \ldots, x_t) \otimes (y_1, \ldots, y_t)$

$$\begin{pmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_t \\ x_2y_1 & x_2y_2 & \cdots & x_2y_t \\ \vdots & \vdots & \ddots & \vdots \\ x_ty_1 & x_ty_2 & \cdots & x_ty_t \end{pmatrix} + \text{randomness} \to \sum \to (z_1, \ldots, z_t)$$

- Set of co-operands of any $x_i \Rightarrow$ all the shares $y_1, \ldots, y_n$
- Data-dependency HO-DCA
  - Scanning all the gates of the circuit
  - For each gate g : $\vec{s}_g = \text{CoOperands}(g)$ *(might contain t shares)*
  - Global t-th order trace = $\Big($ t-th order trace ($\vec{s}_g$) $\Big)_{\forall g}$
  - Apply DCA to global t-th order traces

Data-dependency HO-DCA on #100



Data-dependency HO-DCA against #100, using 767 traces
(18% of the circuit / target: 1st round s-box)

Data-dependency analysis

- Clustering technique applicable to any gray-box attack in the white-box setting
- Principle
  - Scan the gates of the circuit / DD graph
  - For each g, record co-operands of g as potential window
  - Apply a given gray-box attack to the recorded windows
- Possible extensions
  - Include co-operands of degree d
    (co-op. of co-op. of co-op. …)
  - Include incoming / outgoing gates

Conclusion

- Strong WBC (VBB / UBK) hard to achieve in practice
- Practical WBC relies on security through obscurity
  $\Rightarrow$ countermeasures & obfuscation vs. gray-box attacks
- Exponential security can be obtained against some attacks
  $\Rightarrow$ attack window must be large enough
- DDA very effective to reduce the attack window
  - Open problem: how to thwart DDA attacks?
- Fault attacks: to be formalised / investigated more in WB setting
- WhibOx 2021 on ECDSA $\Rightarrow$ WB session next Tuesday