Haystack ciphers: White-box countermeasures as Symmetric encryption

Alex Charlès[1], Aleksei Udovenko[2]

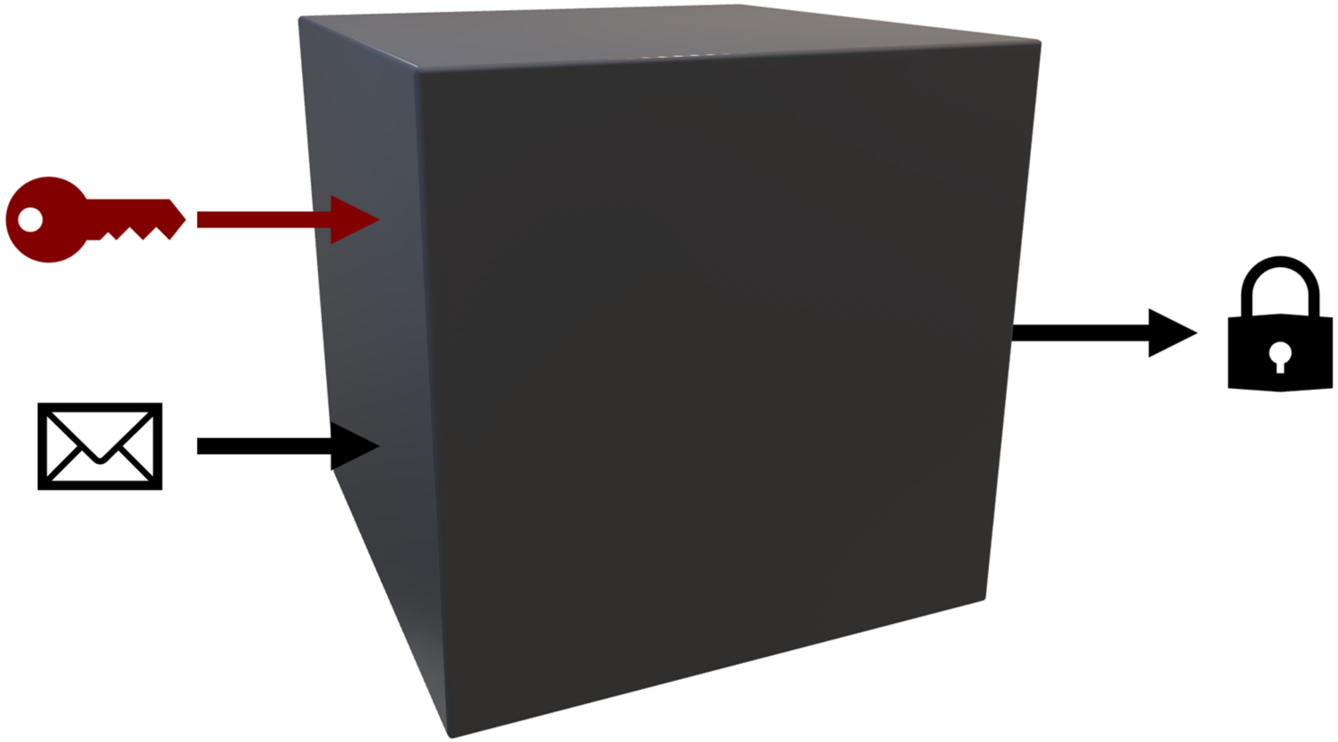
1) DCS, University of Luxembourg : alex.charles205@gmail.com

2) SnT, University of Luxembourg : aleksei@affine.group

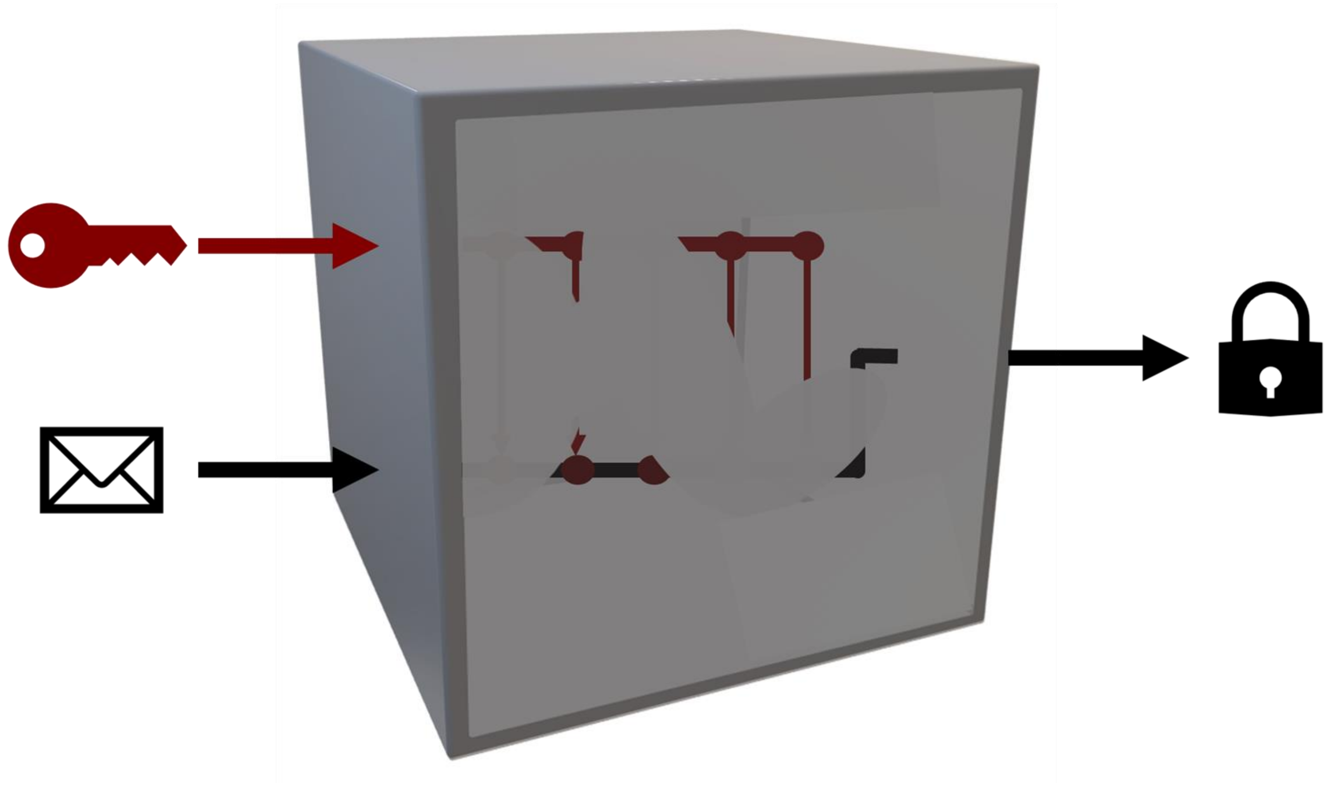
Journée Thématique sur mes Attaques par Injection de Faute 2025

General overview of White-box Cryptography

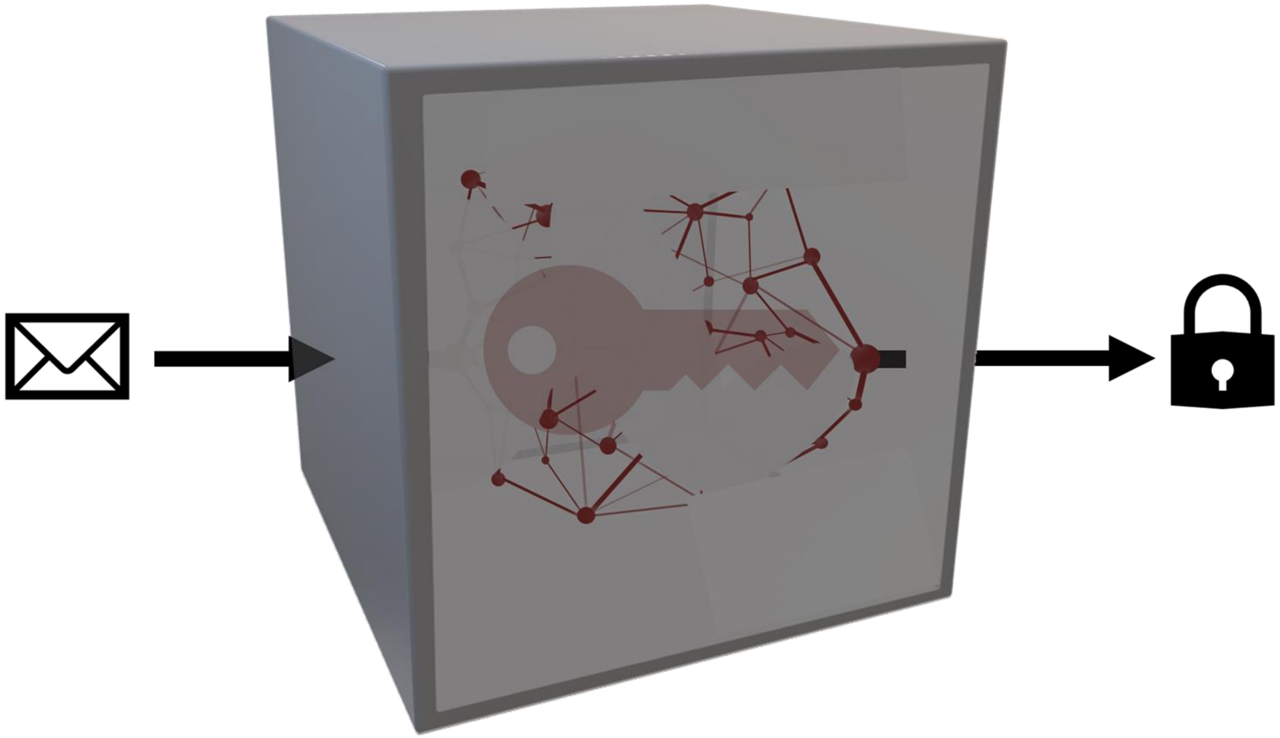
Black-box model



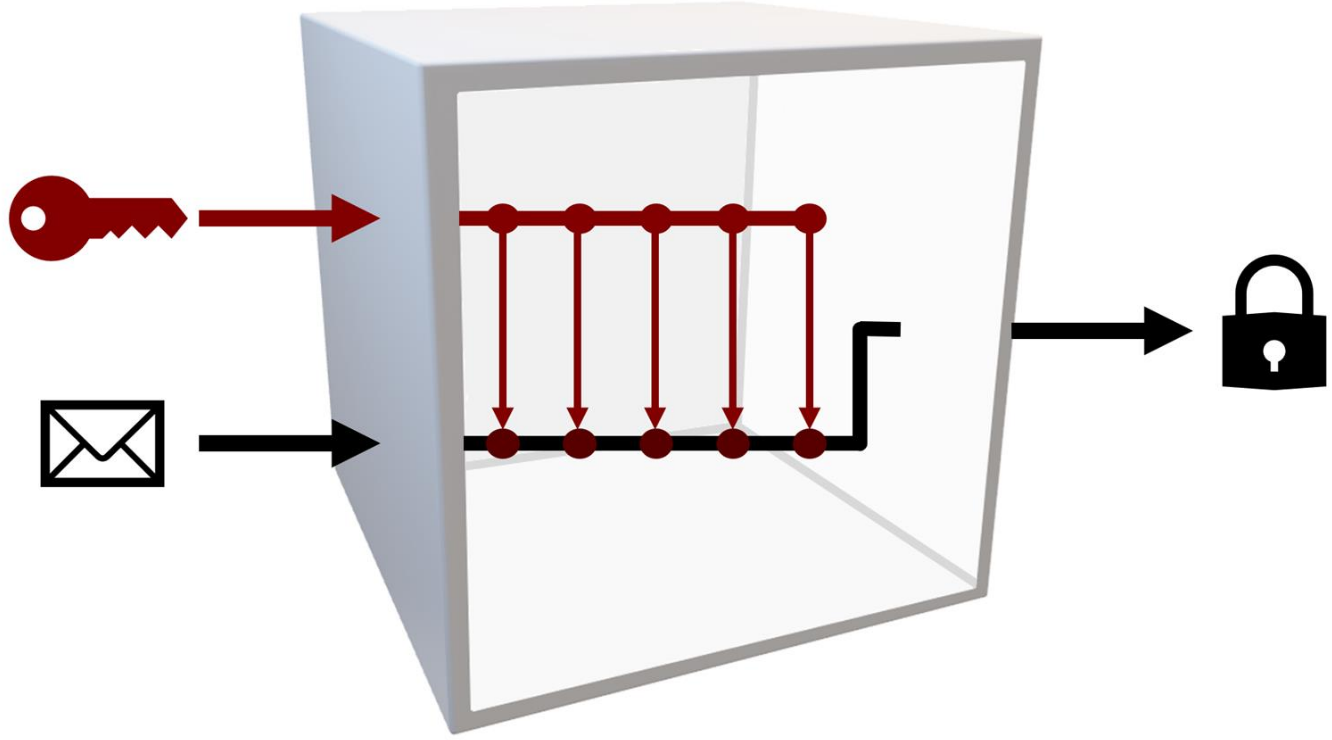
Grey-box model



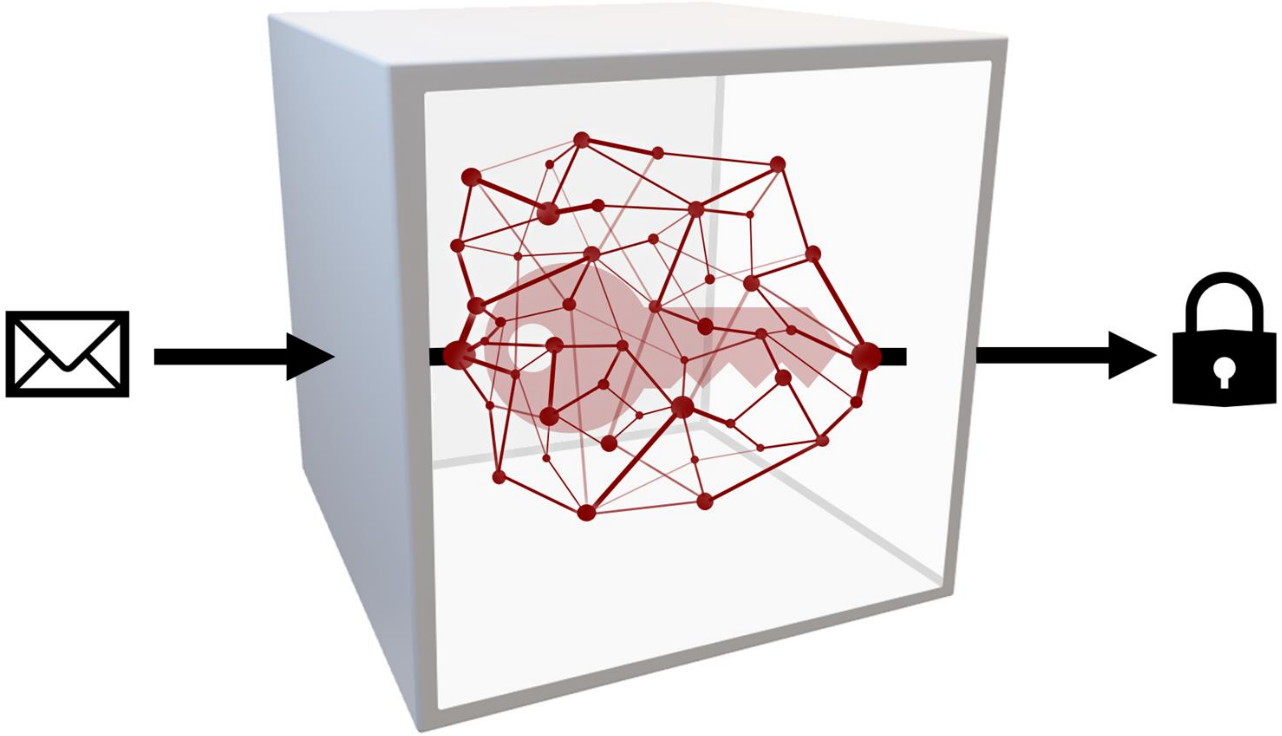
Grey-box model



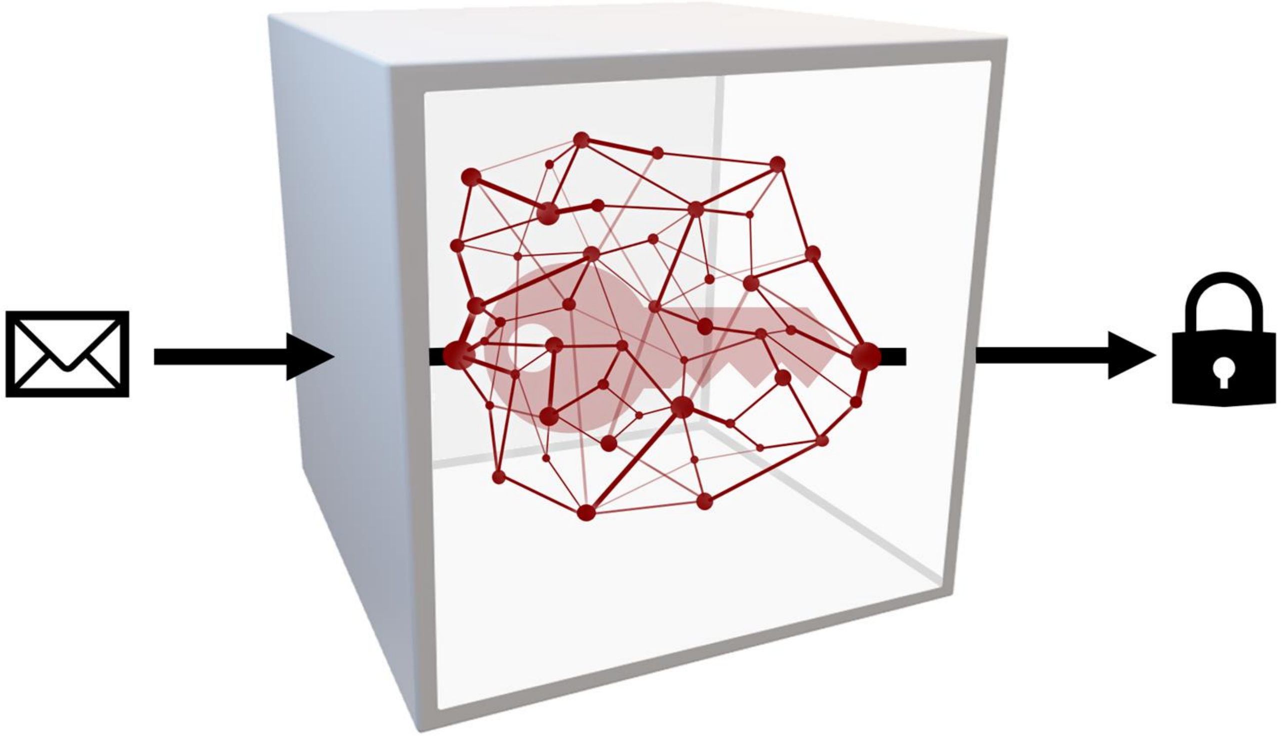
White-box model



White-box model



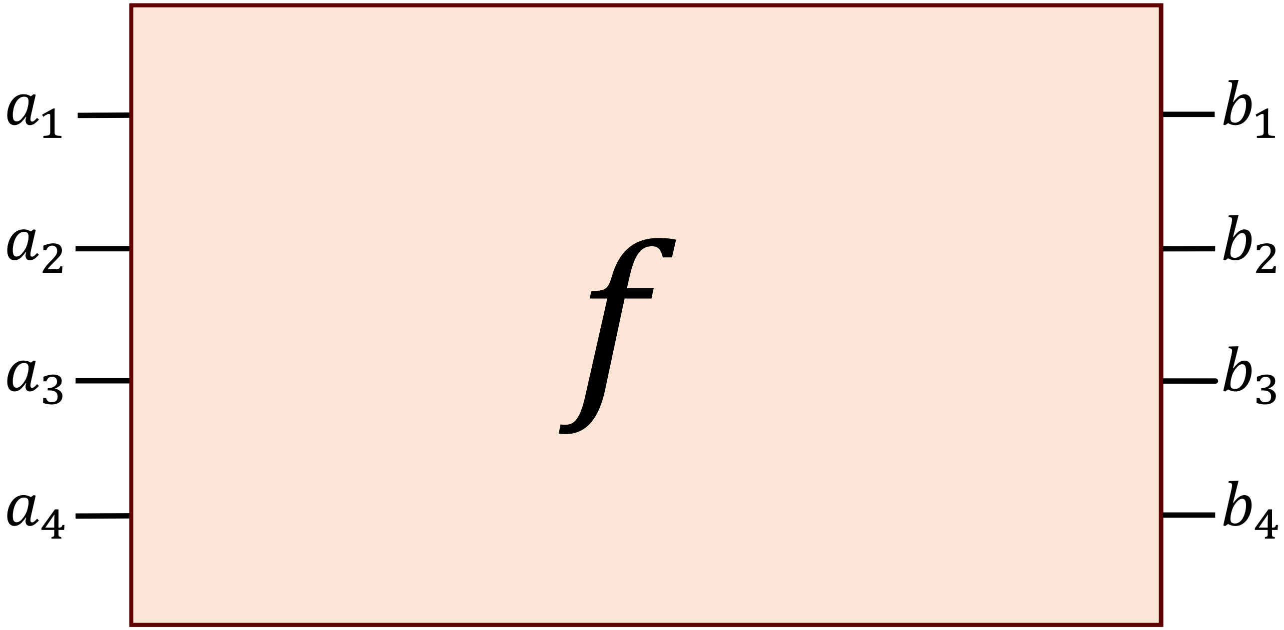
White-box model



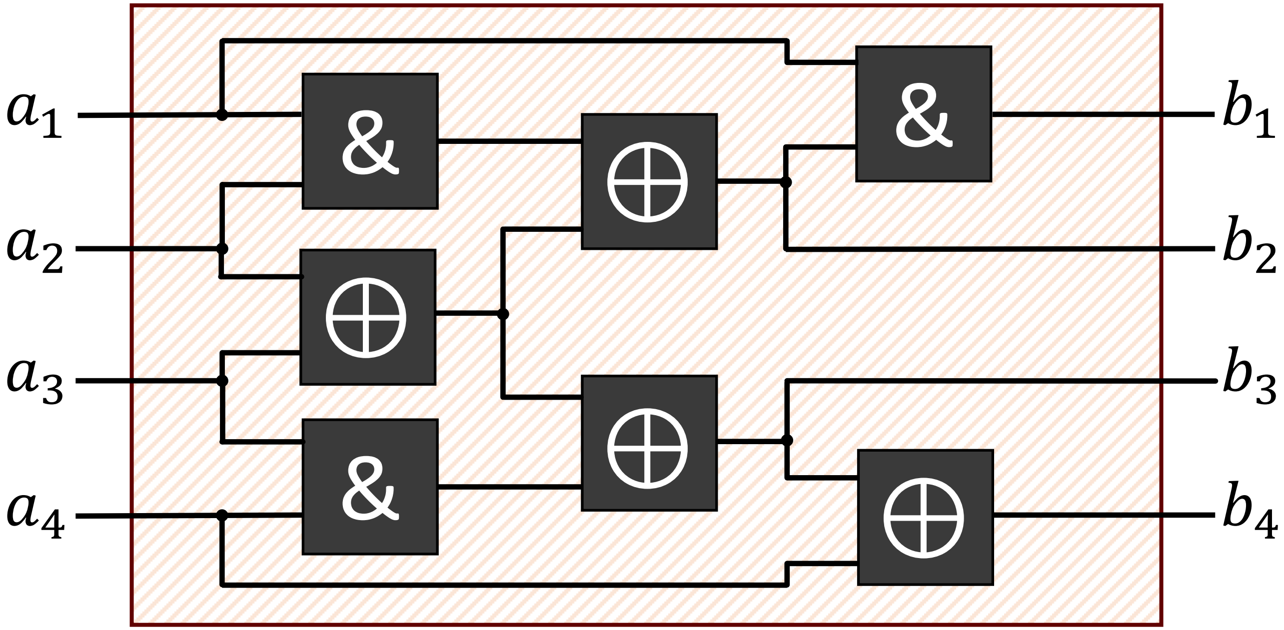
- Controlled environment
  - The only input is the plaintext
  - The implementation is deterministic
- Access to noiseless traces
  - Possibility of algebraic attacks
- All bit gates are available
  - Structure analysis of the implementation
- Cheap bit-precise fault
  - Multi-fault is possible

Side channel attacks in the white-box model

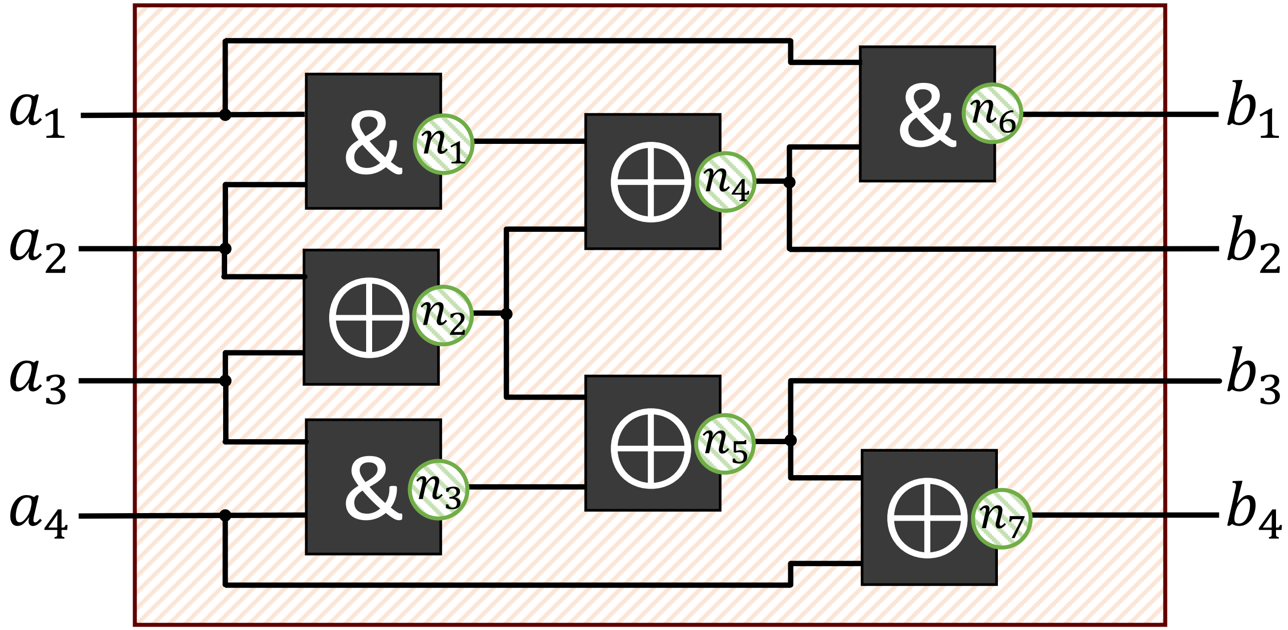
Traces generation



Traces generation

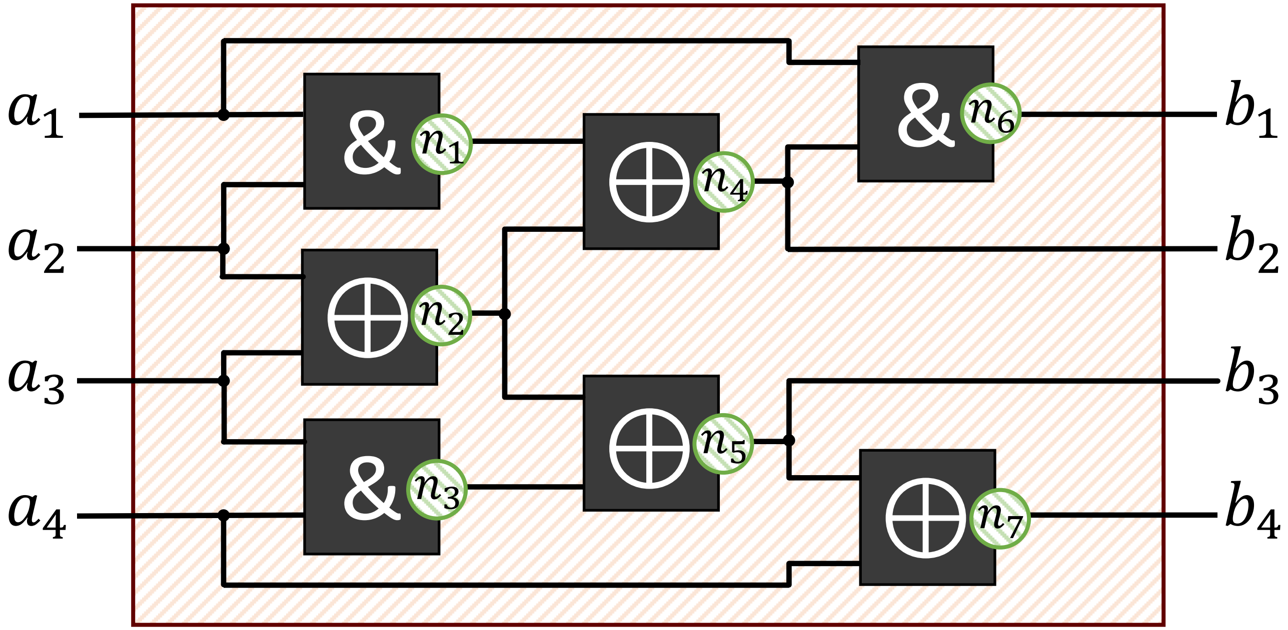


Traces generation



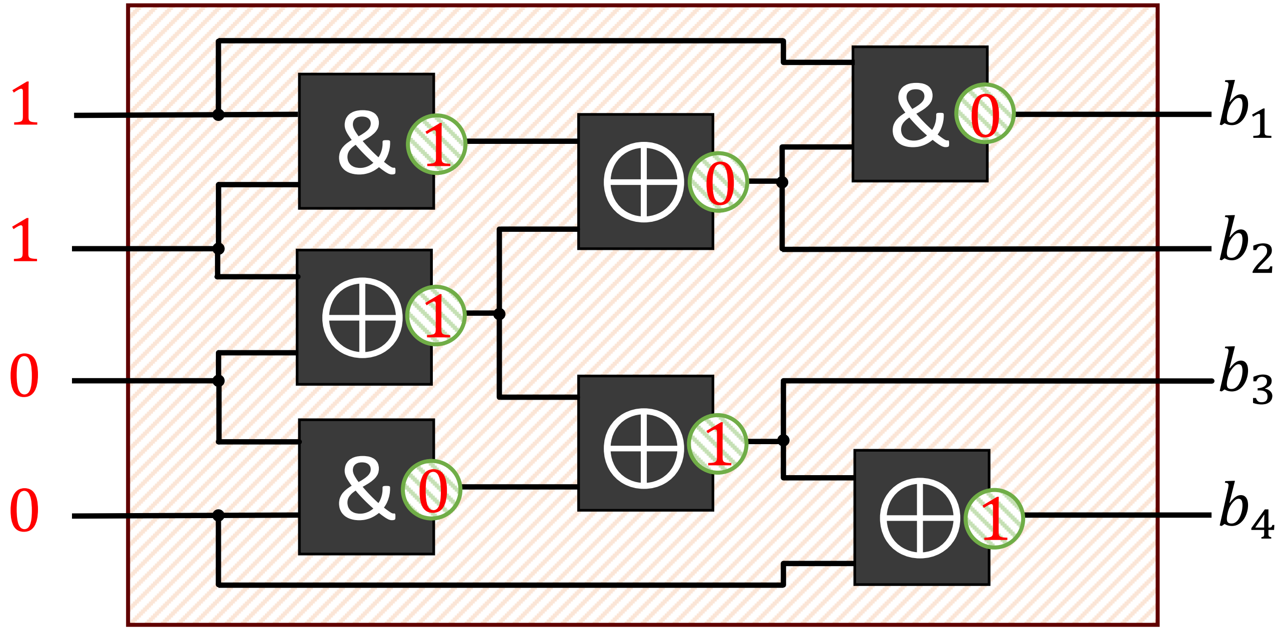
Traces generation

a_1	a_2	a_3	a_4	n_1	n_2	n_3	n_4	n_5	n_6	n_7



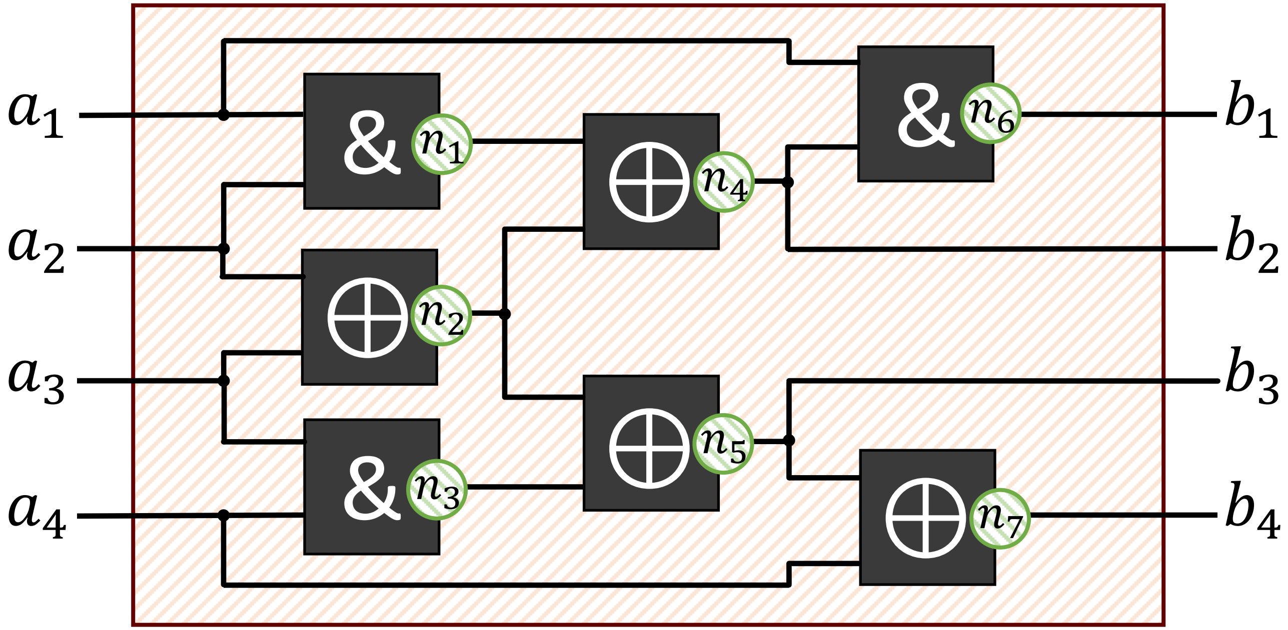
Traces generation

a_1	a_2	a_3	a_4	n_1	n_2	n_3	n_4	n_5	n_6	n_7
1	1	0	0	1	1	0	0	1	0	1

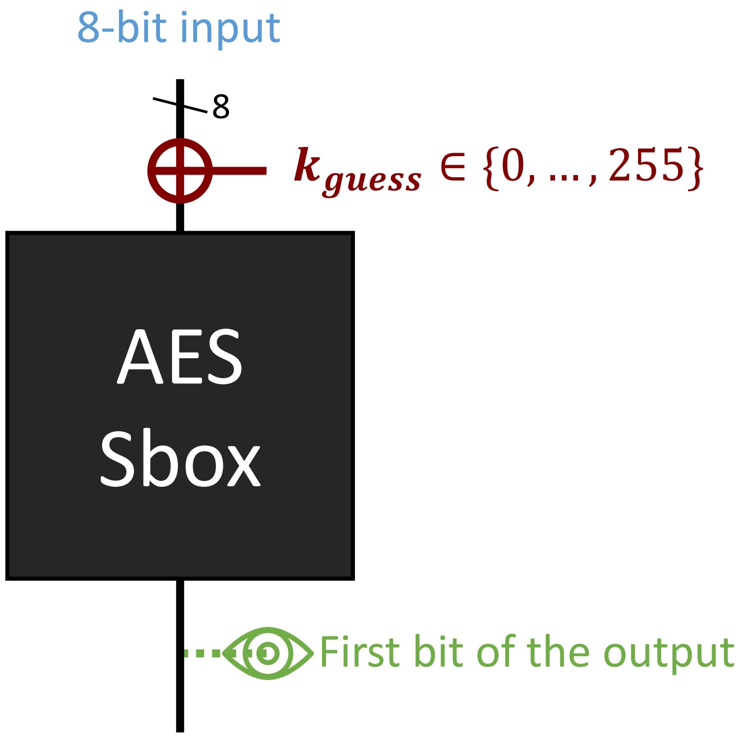


Traces generation

a_1	a_2	a_3	a_4	n_1	n_2	n_3	n_4	n_5	n_6	n_7
1	1	0	0	1	1	0	0	1	0	1
1	0	1	0	0	1	0	1	1	1	1
0	0	0	1	0	0	0	0	0	0	1
1	0	1	1	0	1	1	1	0	1	1
1	0	0	1	0	0	0	0	0	0	1
0	0	1	1	0	1	1	1	0	0	1
1	1	0	1	1	1	0	0	1	0	0
1	1	1	1	1	0	1	1	1	1	0
0	1	1	0	0	0	0	0	0	0	0



Selection function



8-bit inputs	k_0	k_1	k_2	k_3	k_4	...	k_{255}
0b01011011	1	0	1	0	0	...	1
0b11001111	0	0	1	1	1	...	1
0b00101101	1	1	0	1	1	...	0
0b11100101	0	1	1	1	0	...	1
0b00101011	0	0	0	1	0	...	1
0b01011110	1	0	0	1	1	...	1
0b00100110	1	1	1	0	0	...	0
0b10101100	0	0	1	0	1	...	0
⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮
0b11011010	1	1	1	0	1	...	1

Selection function

128-bit inputs	n_1	n_2	n_3	n_4	n_5	n_6	n_7	…	$n_{\#N}$	k_0	k_1	k_2	k_3	k_4	…	k_{255}
0x2a61…e030	1	0	1	1	0	0	1	…	1	1	0	1	0	0	…	1
0x118c…3699	1	1	1	1	1	1	0	…	0	0	0	1	1	1	…	1
0x243d…590c	0	0	1	0	1	1	1	…	1	1	1	0	1	1	…	0
0x39ab…4f21	0	0	0	1	1	0	0	…	0	0	1	1	1	0	…	1
0x21fe…5bf1	1	0	1	1	1	0	1	…	0	0	0	0	1	0	…	1
0x1106…de2f	0	0	0	1	1	0	1	…	1	1	0	0	1	1	…	1
0x30d5…494c	1	1	1	0	0	1	0	…	0	1	1	1	0	0	…	0
0x27b9…efbd	0	0	0	1	0	1	1	…	1	0	0	1	0	1	…	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮
0x2881…b3f9	1	0	0	0	0	1	0	…	1	1	1	1	0	1	…	1

Selection function

128-bit inputs	n_1	n_2	n_3	n_4	n_5	n_6	n_7	…	$n_{\#N}$	k_0	k_1	k_2	k_3	k_4	…	k_{255}
0x2a61…e030	1	0	1	1	0	0	1	…	1	1	0	1	0	0	…	1
0x118c…3699	1	1	1	1	1	1	0	…	0	0	0	1	1	1	…	1
0x243d…590c	0	0	1	0	1	1	1	…	1	1	1	0	1	1	…	0
0x39ab…4f21	0	0	0	1	1	0	0	…	0	0	1	1	1	0	…	1
0x21fe…5bf1	1	0	1	1	1	0	1	…	0	0	0	0	1	0	…	1
0x1106…de2f	0	0	0	1	1	0	1	…	1	1	0	0	1	1	…	1
0x30d5…494c	1	1	1	0	0	1	0	…	0	1	1	1	0	0	…	0
0x27b9…efbd	0	0	0	1	0	1	1	…	1	0	0	1	0	1	…	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮
0x2881…b3f9	1	0	0	0	0	1	0	…	1	1	1	1	0	1	…	1

$\overrightarrow{n_5}$ and $\overrightarrow{k_3}$ are equal, therefore $\overrightarrow{k_3}$ should be the correct key byte

ISW Masking Scheme and Algebraic attacks

ISW Masking Scheme

- Ishai, Sahai and Wagner introduced in 2003[1] a first masking masking scheme, that has the decoding function:

$$s = x_1 \oplus \cdots \oplus x_l$$

- For simplicity, let us focus on the case $l = 3$:

$$s = x_1 \oplus x_2 \oplus x_3$$

1. Crypto 2003: Private circuits: Securing hardware against probing attacks

ISW Masking Scheme

- There exists $\overrightarrow{n_\mathrm{a}}, \overrightarrow{n_\mathrm{b}}, \overrightarrow{n_\mathrm{c}}$ and $g \in \{0, \ldots, 255\}$ such that $\overrightarrow{n_\mathrm{a}} \oplus \overrightarrow{n_\mathrm{b}} \oplus \overrightarrow{n_\mathrm{c}} = \overrightarrow{k_\mathrm{g}}$.

128-bit inputs	n_1	n_2	n_3	n_4	n_5	n_6	n_7	…	$n_{\#N}$	k_0	k_1	k_2	k_3	k_4	…	k_{255}
0x2a61…e030	0	0	1	1	0	0	1	…	1	1	0	1	0	0	…	1
0x118c…3699	0	1	0	1	1	1	0	…	0	0	0	1	1	1	…	1
0x243d…590c	0	1	1	0	1	1	1	…	1	1	1	0	1	1	…	0
0x39ab…4f21	1	0	0	1	0	0	0	…	0	0	1	1	1	0	…	1
0x21fe…5bf1	0	0	1	1	0	0	1	…	0	0	0	0	1	0	…	1
0x1106…de2f	1	1	0	1	1	0	1	…	1	1	0	0	1	1	…	1
0x30d5…494c	1	1	1	0	1	1	0	…	0	1	1	1	0	0	…	0
0x27b9…efbd	0	0	0	1	0	1	1	…	1	0	0	1	0	1	…	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮
0x2881…b3f9	1	1	0	0	1	1	0	…	1	1	1	1	0	1	…	1

ISW Masking Scheme

$$\overrightarrow{n_1} \oplus \overrightarrow{n_3} \oplus \overrightarrow{n_5} = \overrightarrow{k_2}$$

128-bit inputs	n_1	n_2	n_3	n_4	n_5	n_6	n_7	…	$n_{\#N}$	k_0	k_1	k_2	k_3	k_4	…	k_{255}
0x2a61…e030	1	0	1	1	0	0	1	…	1	1	0	1	0	0	…	1
0x118c…3699	1	1	0	1	1	1	0	…	0	0	0	1	1	1	…	1
0x243d…590c	0	1	0	0	1	1	1	…	1	1	1	0	1	1	…	0
0x39ab…4f21	1	0	0	1	0	0	0	…	0	0	1	1	1	0	…	1
0x21fe…5bf1	0	0	1	1	1	0	1	…	0	0	0	0	1	0	…	1
0x1106…de2f	1	1	0	1	1	0	1	…	1	1	0	0	1	1	…	1
0x30d5…494c	1	1	1	0	1	1	0	…	0	1	1	1	0	0	…	0
0x27b9…efbd	1	0	0	1	1	1	1	…	1	0	0	1	0	1	…	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮
0x2881…b3f9	1	1	1	0	1	1	0	…	1	1	1	1	0	1	…	1

Linear Decoding Attack[1]

$W = 5$

$M=$

n_1	n_2	n_3	n_4	n_5
1	0	1	1	0
1	1	0	1	1
0	1	0	0	1
1	0	0	1	0
0	0	1	1	1
1	1	0	1	1
1	1	1	0	1
1	0	0	1	1
⋮	⋮	⋮	⋮	⋮
1	1	1	0	1

$S=$

k_0	k_1	k_2	k_3	k_4	…	k_{255}
1	0	1	0	0	…	1
0	0	1	1	1	…	1
1	1	0	1	1	…	0
0	1	1	1	0	…	1
0	0	0	1	0	…	1
1	0	0	1	1	…	1
1	1	1	0	0	…	0
0	0	1	0	1	…	0
⋮	⋮	⋮	⋮	⋮	⋱	⋮
1	1	1	0	1	…	1

1. Journal of Cryptographic Engineering 2020: How to reveal the secrets of an obscure white-box implementation

Linear Decoding Attack

$$
\begin{array}{c}
\begin{array}{ccccc} n_1 & n_2 & n_3 & n_4 & n_5 \end{array} \\
\begin{bmatrix}
1 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
1 & 1 & 1 & 0 & 1
\end{bmatrix}
\end{array}
\begin{array}{c}
\vec{x} \\
\begin{bmatrix} ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}
\end{array}
=
$$

k_0	k_1	k_2	k_3	k_4	...	k_{255}
1	0	1	0	0	...	1
0	0	1	1	1	...	1
1	1	0	1	1	...	0
0	1	1	1	0	...	1
0	0	0	1	0	...	1
1	0	0	1	1	...	1
1	1	1	0	0	...	0
0	0	1	0	1	...	0
⋮	⋮	⋮	⋮	⋮	⋱	⋮
1	1	1	0	1	...	1

Linear Decoding Attack

$$
\begin{array}{c}
\begin{array}{ccccc} n_1 & n_2 & n_3 & n_4 & n_5 \end{array} \\
\begin{bmatrix}
1 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
1 & 1 & 1 & 0 & 1
\end{bmatrix}
\end{array}
\begin{array}{c}
\vec{x} \\
\begin{bmatrix} ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}
\end{array}
=
\begin{array}{ccccccc}
k_0 & k_1 & k_2 & k_3 & k_4 & \ldots & k_{255} \\
1 & 0 & 1 & 0 & 0 & \ldots & 1 \\
0 & 0 & 1 & 1 & 1 & \ldots & 1 \\
1 & 1 & 0 & 1 & 1 & \ldots & 0 \\
0 & 1 & 1 & 1 & 0 & \ldots & 1 \\
0 & 0 & 0 & 1 & 0 & \ldots & 1 \\
1 & 0 & 0 & 1 & 1 & \ldots & 1 \\
1 & 1 & 1 & 0 & 0 & \ldots & 0 \\
0 & 0 & 1 & 0 & 1 & \ldots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & 1 & 1 & 0 & 1 & \ldots & 1
\end{array}
$$

Linear Decoding Attack

n_1	n_2	n_3	n_4	n_5
1	0	1	1	0
1	1	0	1	1
0	1	0	0	1
1	0	0	1	0
0	0	1	1	1
1	1	0	1	1
1	1	1	0	1
1	0	0	1	1
⋮	⋮	⋮	⋮	⋮
1	1	1	0	1

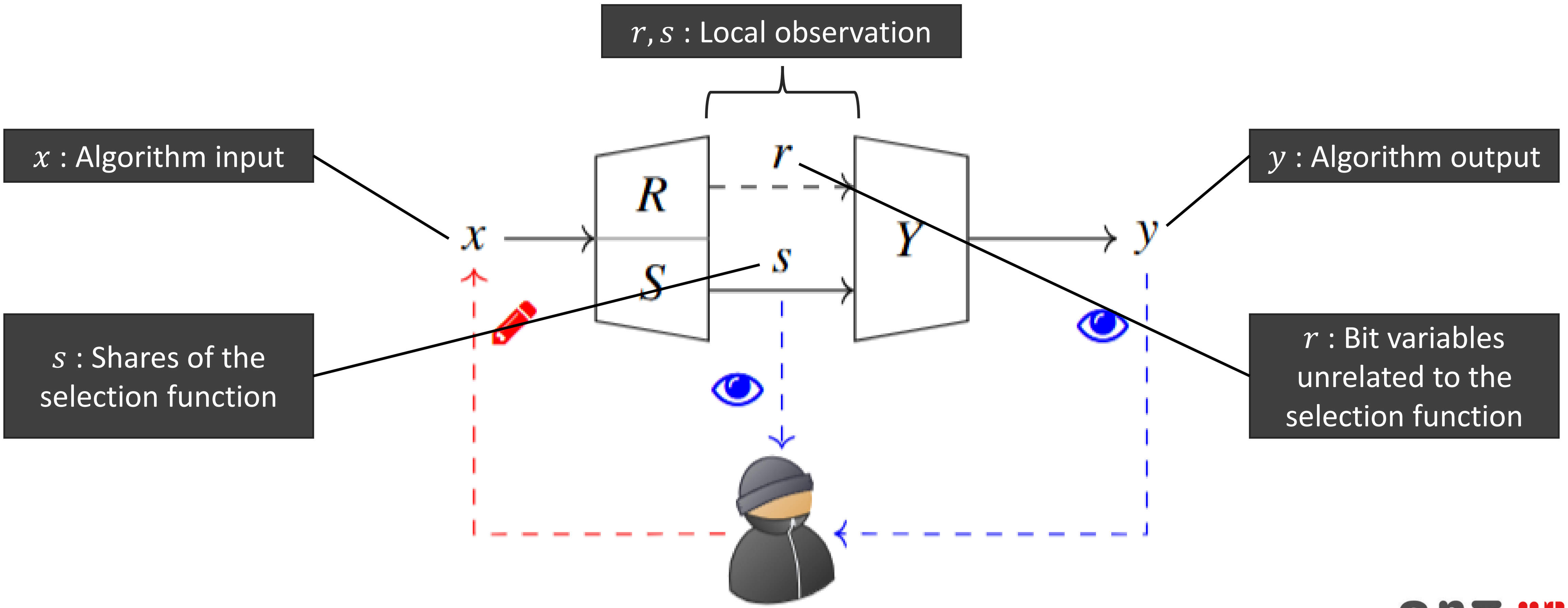
$\vec{x}$
1
0
1
0
1

=

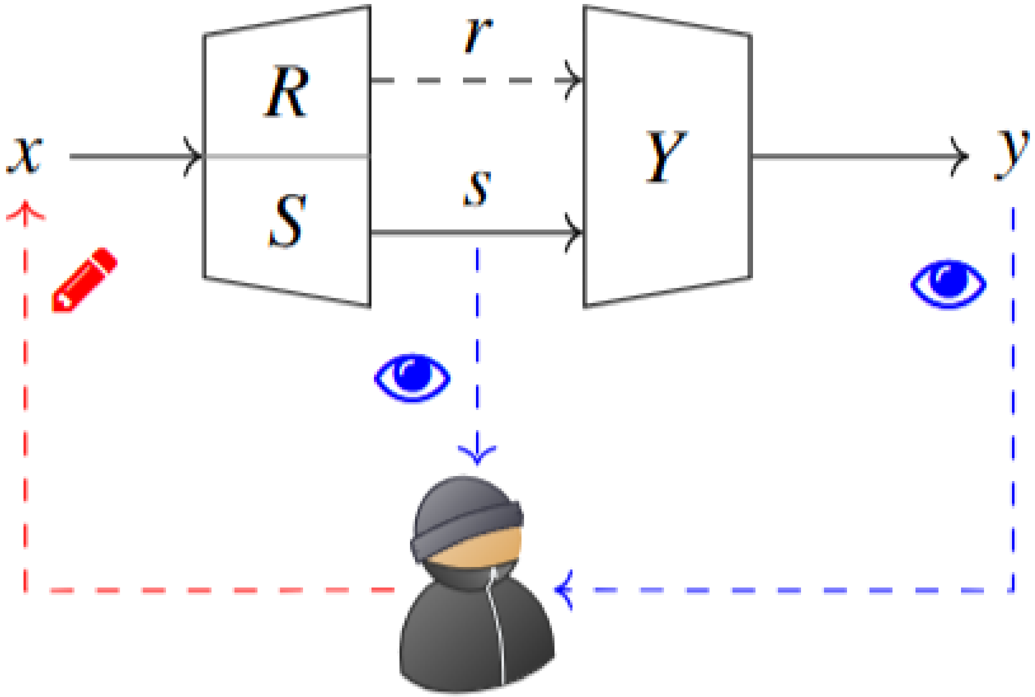
k_0	k_1	k_2	k_3	k_4	…	k_{255}
1	0	1	0	0	…	1
0	0	1	1	1	…	1
1	1	0	1	1	…	0
0	1	1	1	0	…	1
0	0	0	1	0	…	1
1	0	0	1	1	…	1
1	1	1	0	0	…	0
0	0	1	0	1	…	0
⋮	⋮	⋮	⋮	⋮	⋱	⋮
1	1	1	0	1	…	1

Side-channel Attacks under our Haystack model

Side-channel attacks



ISW Haystack cipher



- To encode a bit variable p into l ISW shares, you compute :
  - $x_1 = p \oplus \$_2 \cdots \oplus \$_l$, with $\$_i$ being random values
  - $x_2 = \$_2,\ x_3 = \$_3,\ \cdots,\ x_l = \$_l$
  - We have : $p = x_1 \oplus x_2 \cdots \oplus x_l$

$$E_{k,\ell,r}(p) = \mathrm{Shuffle}_k(\underbrace{p \oplus \$_2 \oplus \ldots \oplus \$_\ell,\ \$_2, \$_3, \ldots, \$_\ell,}_{\text{ISW Shares}}\ \underbrace{\tilde{\$}_1, \ldots, \tilde{\$}_r}_{\text{Other node vectors}})$$

ISW Haystack cipher

p : Plaintext

$$E_{k,\ell,r}(p) = \overbrace{\mathrm{Shuffle}_k(\underbrace{p \oplus \$_2 \oplus \ldots \oplus \$_\ell,\ \$_2, \$_3, \ldots, \$_\ell,}_{\textbf{ISW Shares}}\ \underbrace{\tilde{\$}_1, \ldots, \tilde{\$}_r}_{\textbf{Other node vectors}})}^{\textbf{Ciphertext}}$$

k : Secret key

- Shuffle$_k$ is a fixed permutation, chosen uniformly at random per each value k

INDistinguishability under Chosen-Plaintext Attack

$$E_{k,\ell,r}(p) = \mathrm{Shuffle}_k(p \oplus \$_2 \oplus \ldots \oplus \$_\ell,\ \$_2, \$_3, \ldots, \$_\ell,\ \tilde{\$}_1, \ldots, \tilde{\$}_r)$$

$\triangleright$ (IND-CPA)

proc Initialize()

$k \xleftarrow{\$} K();\ b \xleftarrow{\$} \{0,1\}$

proc LR(p_0, p_1)

$c \xleftarrow{\$} E_k(p_b)$; **return** c

proc Enc(p)

$c \xleftarrow{\$} E_k(p)$; **return** c

proc Finalize(b')

If $(b' = b)$ **return** Win

else **return** Loose

- Generating a trace is querying an encryption
- The attacker should not be able to recover information of the selection function from the shares

Fault Attacks under our Haystack model

Fault attacks

r, s : Local observation

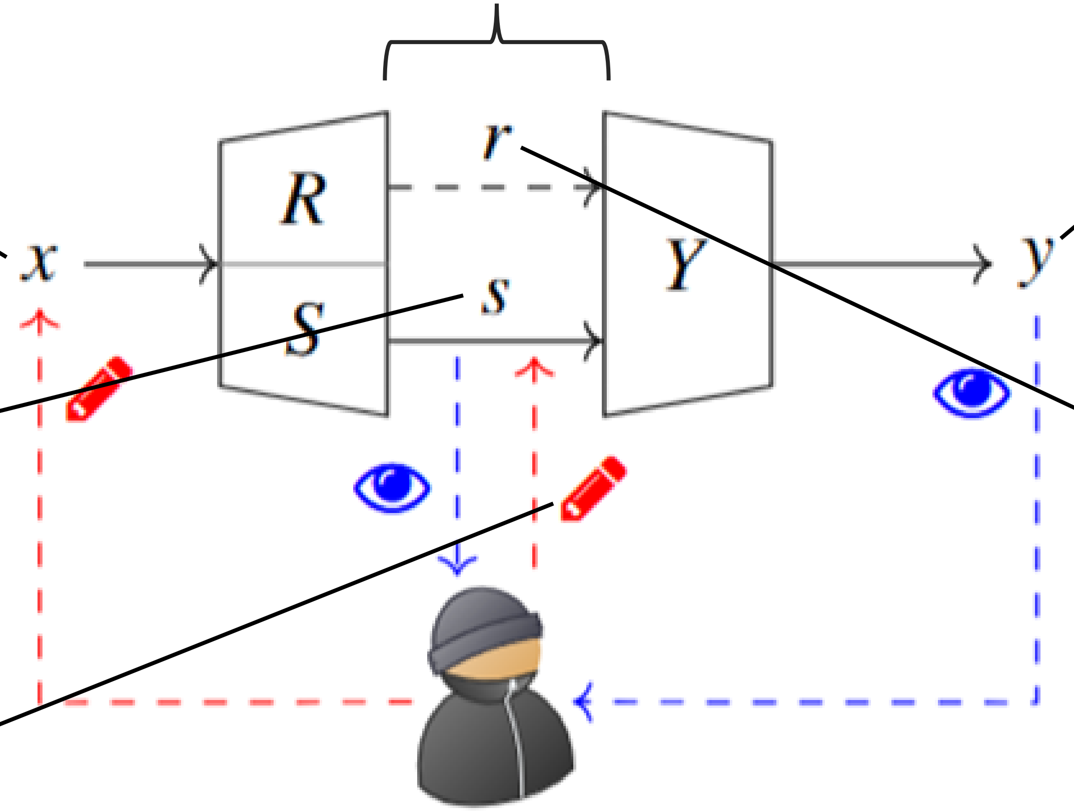
x : Algorithm input

y : Algorithm output

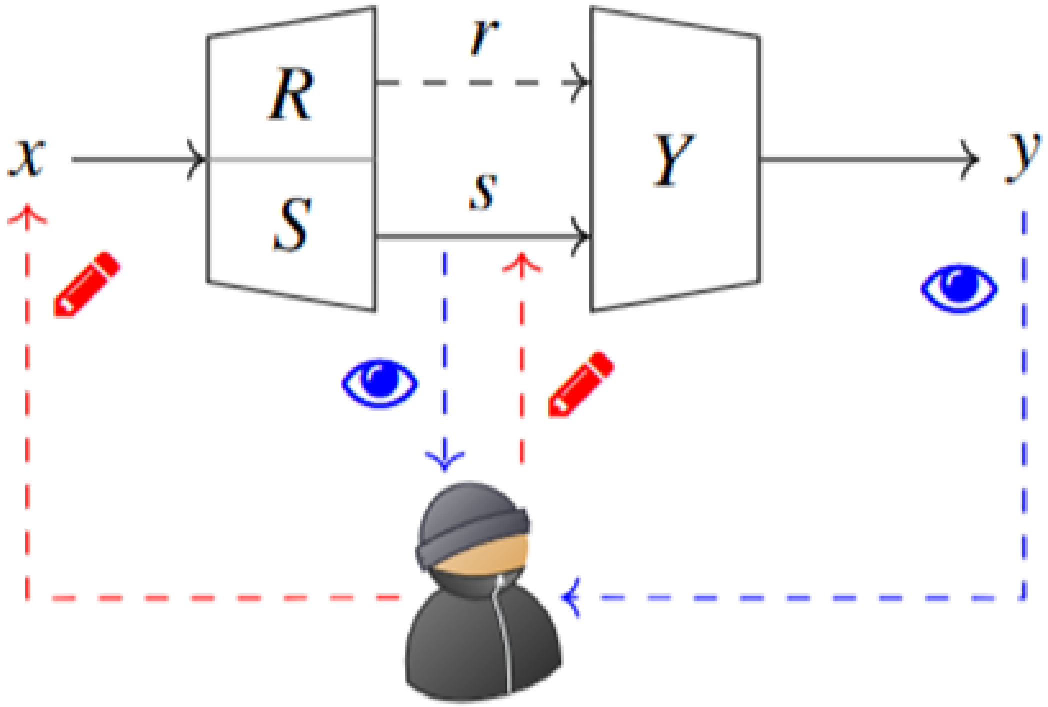
s : Shares of the selection function

r : Bit variables unrelated to the selection function

The attacker can modify s, r and observe the impact on y



Fault attack model



ASSUMPTION :

Injecting any fault on s, r is enough for the attacker to retrieve the value of the corresponding selection function

Fault attack model

- Let $E_k(p) = \text{Shuffle}_\text{k}(\underbrace{x_1, x_2, \dots, x_s}_{\textbf{Shares}}, \underbrace{\$_1, \dots, \$_r}_{\textbf{Other node vectors}})$ be a Haystack cipher

- The attacker can fault any variable bit variable of the ciphertext, and recover the associated plaintext p

ASSUMPTION :

Injecting any fault on s, r is enough for the attacker to retrieve the value of the corresponding selection function

- It is a Chosen Ciphertext Attack !

Fault attack model

- Let $E_k(p) = \text{Shuffle}_\text{k}(x_1, x_2, \dots, x_s, \$_1, \dots, \$_r)$ be a Haystack cipher

▷ (IND-CCA1 / non-adaptive)

proc Initialize()

$k \xleftarrow{\$} K();\ b \xleftarrow{\$} (0,1);\ f \leftarrow 0$

proc Enc(p)

$c \xleftarrow{\$} E_k(p)$; **return** c

proc LR(p_0, p_1)

$c \xleftarrow{\$} E_k(p_b);\ f \leftarrow 1$; **return** c

proc Dec(c)

If $f = 0$ then **return** $D_k(c)$
else **return** $\perp$

proc Finalize(b)

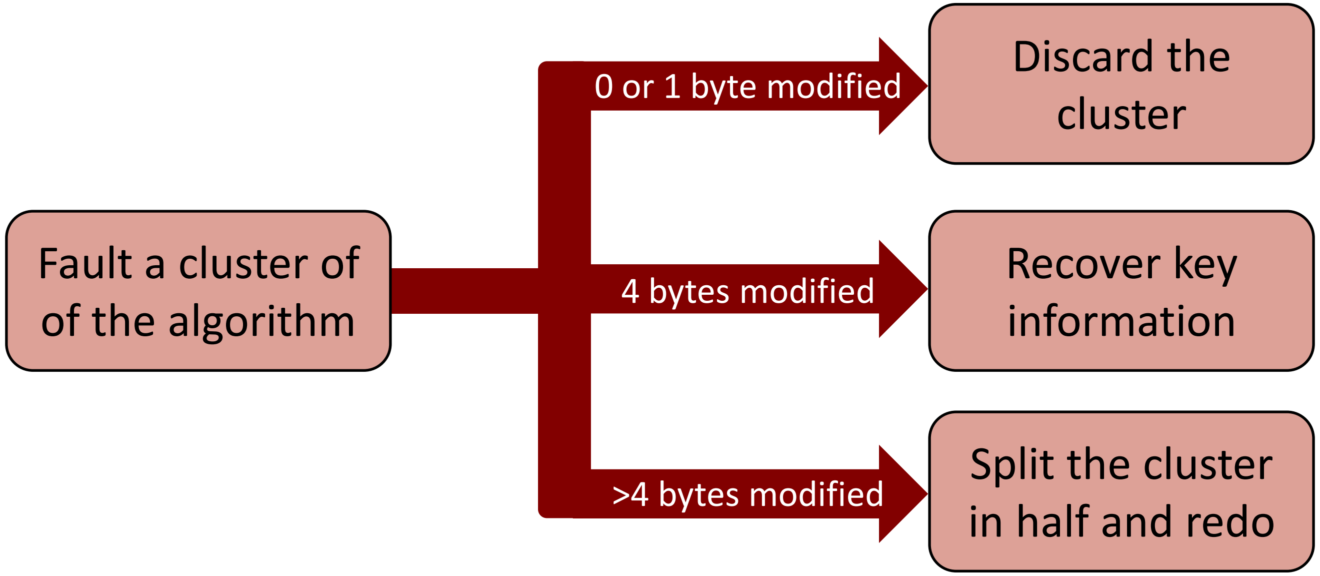
If $(b' = b)$ **return** Win
else **return** Loose

- Generating a trace is querying an encryption
- Faulting the ciphertext is querying a decryption
- The attacker should not be able to recover information of the selection function

White-box Fault Attacks

White-box Differential Fault attack

- It is possible to recover key information by injecting a fault between the two last AES rounds[1]
- In white-box, mounting such attack is easier[2] :



1. Crypto 1997: Differential Fault Analysis of secret key cryptosystems
2. https://blog.quarkslab.com/differential-fault-analysis-on-white-box-aes-implementations.html

Linear Fault Recoding attack

- Let us define a Code-based fault countermeasure:

Let $\mathbb{F}$ be a finite field, $\ell \in \mathbb{Z}_{>0}$ and $G \in \mathbb{F}^{(n+\ell)\times s}$ be a right-invertible matrix (in particular, $s \geq n + \ell$). Define an encoding function

$$\mathrm{Encode}^{\mathrm{CB}}_{\ell,G} : \mathbb{F}^n \to \mathbb{F}^s : \boldsymbol{p} \mapsto (\boldsymbol{p}||\$^\ell) \times G,$$

where $\$^\ell$ is sampled uniformly at random from $\mathbb{F}^\ell$.

- Once again, the shares follows a linear structure, that we can observe in white-box
- For example : If we find that $x_1 = x_2 \oplus x_3 \oplus x_5$ and $x_2 = x_3 \oplus x_4$, then we can choose x_3, x_4, x_5 and set x_1, x_2 accordingly, resulting in a valid codeword

Randomness removal by faulting

- Let us take back $E_k(p) = \mathrm{Shuffle}_\mathrm{k}(x_1, x_2, \ldots, x_s, \$_1, \ldots, \$_r)$
- Its decryption function is :

$$D_k(c) = \mathrm{Decode}(\mathrm{Unshuffle}_\mathrm{k}(c)\big|_{1,\ldots,s})$$

- Faulting the randomness $\$_1, \ldots, \$_r$ does not impact the decryption !

  ➡ An attacker can distinguish randomness from shared values

- This corresponds to a forgery attack in the CCA model

Forgery attacks : ineffective faults

- Randomness removal
- Double fault against ISW:

$$p = x_1 \oplus x_2 \oplus \cdots \oplus x_l = x_1 \oplus x_2 \oplus \cdots \oplus x_l$$

- Detection of Non-linear shares:

$$p = \underbrace{x_1 \oplus \cdots \oplus x_l}_{\text{Fault succeeds 100\% of the time}} \oplus \underbrace{x_{l+1} \cdot x_{l+2} \cdots x_{l+d}}_{\text{Fault succeeds if all the } d \text{ non-linear shares are equal to one}}$$

Symmetric cryptography and Physical cryptanalysis link

Security	Symmetric-key cryptography	White-box cryptography
CPA	Plaintext Ciphertext	Selection function Trace window
CCA1	Decryption query Decryption failure ($\perp$)	Fault injection Fault detection ($\perp$)
CCA2	Relative forgery (Malleability)	Targeted fault injection
CCA3	Existential forgery attack	Undetected fault

Thank you for your attention !

Haystack ciphers: White-box countermeasures as Symmetric encryption

https://eprint.iacr.org/2025/1635

Alex Charlès[1], Aleksei Udovenko[2]

1) DCS, University of Luxembourg : alex.charles205@gmail.com

2) SnT, University of Luxembourg : aleksei@affine.group