From Random Probing to Noisy Leakages Without Field-Size Dependence

Gianluca Brian
ETH Zurich
Switzerland

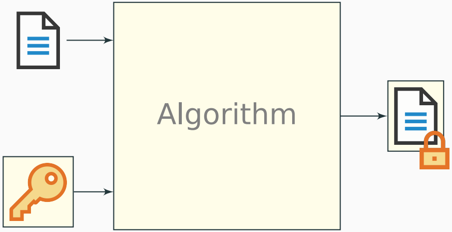
Stefan Dziembowski
University of Warsaw
Poland

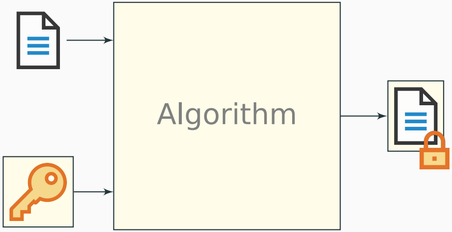
Sebastian Faust
TU Darmstadt
Germany

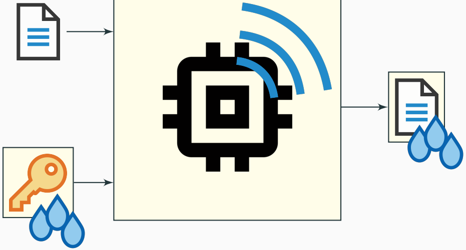
Eurocrypt 2024

Zurich, Switzerland

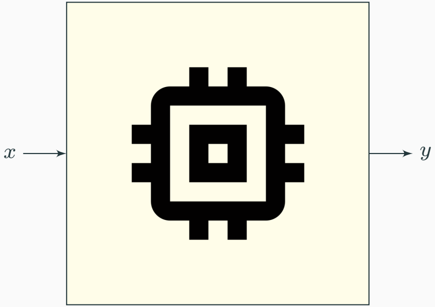
Introduction



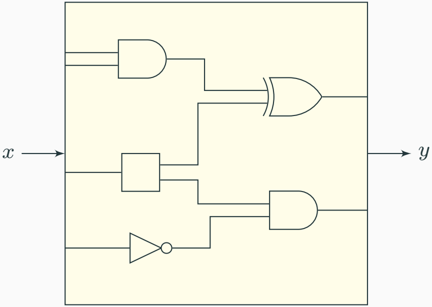

Algorithm



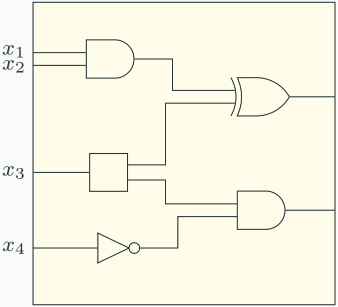
Computation



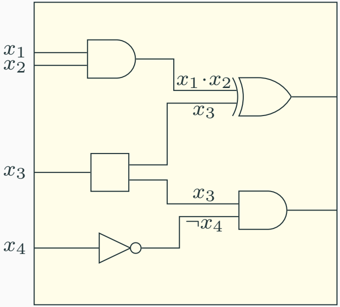
Computation



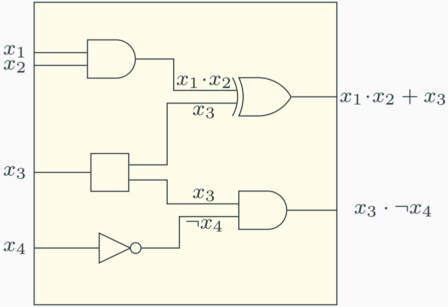
Computation



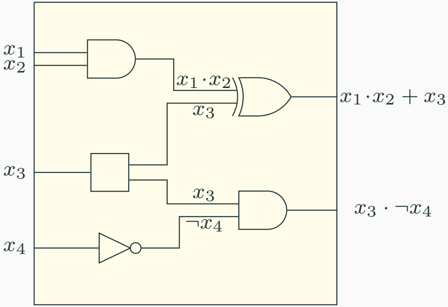
Computation



Computation

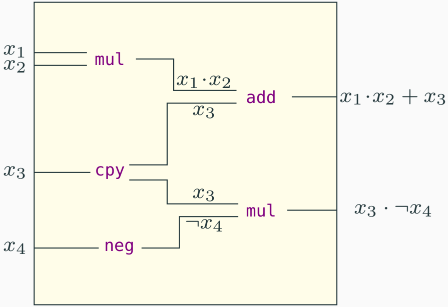


Computation



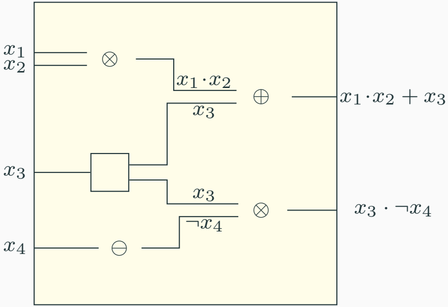
$$f(x_1, x_2, x_3, x_4) := (x_1 \cdot x_2 + x_3, x_3 \cdot \neg x_4)$$

Computation



$$f(x_1, x_2, x_3, x_4) := (x_1 \cdot x_2 + x_3, x_3 \cdot \neg x_4)$$

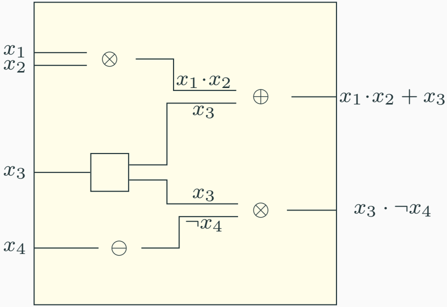
Computation



$$f(x_1, x_2, x_3, x_4) := (x_1 \cdot x_2 + x_3, x_3 \cdot \neg x_4)$$

- Computation in a field $\mathbb{F}$

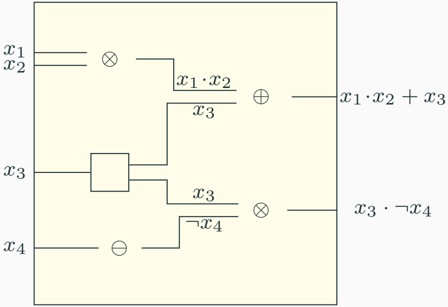
Computation



$$f(x_1, x_2, x_3, x_4) := (x_1 \cdot x_2 + x_3, x_3 \cdot \neg x_4)$$

- Computation in a field $\mathbb{F}$
- Addition: $\oplus$
- Subtraction: $\ominus$
- Multiplication: $\otimes$

Computation



$$f(x_1, x_2, x_3, x_4) := (x_1 \cdot x_2 + x_3, x_3 \cdot \neg x_4)$$

- Computation in a field $\mathbb{F}$
- Addition: $\oplus$
- Subtraction: $\ominus$
- Multiplication: $\otimes$
- Copy: $\square$
- Random: $\$$

Leakage models

t-probing [ISW03]

- Simple
- Limited amount of probes

Random Probing

Average Probing

Noisy Leakages

- Most realistic
- Difficult to achieve

[ISW03] Y. Ishai, A. Sahai, D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks", *Crypto 2003*

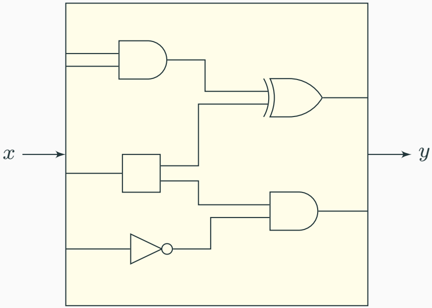
Leakage models

t-probing [ISW03]	Random Probing	Average Probing	Noisy Leakages
• Simple • Limited amount of probes			• Most realistic • Difficult to achieve

[ISW03] Y. Ishai, A. Sahai, D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks", *Crypto 2003*

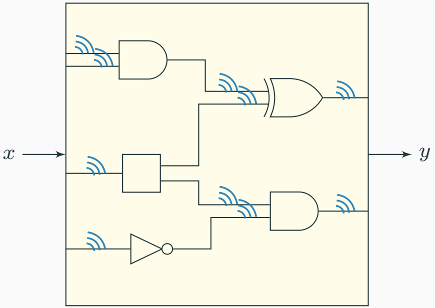
Leakage models

Random probing



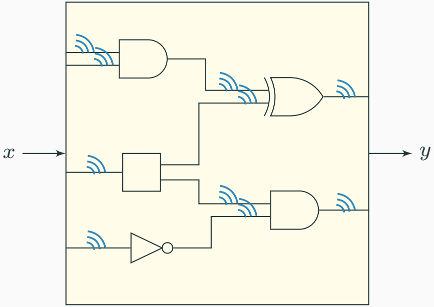
Leakage models

Random probing



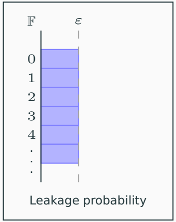
Leakage models

Random probing



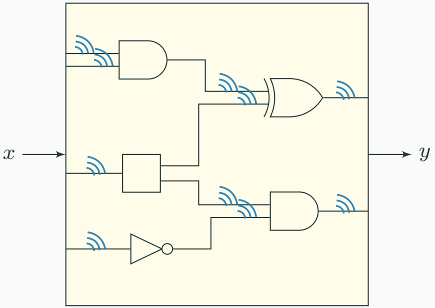
- Probing function $= \boldsymbol{\rho}$
- Random probing:

$$\forall w, \Pr\left[\boldsymbol{\rho}(w) = w\right] = \varepsilon$$



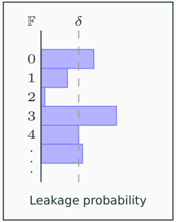
Leakage models

Average probing



- Probing function = $\boldsymbol{\alpha}$
- Average probing [DFS15]:

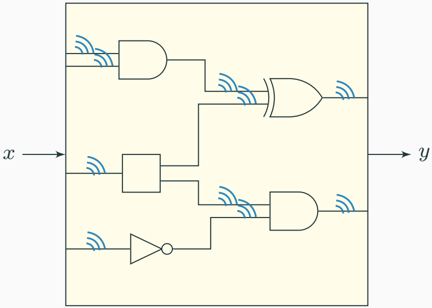
For uniform $\mathbf{U}, \Pr[\boldsymbol{\alpha}(\mathbf{U}) = \mathbf{U}] = \delta$



[DFS15] S. Dziembowski, S. Faust, M. Skorski, "Noisy Leakage Revisited", *Eurocrypt 2015*

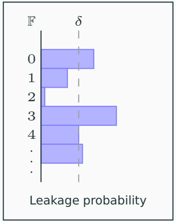
Leakage models

Average probing



- Probing function $= \boldsymbol{\alpha}$
- Average probing [DFS15]:

  For uniform $\mathbf{U}, \Pr\left[\boldsymbol{\alpha}(\mathbf{U}) = \mathbf{U}\right] = \delta$



- Reveals more information!

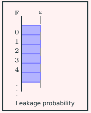
[DFS15] S. Dziembowski, S. Faust, M. Skorski, "Noisy Leakage Revisited", *Eurocrypt 2015*

Leakage models

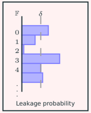
t-probing [ISW03]

- Simple
- Limited amount of probes

Random Probing



Average Probing



Noisy Leakages

- Most realistic
- Difficult to achieve

[ISW03] Y. Ishai, A. Sahai, D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks", *Crypto 2003*

Leakage models

[DDF14]

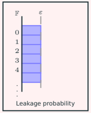
$\eta = \frac{\varepsilon}{|\mathbb{F}|}$

t-probing [ISW03]

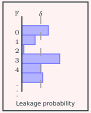
- Simple
- Limited amount of probes

[DDF14]

Random Probing



Average Probing

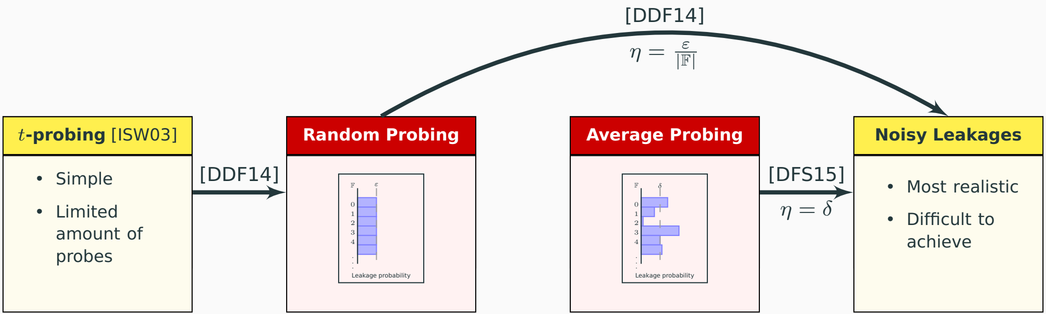


Noisy Leakages

- Most realistic
- Difficult to achieve

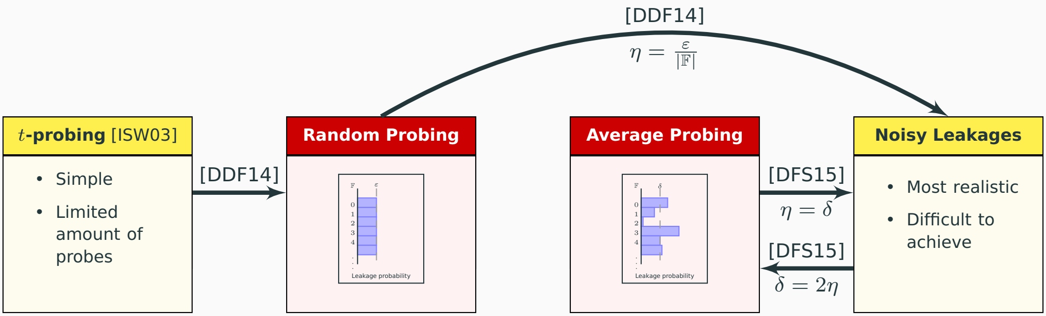
[ISW03] Y. Ishai, A. Sahai, D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks", *Crypto 2003*
[DDF14] A. Duc, S. Dziembowski, S. Faust, "Unifying Leakage Models: from Probing Attacks to Noisy Leakage", *Eurocrypt 2014*

Leakage models



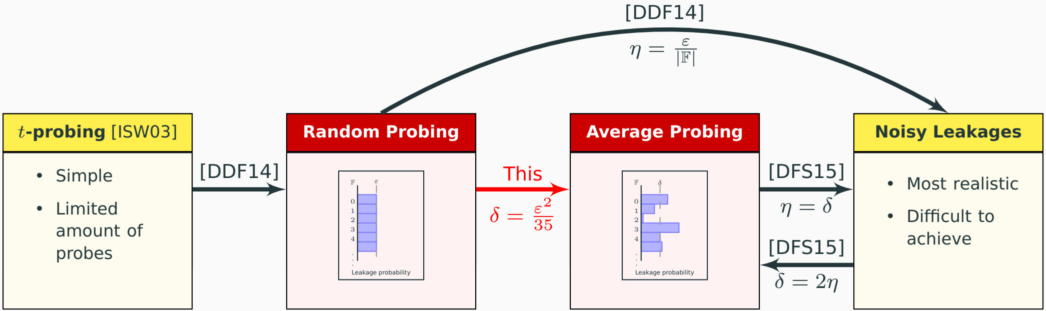
[ISW03] Y. Ishai, A. Sahai, D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks", *Crypto 2003*
[DDF14] A. Duc, S. Dziembowski, S. Faust, "Unifying Leakage Models: from Probing Attacks to Noisy Leakage", *Eurocrypt 2014*
[DFS15] S. Dziembowski, S. Faust, M. Skorski, "Noisy Leakage Revisited", *Eurocrypt 2015*

Leakage models



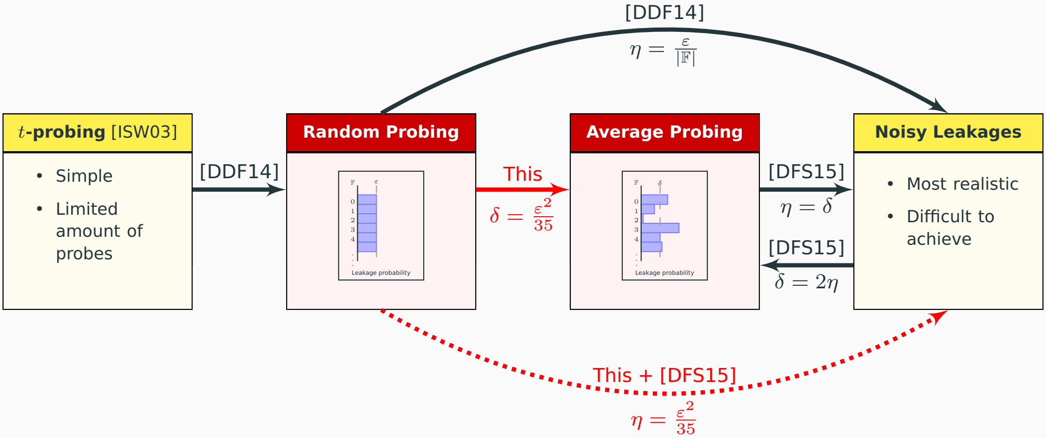
[ISW03] Y. Ishai, A. Sahai, D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks", *Crypto 2003*
[DDF14] A. Duc, S. Dziembowski, S. Faust, "Unifying Leakage Models: from Probing Attacks to Noisy Leakage", *Eurocrypt 2014*
[DFS15] S. Dziembowski, S. Faust, M. Skorski, "Noisy Leakage Revisited", *Eurocrypt 2015*

Leakage models



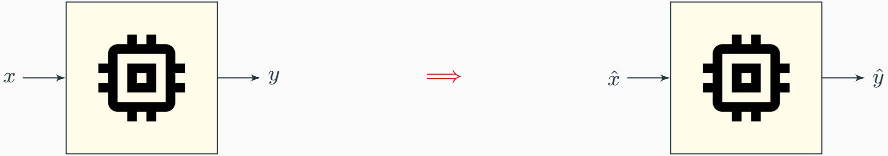
[ISW03] Y. Ishai, A. Sahai, D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks", *Crypto 2003*
[DDF14] A. Duc, S. Dziembowski, S. Faust, "Unifying Leakage Models: from Probing Attacks to Noisy Leakage", *Eurocrypt 2014*
[DFS15] S. Dziembowski, S. Faust, M. Skorski, "Noisy Leakage Revisited", *Eurocrypt 2015*

Leakage models

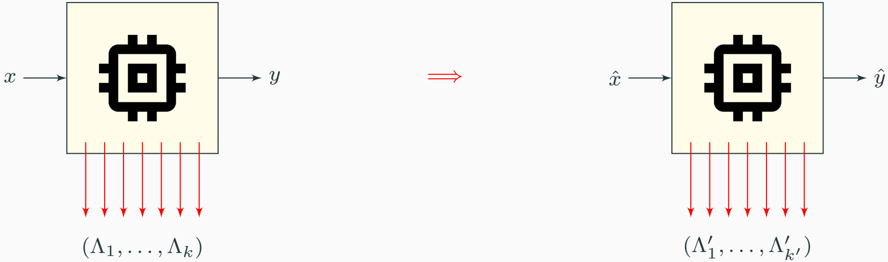


[ISW03] Y. Ishai, A. Sahai, D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks", *Crypto 2003*
[DDF14] A. Duc, S. Dziembowski, S. Faust, "Unifying Leakage Models: from Probing Attacks to Noisy Leakage", *Eurocrypt 2014*
[DFS15] S. Dziembowski, S. Faust, M. Skorski, "Noisy Leakage Revisited", *Eurocrypt 2015*

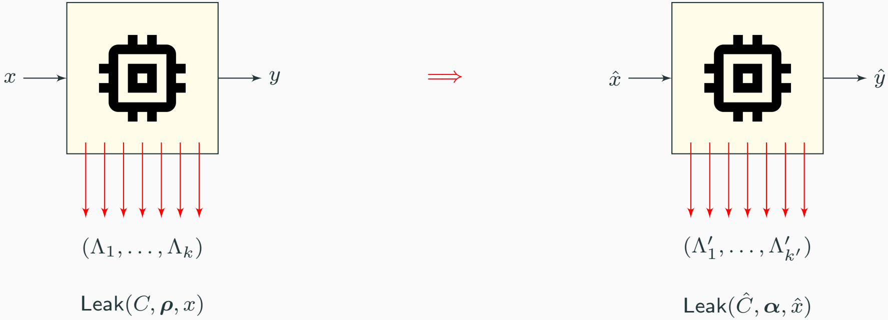
The simulation paradigm



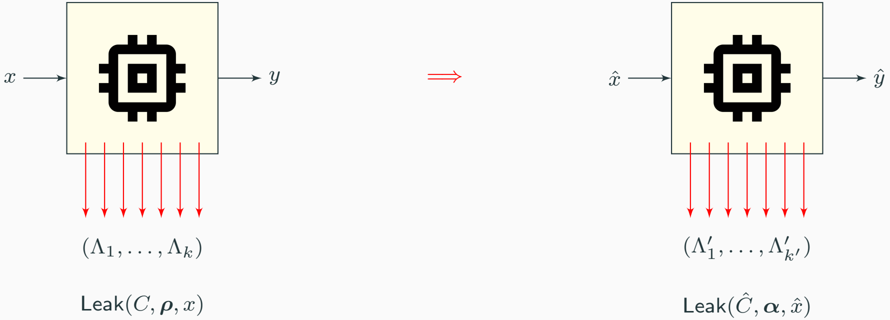
The simulation paradigm



The simulation paradigm

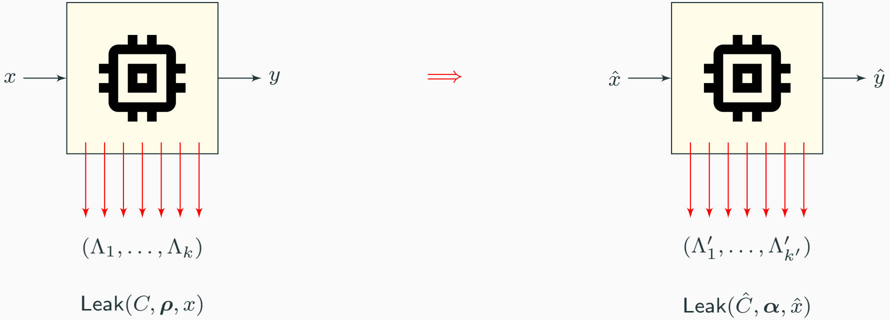


The simulation paradigm



Simulatability: there exists a simulator Sim such that $\mathsf{Sim}(\mathsf{Leak}(C, \boldsymbol{\rho}, x)) \equiv \mathsf{Leak}(\hat{C}, \boldsymbol{\alpha}, \hat{x})$.

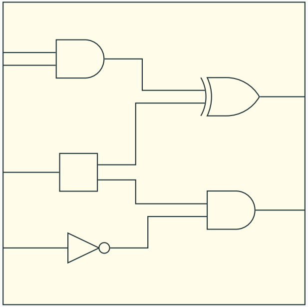
The simulation paradigm



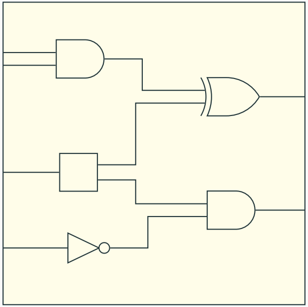
Simulatability: there exists a simulator Sim such that $\mathsf{Sim}(\mathsf{Leak}(C, \boldsymbol{\rho}, x)) \equiv \mathsf{Leak}(\hat{C}, \boldsymbol{\alpha}, \hat{x})$.

Security: if C is ε-**random**-probing secure and leakage from $\hat{C}$ is perfectly simulatable, then $\hat{C}$ is δ-**average**-probing secure.

Our compiler



Our compiler



Compiler →

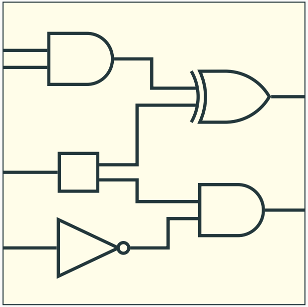
Masking:

$$x \mapsto \hat{x} = (x_1, x_2)$$

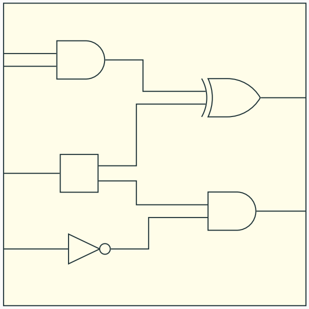
$$x_1 + x_2 = x$$

$$g \mapsto \hat{g}$$

$$g(x) = \mathsf{Dec}(\hat{g}(\hat{x}))$$



Our compiler



Compiler →

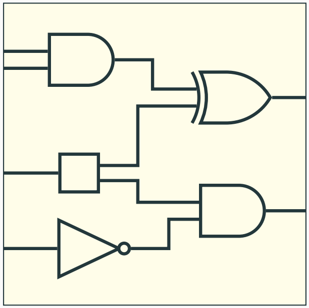
Masking:

$$x \mapsto \hat{x} = (x_1, x_2)$$

$$x_1 + x_2 = x$$

$$g \mapsto \hat{g}$$

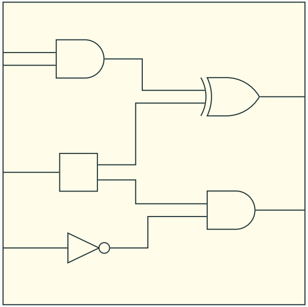
$$g(x) = \mathsf{Dec}(\hat{g}(\hat{x}))$$



Security proof: standard hybrid argument.

$$\mathsf{Real}(\hat{C}, \boldsymbol{\alpha}, \hat{x})$$

Our compiler



Compiler

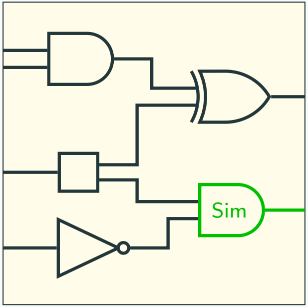
Masking:

$$x \mapsto \hat{x} = (x_1, x_2)$$

$$x_1 + x_2 = x$$

$$g \mapsto \hat{g}$$

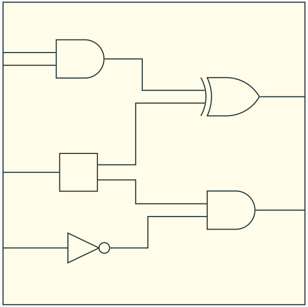
$$g(x) = \mathsf{Dec}(\hat{g}(\hat{x}))$$



Security proof: standard hybrid argument.

$$\mathsf{Real}(\hat{C}, \boldsymbol{\alpha}, \hat{x}) \equiv \mathsf{Hyb}_1(\hat{C}, \hat{x})$$

Our compiler



Compiler

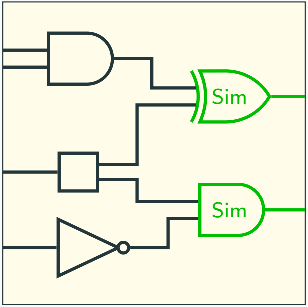
Masking:

$$x \mapsto \hat{x} = (x_1, x_2)$$

$$x_1 + x_2 = x$$

$$g \mapsto \hat{g}$$

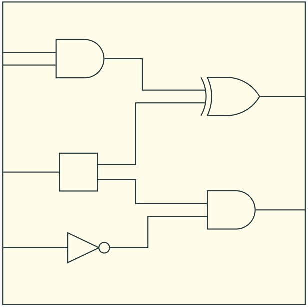
$$g(x) = \mathsf{Dec}(\hat{g}(\hat{x}))$$



Security proof: standard hybrid argument.

$$\mathsf{Real}(\hat{C}, \boldsymbol{\alpha}, \hat{x}) \equiv \mathsf{Hyb}_1(\hat{C}, \hat{x}) \equiv \mathsf{Hyb}_2(\hat{C}, \hat{x})$$

Our compiler



Compiler →

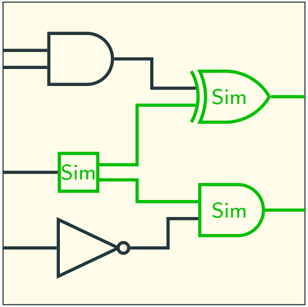
Masking:

$$x \mapsto \hat{x} = (x_1, x_2)$$

$$x_1 + x_2 = x$$

$$g \mapsto \hat{g}$$

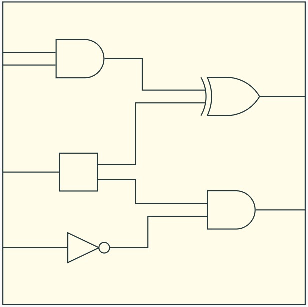
$$g(x) = \mathsf{Dec}(\hat{g}(\hat{x}))$$



Security proof: standard hybrid argument.

$$\mathsf{Real}(\hat{C}, \boldsymbol{\alpha}, \hat{x}) \equiv \mathsf{Hyb}_1(\hat{C}, \hat{x}) \equiv \mathsf{Hyb}_2(\hat{C}, \hat{x}) \equiv \ldots$$

Our compiler



Compiler

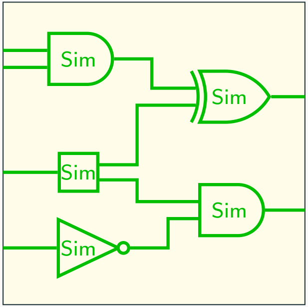
Masking:

$$x \mapsto \hat{x} = (x_1, x_2)$$

$$x_1 + x_2 = x$$

$$g \mapsto \hat{g}$$

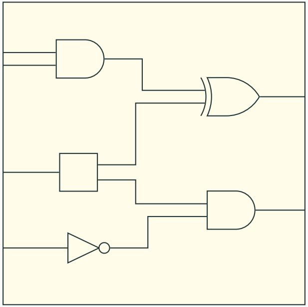
$$g(x) = \mathsf{Dec}(\hat{g}(\hat{x}))$$



Security proof: standard hybrid argument.

$$\mathsf{Real}(\hat{C}, \boldsymbol{\alpha}, \hat{x}) \equiv \mathsf{Hyb}_1(\hat{C}, \hat{x}) \equiv \mathsf{Hyb}_2(\hat{C}, \hat{x}) \equiv \ldots \equiv \mathsf{Hyb}_n(\hat{C}, \hat{x})$$

Our compiler



Compiler

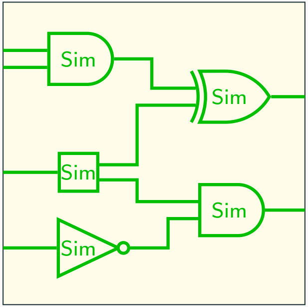
Masking:

$x \mapsto \hat{x} = (x_1, x_2)$

$x_1 + x_2 = x$

$g \mapsto \hat{g}$

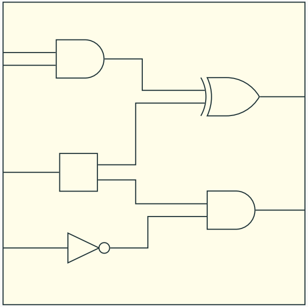
$g(x) = \mathsf{Dec}(\hat{g}(\hat{x}))$



Security proof: standard hybrid argument.

$$\mathsf{Real}(\hat{C}, \boldsymbol{\alpha}, \hat{x}) \equiv \mathsf{Hyb}_1(\hat{C}, \hat{x}) \equiv \mathsf{Hyb}_2(\hat{C}, \hat{x}) \equiv \ldots \equiv \mathsf{Hyb}_n(\hat{C}, \hat{x}) = \mathsf{Sim}(\hat{C}, \mathsf{Leak}(C, \boldsymbol{\rho}, x))$$

Our compiler



Compiler

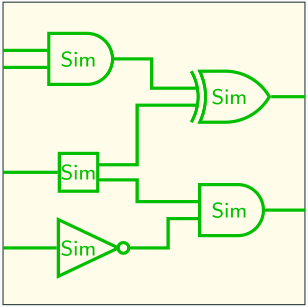
Masking:

$x \mapsto \hat{x} = (x_1, x_2)$

$x_1 + x_2 = x$

$g \mapsto \hat{g}$

$g(x) = \mathsf{Dec}(\hat{g}(\hat{x}))$



Security proof: standard hybrid argument.

Problem: we need composable gadgets!

$$\mathsf{Real}(\hat{C}, \boldsymbol{\alpha}, \hat{x}) \equiv \mathsf{Hyb}_1(\hat{C}, \hat{x}) \equiv \mathsf{Hyb}_2(\hat{C}, \hat{x}) \equiv \ldots \equiv \mathsf{Hyb}_n(\hat{C}, \hat{x}) = \mathsf{Sim}(\hat{C}, \mathsf{Leak}(C, \boldsymbol{\rho}, x))$$

Composable gadgets

Example: gate $g : \mathbb{F} \to \mathbb{F}$ with fan-in 1 and fan-out 1:

$$\mathsf{Real}_g\left(\hat{x}\right) \equiv \mathsf{Sim}_g\left(\boldsymbol{\rho}(x)\right)$$

Composable gadgets

Example: gate $g : \mathbb{F} \to \mathbb{F}$ with fan-in 1 and fan-out 1:

$$\mathsf{Real}_g\left(\hat{x}\right) \equiv \mathsf{Sim}_g\left(\boldsymbol{\rho}(x)\right)$$

Problem: simulation may be inconsistent with other parts of the circuit.

Composable gadgets

Example: gate $g : \mathbb{F} \to \mathbb{F}$ with fan-in 1 and fan-out 1:

$$\mathsf{Real}_g\left(\hat{x}\right) \equiv \mathsf{Sim}_g\left(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise}\right)$$

Problem: simulation may be inconsistent with other parts of the circuit.

Solution: give the simulator the full encoded input $\hat{x}$ if the original input x leaks.

Composable gadgets

Example: gate $g : \mathbb{F} \to \mathbb{F}$ with fan-in 1 and fan-out 1:

$$\mathsf{Real}_g\,(\hat{x}) \equiv \mathsf{Sim}_g\,(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise})$$

Problem: simulation may be inconsistent with other parts of the circuit.

Solution: give the simulator the full encoded input $\hat{x}$ if the original input x leaks.

Problem: what about gates with fan-in > 1?

Composable gadgets

Example: gate $g : \mathbb{F}^k \to \mathbb{F}$ with fan-in k and fan-out 1:

$$\mathsf{Real}_g\,(\hat{x}) \equiv \mathsf{Sim}_g\,(\hat{x} \text{ if } \forall i, x_i \text{ leaks}, \bot \text{ otherwise})$$

Problem: simulation may be inconsistent with other parts of the circuit.

Solution: give the simulator the full encoded input $\hat{x}$ if the original input x leaks.

Problem: what about gates with fan-in > 1?

Solution: if $x \in \mathbb{F}^k$, the simulator gets $\hat{x} \in \mathbb{F}^{2k}$ only if random probing was successful on all components of x, and $\bot$ otherwise.

Our compiler

Gadget simulation

For 1 input wire:

$$\mathsf{Real}_g(\hat{x}) \equiv \mathsf{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise})$$

Our compiler

Gadget simulation

For 1 input wire:

$$\mathsf{Real}_g\left(\hat{x}\right) \equiv \mathsf{Sim}_g\left(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise}\right)$$
$$\iff \mathbb{P}\left[\mathsf{Real}_g\left(\hat{x}\right) = \Lambda\right] = \mathbb{P}\left[\mathsf{Sim}_g\left(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise}\right) = \Lambda\right]$$

Our compiler

Gadget simulation

For 1 input wire:

$$\mathsf{Real}_g(\hat{x}) \equiv \mathsf{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise})$$

$$\iff \mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda] = \mathbb{P}[\mathsf{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise}) = \Lambda]$$

$$\iff \mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda] = \varepsilon \ \mathbb{P}[\mathsf{Sim}_g(\hat{x}) = \Lambda] + (1 - \varepsilon \ )\mathbb{P}[\mathsf{Sim}_g(\perp) = \Lambda]$$

ε-random-probing

Our compiler

Gadget simulation

For 1 input wire:

$$\mathsf{Real}_g(\hat{x}) \equiv \mathsf{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \bot \text{ otherwise})$$

$$\iff \mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda] = \mathbb{P}[\mathsf{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \bot \text{ otherwise}) = \Lambda]$$

$$\iff \mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda] = \varepsilon\ \mathbb{P}[\mathsf{Sim}_g(\hat{x}) = \Lambda] + (1 - \varepsilon\ )\mathbb{P}[\mathsf{Sim}_g(\bot) = \Lambda] \qquad \varepsilon\text{-random-probing}$$

Case $\Lambda \neq \bot$: $\mathbb{P}[\mathsf{Sim}_g(\bot) = \Lambda] = 0$

Case $\Lambda = \bot$: $\mathbb{P}[\mathsf{Sim}_g(\bot) = \bot] = 1$

Gadget simulation

For 1 input wire:

$$\text{Real}_g(\hat{x}) \equiv \text{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise})$$

$$\iff \mathbb{P}[\text{Real}_g(\hat{x}) = \Lambda] = \mathbb{P}[\text{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise}) = \Lambda]$$

$$\iff \mathbb{P}[\text{Real}_g(\hat{x}) = \Lambda] = \varepsilon \ \mathbb{P}[\text{Sim}_g(\hat{x}) = \Lambda] + (1 - \varepsilon \ )\mathbb{P}[\text{Sim}_g(\perp) = \Lambda]$$ ε-random-probing

Case $\Lambda \neq \perp$: $\mathbb{P}[\text{Sim}_g(\perp) = \Lambda] = 0$

$$\implies \mathbb{P}[\text{Sim}_g(\hat{x}) = \Lambda] = \frac{1}{\varepsilon}\mathbb{P}[\text{Real}_g(\hat{x}) = \Lambda]$$

Case $\Lambda = \perp$: $\mathbb{P}[\text{Sim}_g(\perp) = \perp] = 1$

Gadget simulation

For 1 input wire:

$$\mathsf{Real}_g(\hat{x}) \equiv \mathsf{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \bot \text{ otherwise})$$

$$\iff \mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda] = \mathbb{P}[\mathsf{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \bot \text{ otherwise}) = \Lambda]$$

$$\iff \mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda] = \varepsilon\ \mathbb{P}[\mathsf{Sim}_g(\hat{x}) = \Lambda] + (1 - \varepsilon\ )\mathbb{P}[\mathsf{Sim}_g(\bot) = \Lambda]$$

ε-random-probing

Case $\Lambda \neq \bot$: $\mathbb{P}[\mathsf{Sim}_g(\bot) = \Lambda] = 0$

$$\implies \mathbb{P}[\mathsf{Sim}_g(\hat{x}) = \Lambda] = \frac{1}{\varepsilon}\mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda]$$

Case $\Lambda = \bot$: $\mathbb{P}[\mathsf{Sim}_g(\bot) = \bot] = 1$

$$\implies \mathbb{P}[\mathsf{Sim}_g(\hat{x}) = \bot] = \frac{\mathbb{P}[\mathsf{Real}_g(\hat{x}) = \bot] - (1 - \varepsilon\ )}{\varepsilon}$$

Our compiler

Gadget simulation

For 1 input wire:

$$\mathsf{Real}_g(\hat{x}) \equiv \mathsf{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise})$$
$$\iff \mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda] = \mathbb{P}[\mathsf{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise}) = \Lambda]$$
$$\iff \mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda] = \varepsilon\ \mathbb{P}[\mathsf{Sim}_g(\hat{x}) = \Lambda] + (1-\varepsilon\ )\mathbb{P}[\mathsf{Sim}_g(\perp) = \Lambda]$$

ε-random-probing

Case $\Lambda \neq \perp$: $\mathbb{P}[\mathsf{Sim}_g(\perp) = \Lambda] = 0$

$$\implies \mathbb{P}[\mathsf{Sim}_g(\hat{x}) = \Lambda] = \frac{1}{\varepsilon}\mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda]$$

Case $\Lambda = \perp$: $\mathbb{P}[\mathsf{Sim}_g(\perp) = \perp] = 1$

$$\implies \mathbb{P}[\mathsf{Sim}_g(\hat{x}) = \perp] = \frac{\mathbb{P}[\mathsf{Real}_g(\hat{x}) = \perp] - (1-\varepsilon\ )}{\varepsilon}$$

must be ≥ 0

Our compiler

Gadget simulation

For 1 input wire:

$$\mathsf{Real}_g(\hat{x}) \equiv \mathsf{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise})$$

$$\iff \mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda] = \mathbb{P}[\mathsf{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise}) = \Lambda]$$

$$\iff \mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda] = \varepsilon \ \mathbb{P}[\mathsf{Sim}_g(\hat{x}) = \Lambda] + (1 - \varepsilon\ )\mathbb{P}[\mathsf{Sim}_g(\perp) = \Lambda]$$

ε-random-probing

Case $\Lambda \neq \perp$: $\mathbb{P}[\mathsf{Sim}_g(\perp) = \Lambda] = 0$

$$\implies \mathbb{P}[\mathsf{Sim}_g(\hat{x}) = \Lambda] = \frac{1}{\varepsilon}\mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda]$$

Case $\Lambda = \perp$: $\mathbb{P}[\mathsf{Sim}_g(\perp) = \perp] = 1$

$$\implies \mathbb{P}[\mathsf{Sim}_g(\hat{x}) = \perp] = \frac{\mathbb{P}[\mathsf{Real}_g(\hat{x}) = \perp] - (1 - \varepsilon\ )}{\varepsilon}$$

must be ≥ 0

$$\implies \mathbb{P}[\mathsf{Real}_g(\hat{x}) = \perp] \geq 1 - \varepsilon \quad \implies \mathbb{P}[\mathsf{Real}_g(\hat{x}) \neq \perp] \leq \varepsilon$$

Our compiler

Gadget simulation

For 2 input wires:

$$\mathsf{Real}_g(\hat{x}) \equiv \mathsf{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise})$$

$$\iff \mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda] = \mathbb{P}[\mathsf{Sim}_g(\hat{x} \text{ if } x \text{ leaks}, \perp \text{ otherwise}) = \Lambda]$$

$$\iff \mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda] = \varepsilon^2 \mathbb{P}[\mathsf{Sim}_g(\hat{x}) = \Lambda] + (1-\varepsilon^2)\mathbb{P}[\mathsf{Sim}_g(\perp) = \Lambda]$$

ε-random-probing

Case $\Lambda \neq \perp$: $\mathbb{P}[\mathsf{Sim}_g(\perp) = \Lambda] = 0$

$$\implies \mathbb{P}[\mathsf{Sim}_g(\hat{x}) = \Lambda] = \frac{1}{\varepsilon^2}\mathbb{P}[\mathsf{Real}_g(\hat{x}) = \Lambda]$$

Case $\Lambda = \perp$: $\mathbb{P}[\mathsf{Sim}_g(\perp) = \perp] = 1$

$$\implies \mathbb{P}[\mathsf{Sim}_g(\hat{x}) = \perp] = \frac{\mathbb{P}[\mathsf{Real}_g(\hat{x}) = \perp] - (1-\varepsilon^2)}{\varepsilon^2}$$

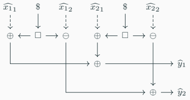
must be ≥ 0

$$\implies \mathbb{P}[\mathsf{Real}_g(\hat{x}) = \perp] \geq 1 - \varepsilon^2 \implies \mathbb{P}[\mathsf{Real}_g(\hat{x}) \neq \perp] \leq \varepsilon^2$$

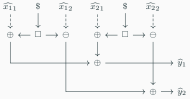
Our compiler

Our gadgets

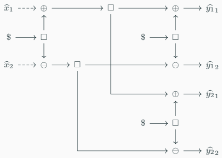
Simulation condition: $\mathbb{P}\left[\text{Real}_g\left(\hat{x}\right) \neq \perp\right] \leq \varepsilon^2$.



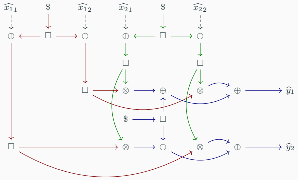
Addition gadget



Subtraction gadget



Copy gadget

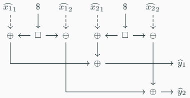


Multiplication gadget

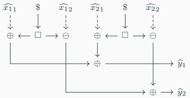
Our compiler

Our gadgets

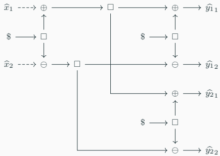
Simulation condition: $\mathbb{P}\left[\mathsf{Real}_g\left(\hat{x}\right) \neq \bot\right] \leq \varepsilon^2$.
Most complex gadget: multiplication gadget, with $\mathbb{P}\left[\mathsf{Real}_g\left(\hat{x}\right) \neq \bot\right] \leq 35\delta$.



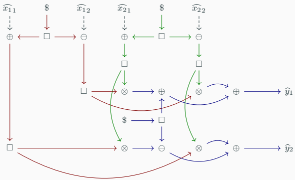
Addition gadget



Subtraction gadget



Copy gadget



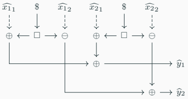
Multiplication gadget

Our compiler

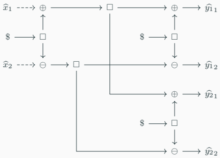
Our gadgets

Simulation condition: $35\delta \leq \varepsilon^2$.

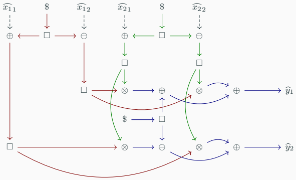
Most complex gadget: multiplication gadget, with $\mathbb{P}\left[\text{Real}_g\left(\hat{x}\right) \neq \bot\right] \leq 35\delta$.



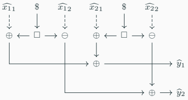
Addition gadget



Copy gadget



Multiplication gadget

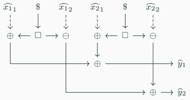


Subtraction gadget

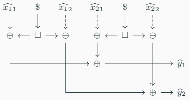
Our compiler

Our gadgets

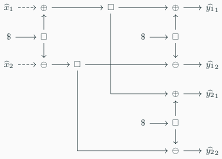
Simulation condition: $35\delta \leq \varepsilon^2$.
Most complex gadget: multiplication gadget, with $\mathbb{P}\left[\text{Real}_g(\hat{x}) \neq \perp\right] \leq 35\delta$.



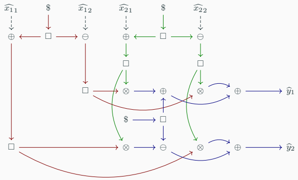
Addition gadget



Subtraction gadget



Copy gadget



Multiplication gadget

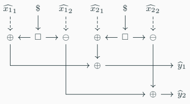
Main theorem: there exists a circuit compiler that compiles ε-RP secure circuits into $\frac{\varepsilon^2}{35}$-AP secure circuits.

Our compiler

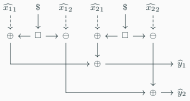
Our gadgets

Simulation condition: $35\delta \leq \varepsilon^2$.

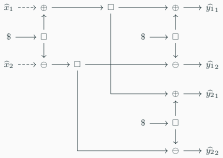
Most complex gadget: multiplication gadget, with $\mathbb{P}\left[\mathrm{Real}_g\left(\hat{x}\right) \neq \bot\right] \leq 35\delta$.



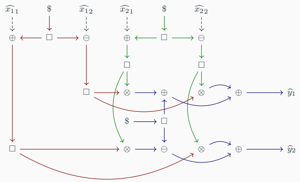
Addition gadget



Subtraction gadget



Copy gadget



Multiplication gadget

Main theorem: there exists a circuit compiler that compiles ε-RP secure circuits into $\frac{\varepsilon^2}{35}$-AP secure circuits.

Corollary: by applying [DFS15], there exists a circuit compiler that compiles ε-RP secure circuits into $\frac{\varepsilon^2}{35}$-SD-noisy-leakage resilient circuits.

Conclusions

Summary of our results

- We construct a compiler from ε-random probing to $\frac{\varepsilon^2}{35}$-average probing.
- By applying [DFS15], we get a compiler from ε-random probing to $\frac{\varepsilon^2}{35}$-noisy leakage.

Conclusions

Summary of our results

- We construct a compiler from ε-random probing to $\frac{\varepsilon^2}{35}$-average probing.
- By applying [DFS15], we get a compiler from ε-random probing to $\frac{\varepsilon^2}{35}$-noisy leakage.

Open problems

- Is it possible to do better (i.e., get rid of the square)?

Conclusions

Summary of our results

- We construct a compiler from ε-random probing to $\frac{\varepsilon^2}{35}$-average probing.
- By applying [DFS15], we get a compiler from ε-random probing to $\frac{\varepsilon^2}{35}$-noisy leakage.

Open problems

- Is it possible to do better (i.e., get rid of the square)?
- Is it possible to extend the result to more general types of circuits (e.g., arithmetic circuit over *rings*)?

Conclusions

Summary of our results

- We construct a compiler from ε-random probing to $\frac{\varepsilon^2}{35}$-average probing.
- By applying [DFS15], we get a compiler from ε-random probing to $\frac{\varepsilon^2}{35}$-noisy leakage.

Open problems

- Is it possible to do better (i.e., get rid of the square)?
- Is it possible to extend the result to more general types of circuits (e.g., arithmetic circuit over *rings*)?
- Can the same technique be used to achieve other interesting results?

Conclusions

Summary of our results

- We construct a compiler from ε-random probing to $\frac{\varepsilon^2}{35}$-average probing.
- By applying [DFS15], we get a compiler from ε-random probing to $\frac{\varepsilon^2}{35}$-noisy leakage.

Open problems

- Is it possible to do better (i.e., get rid of the square)?
- Is it possible to extend the result to more general types of circuits (e.g., arithmetic circuit over *rings*)?
- Can the same technique be used to achieve other interesting results?

Thank you!