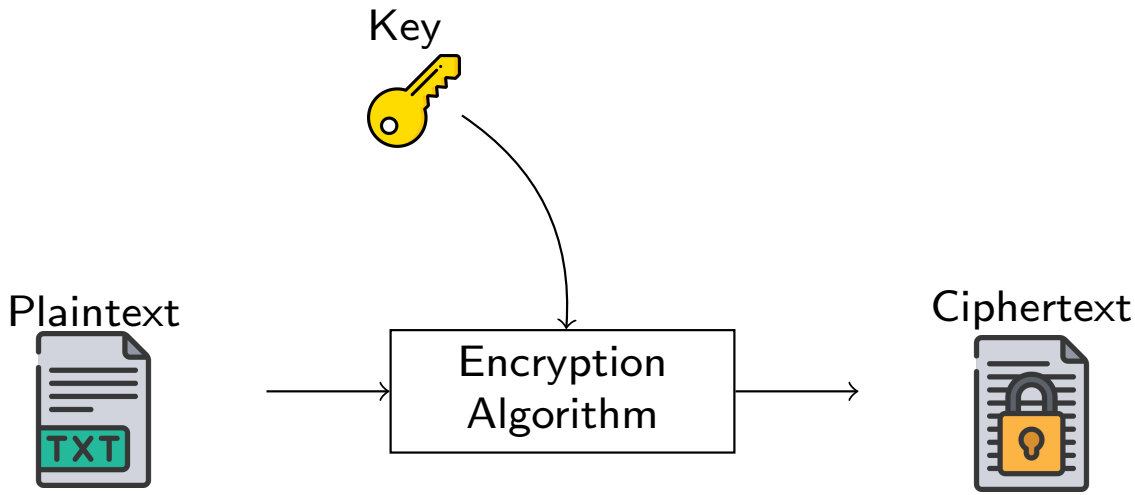
Masked Circuit Compiler in the Cardinal Random Probing Composability Framework

Sonia Belaïd, **Victor Normand**, Matthieu Rivain

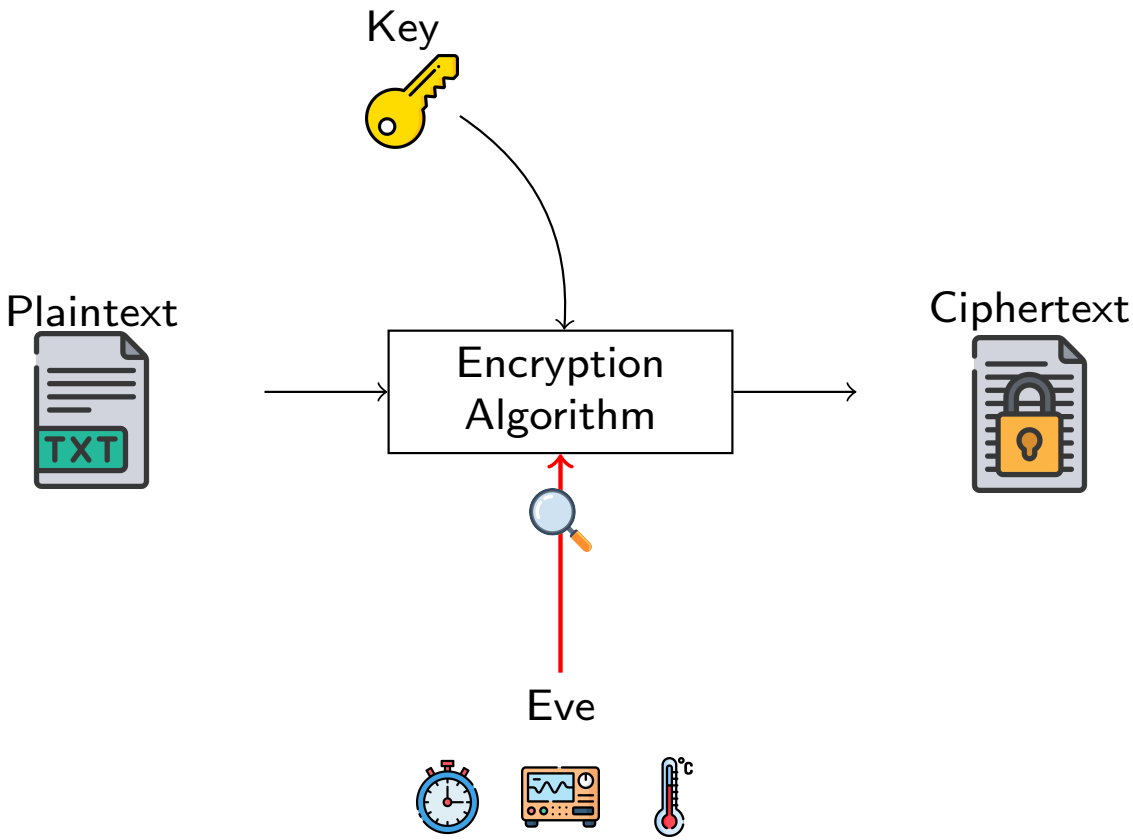
Almasty Seminar

May 5, 2026

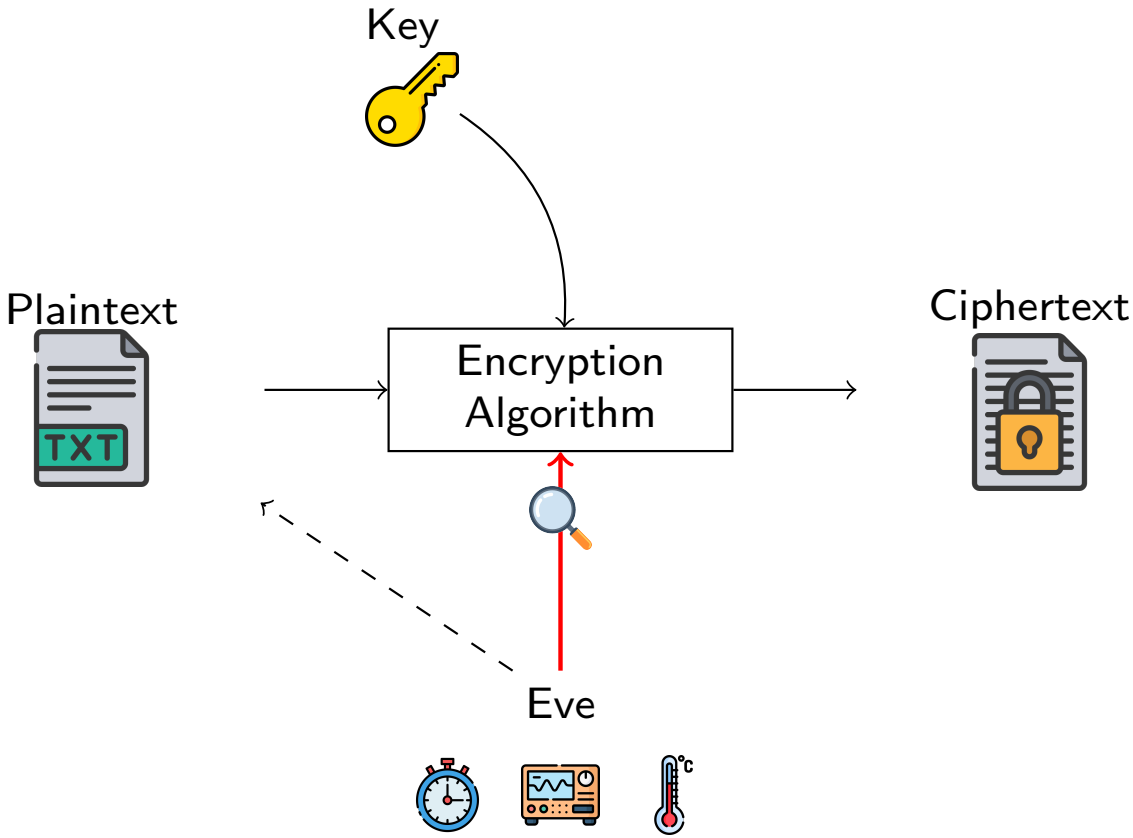
Side Channel Attacks



Side Channel Attacks



Side Channel Attacks



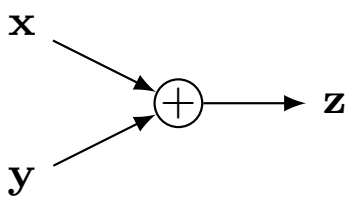
Masking countermeasure

- Replace each sensitive value $\mathbf{x}$ of the Algorithm by a tuple $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.
- Any set of $(n-1)$ shares is independent from $\mathbf{x}$.
- Arithmetic masking :

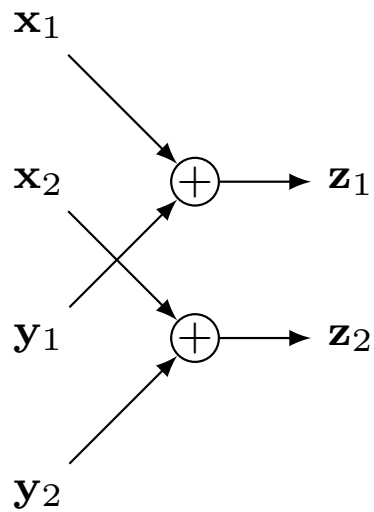
$$\mathbf{x} = \sum_{i=1}^{n} \mathbf{x}_i$$

Gadget

- Operations on sensitives values $\Longrightarrow$ Operations on their respective shares.
- Secret variable : $\mathbf{x}$ and $\mathbf{y}$.

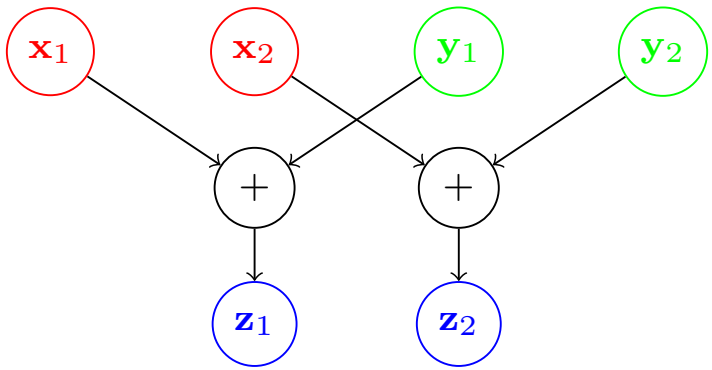


(a) Non masked version



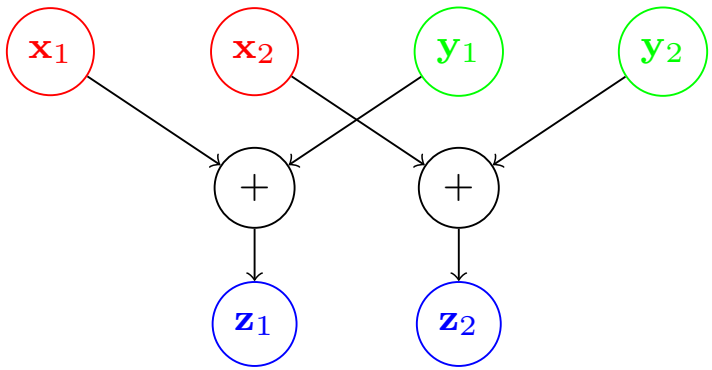
(b) Masked version with 2 shares.

Circuits



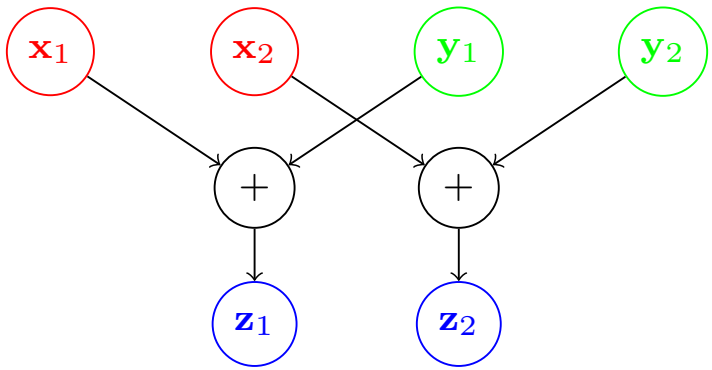
- **Circuits**
  - Directed acyclic graph.
  - Each vertex is an input/output share or arithmetic gate.
  - Each edge is a wire of the circuit.

t-probing model



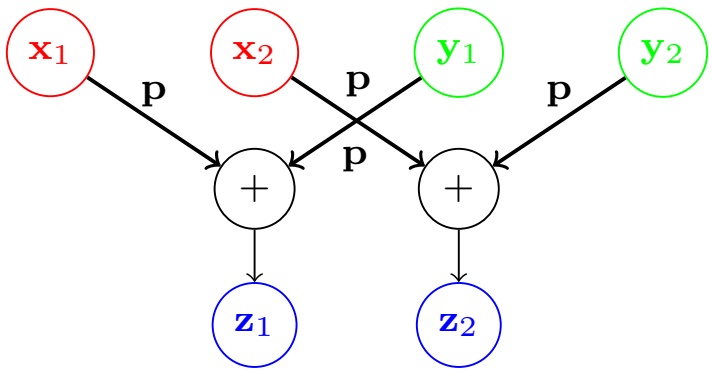
- **t-Probing Model**
  - Up to t wires of the circuits leak their value.
  - Secure if all leakages sets is independant from the secrets.

(p,ε)-random-probing-model



- **(p, ε)-Random-Probing Model**
  - Every wire leaks its value with a probability $p \in [0, 1]$.
  - Secure if there is a negligible probability ($\leq \varepsilon$) to get information on the secrets.

(p,ε)-random-probing-model



$(\mathbf{x}_1, \mathbf{x}_2)$ leaks secret $\mathbf{x}$.
$(\mathbf{y}_1, \mathbf{y}_2)$ leaks secret $\mathbf{y}$.

$$\varepsilon = 2p^2(1-p)^2 + 4p^3(1-p) + p^4$$

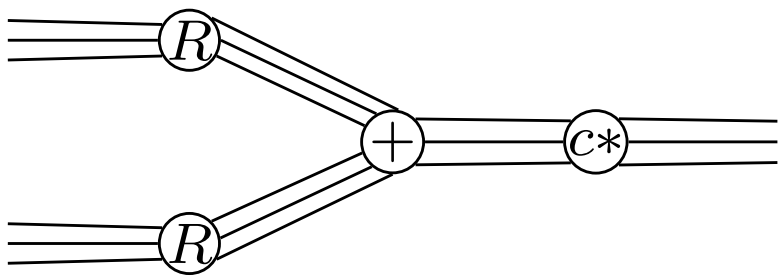
Composability notions in Random Probing Model

- Computing ε is challenging for big circuit.
- Composability notion : Divide circuit and prove a stronger notion.

Threshold Random Probing Composability

(t, p, ε)-tRPC :

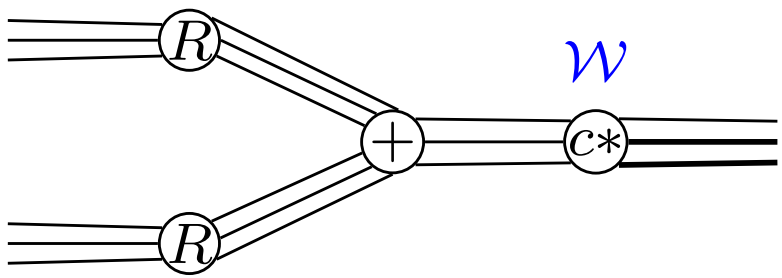
- **Simulate** the internal leakage $\mathcal{W}$ and t output shares with **at most** t input shares of each secret variable.
- Error probability ε.



Threshold Random Probing Composability

(t, p, ε)-tRPC :

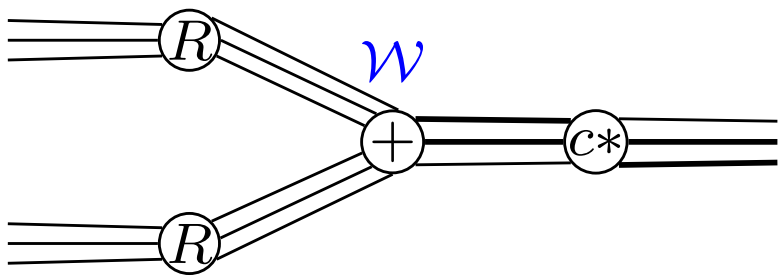
- **Simulate** the internal leakage $\mathcal{W}$ and t output shares with **at most** t input shares of each secret variable.
- Error probability ε.



Threshold Random Probing Composability

(t, p, ε)-tRPC :

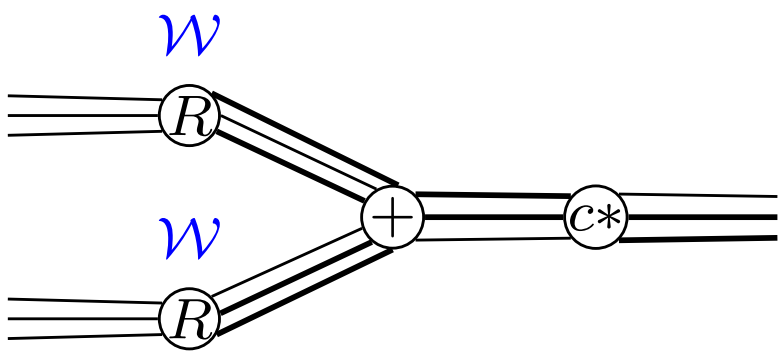
- **Simulate** the internal leakage $\mathcal{W}$ and t output shares with **at most** t input shares of each secret variable.
- Error probability ε.



Threshold Random Probing Composability

(t, p, ε)-tRPC :

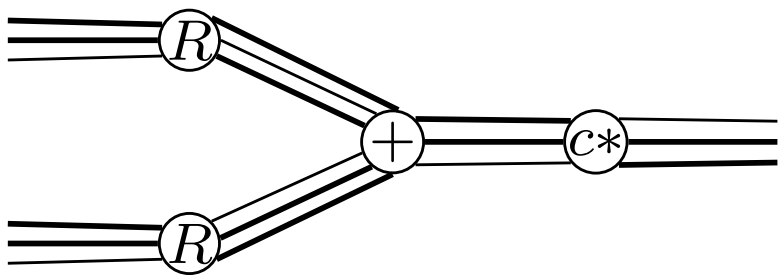
- **Simulate** the internal leakage $\mathcal{W}$ and t output shares with **at most** t input shares of each secret variable.
- Error probability ε.



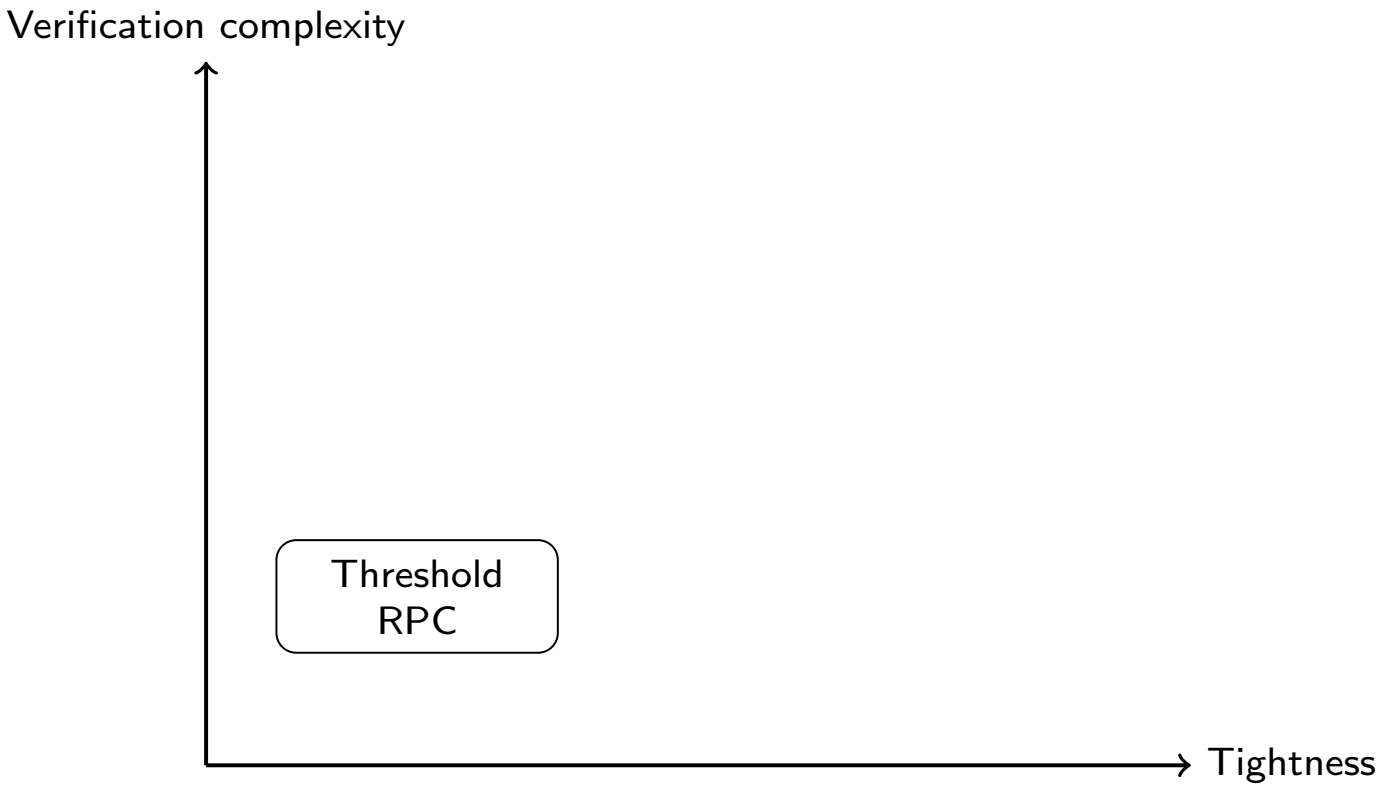
Threshold Random Probing Composability

(t, p, ε)-tRPC :

- **Simulate** the internal leakage $\mathcal{W}$ and t output shares with **at most** t input shares of each secret variable.
- Error probability ε.

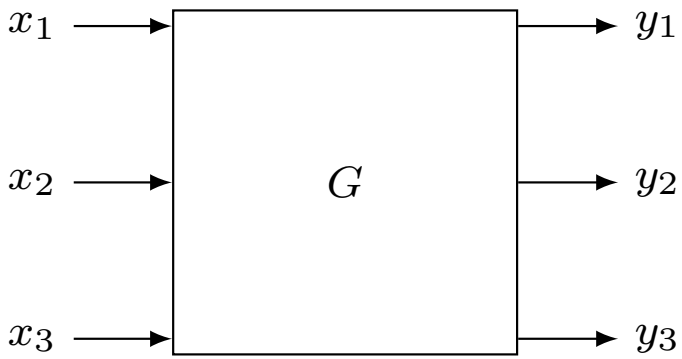


Threshold Random Probing Composability



General Random Probing Composability

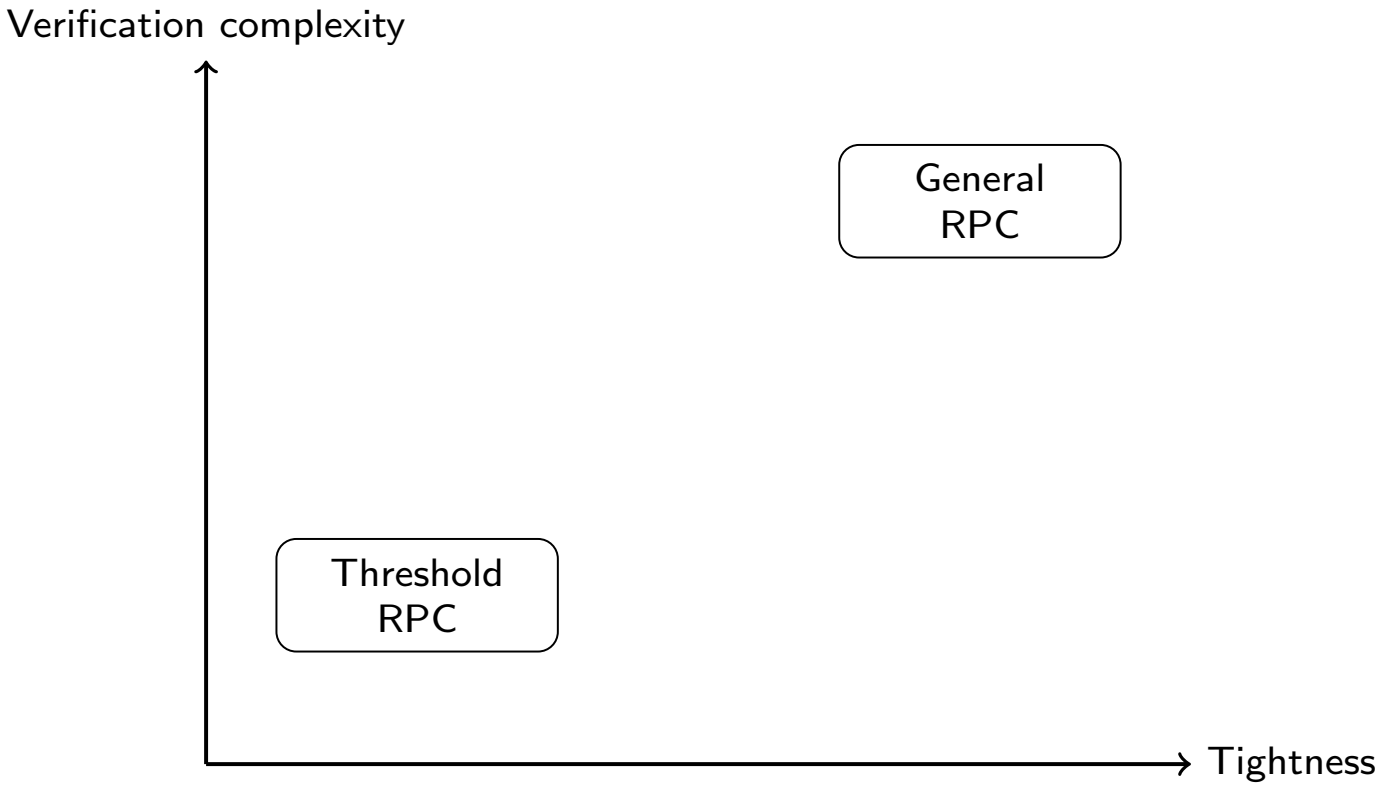
Probe Distribution Table = Probability that we requires the set I of input shares to simulate the internal leakages and the output shares in a set J.
Example : A 3-sharing (x_1, x_2, x_3) of a secret x.



I \ J	$\emptyset$	$\{y_1\}$	$\{y_2\}$	$\{y_3\}$	$\{y_1, y_2\}$	$\{y_1, y_3\}$	$\{y_2, y_3\}$	$\{y_1, y_2, y_3\}$
$\emptyset$								
$\{x_1\}$								
$\{x_2\}$								
$\{x_3\}$								
$\{x_1, x_2\}$								
$\{x_1, x_3\}$								
$\{x_2, x_3\}$								
$\{x_1, x_2, x_3\}$								

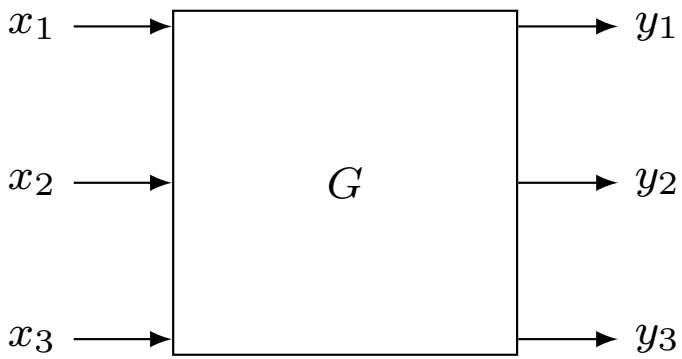
General Random Probing Composability

Probe Distribution Table = Probability that we require the set I of input shares to simulate the internal leakages and the output shares in a set J.



Cardinal Random Probing Composability

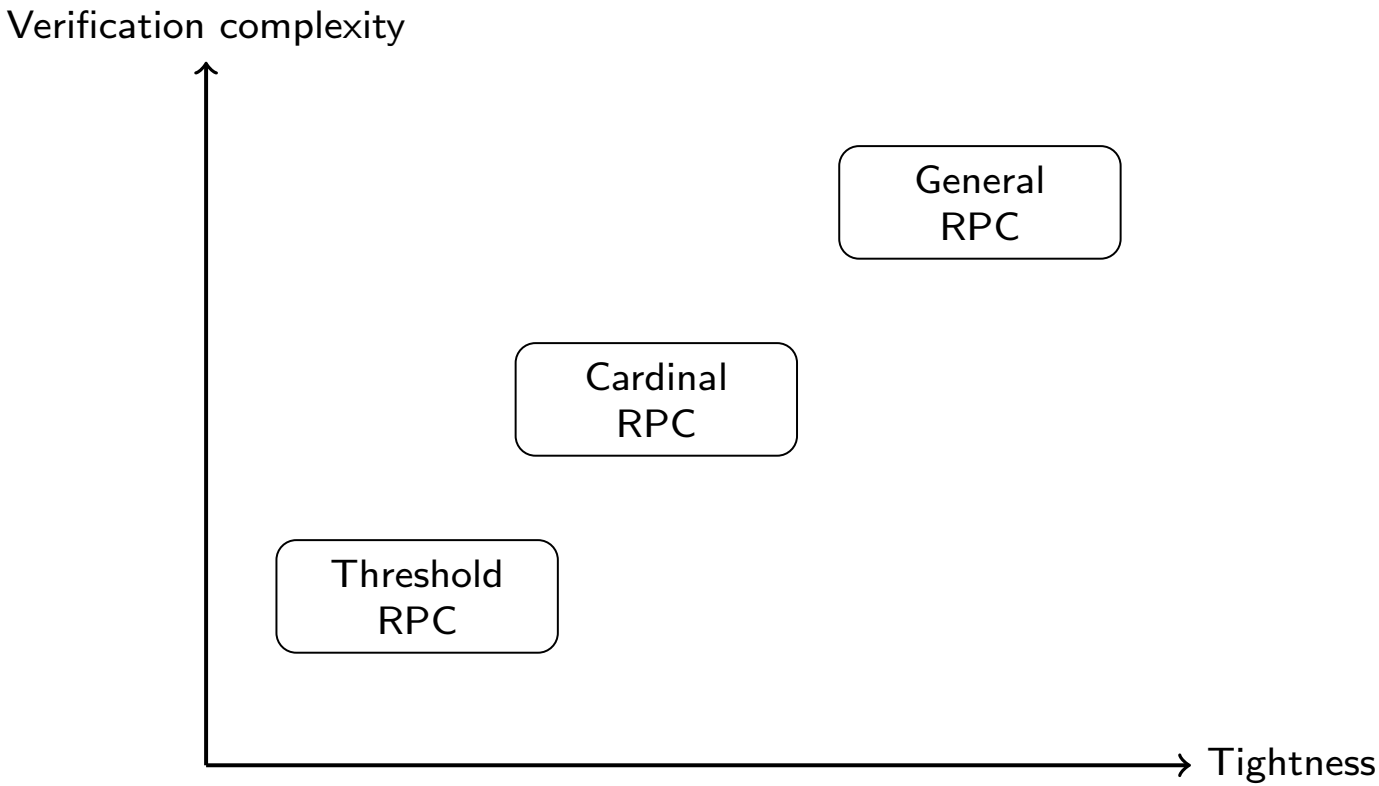
Replace input/output sets with their cardinalities.



I \ J	0	1	2	3
0				
1				
2				
3				

Cardinal Random Probing Composability

Replace input/output sets with their cardinalities.



Uniform Cardinal Random Probing Composability

- Best of both world : Tight security from General RPC and complexity from Cardinal RPC.
- One Condition :

$$\xi^g_{(J_1,\ldots,J_m)}(I_1,\ldots,I_\ell) = \xi^g_{(J'_1,\ldots,J'_m)}(I'_1,\ldots,I'_\ell)$$

when

$$(|I_1|,\ldots,|I_\ell|) = (|I'_1|,\ldots,|I'_\ell|)$$
$$(|J_1|,\ldots,|J_m|) = (|J'_1|,\ldots,|J'_m|)$$

Uniform Cardinal Random Probing Composability

- Best of both world : Tight security from General RPC and complexity from Cardinal RPC.
- One Condition :

$$\xi^g_{(J_1,\ldots,J_m)}(I_1,\ldots,I_\ell) = \xi^g_{(J'_1,\ldots,J'_m)}(I'_1,\ldots,I'_\ell)$$

when

$$(|I_1|,\ldots,|I_\ell|) = (|I'_1|,\ldots,|I'_\ell|)$$
$$(|J_1|,\ldots,|J_m|) = (|J'_1|,\ldots,|J'_m|)$$

I \ J	$\emptyset$	$\{y_1\}$	$\{y_2\}$	$\{y_3\}$	$\{y_1,y_2\}$	$\{y_1,y_3\}$	$\{y_2,y_3\}$	$\{y_1,y_2,y_3\}$
$\emptyset$								
$\{x_1\}$								
$\{x_2\}$								
$\{x_3\}$								
$\{x_1,x_2\}$								
$\{x_1,x_3\}$								
$\{x_2,x_3\}$								
$\{x_1,x_2,x_3\}$								

I \ J	0	1	2	3
0				
1				
2				
3				

Uniform Cardinal Random Probing Composability

- Best of both world : Tight security from General RPC and complexity from Cardinal RPC.
- One Condition :

$$\xi^g_{(J_1,\ldots,J_m)}(I_1,\ldots,I_\ell) = \xi^g_{(J'_1,\ldots,J'_m)}(I'_1,\ldots,I'_\ell)$$

when

$$(|I_1|,\ldots,|I_\ell|) = (|I'_1|,\ldots,|I'_\ell|)$$
$$(|J_1|,\ldots,|J_m|) = (|J'_1|,\ldots,|J'_m|)$$

I \ J	$\emptyset$	$\{y_1\}$	$\{y_2\}$	$\{y_3\}$	$\{y_1,y_2\}$	$\{y_1,y_3\}$	$\{y_2,y_3\}$	$\{y_1,y_2,y_3\}$
$\emptyset$								
$\{x_1\}$								
$\{x_2\}$								
$\{x_3\}$								
$\{x_1,x_2\}$								
$\{x_1,x_3\}$								
$\{x_2,x_3\}$								
$\{x_1,x_2,x_3\}$								

I \ J	0	1	2	3
0				
1				
2				
3				

Uniform Cardinal Random Probing Composability

- Best of both world : Tight security from General RPC and complexity from Cardinal RPC.
- One Condition :

$$\xi^g_{(J_1,\ldots,J_m)}(I_1,\ldots,I_\ell) = \xi^g_{(J'_1,\ldots,J'_m)}(I'_1,\ldots,I'_\ell)$$

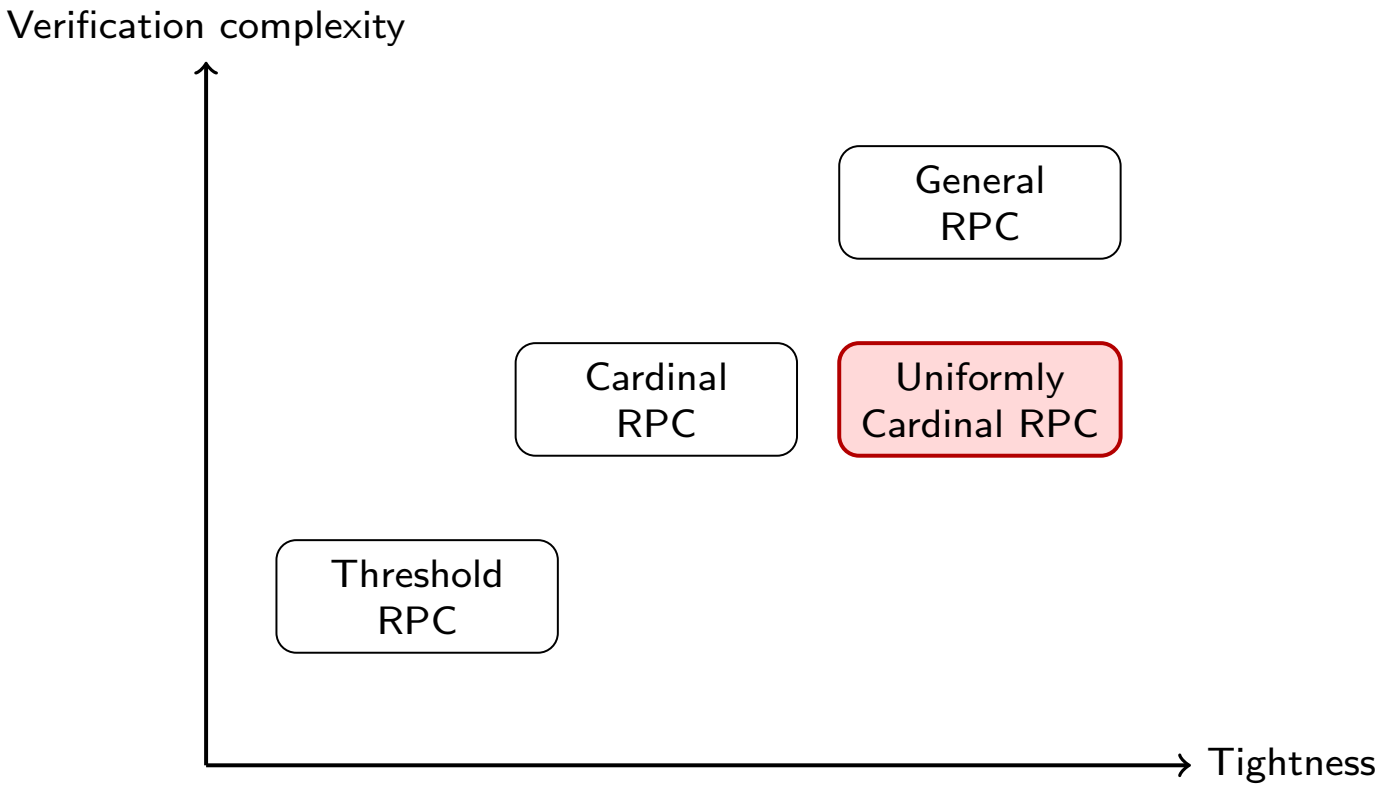
when

$$(|I_1|,\ldots,|I_\ell|) = (|I'_1|,\ldots,|I'_\ell|)$$
$$(|J_1|,\ldots,|J_m|) = (|J'_1|,\ldots,|J'_m|)$$

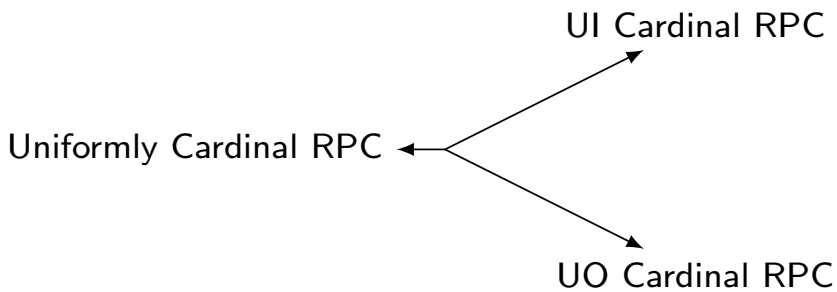
I \ J	$\emptyset$	$\{y_1\}$	$\{y_2\}$	$\{y_3\}$	$\{y_1,y_2\}$	$\{y_1,y_3\}$	$\{y_2,y_3\}$	$\{y_1,y_2,y_3\}$
$\emptyset$								
$\{x_1\}$								
$\{x_2\}$								
$\{x_3\}$								
$\{x_1,x_2\}$								
$\{x_1,x_3\}$								
$\{x_2,x_3\}$								
$\{x_1,x_2,x_3\}$								

I \ J	0	1	2	3
0				
1				
2				
3				

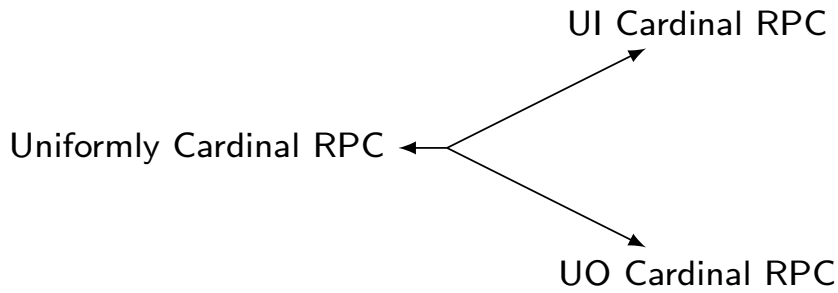
Uniform Cardinal Random PProbing Composability



UI/UO cardinal RPC



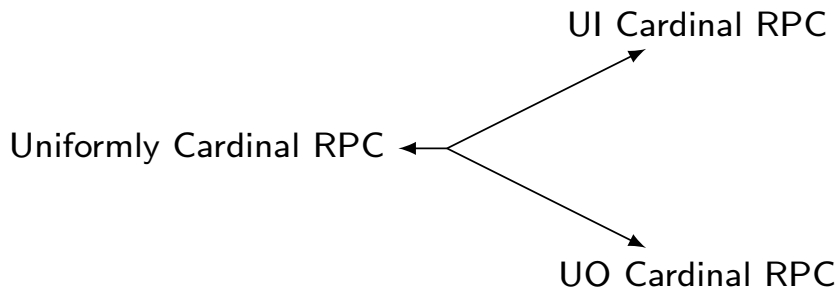
UI/UO cardinal RPC



- UI cardinal RPC, uniformity on the input shares **only**.

$$\xi^g_{(J_1,\ldots,J_m)}(I_1,\ldots,I_\ell) = \xi^g_{(J_1,\ldots,J_m)}(I'_1,\ldots,I'_\ell)$$

UI/UO cardinal RPC



- UI cardinal RPC, uniformity on the input shares **only**.

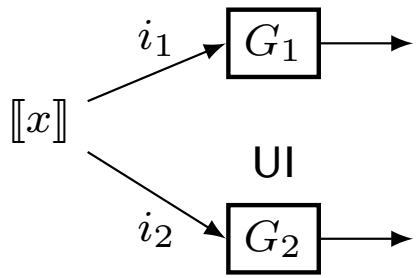
$$\xi^g_{(J_1,\ldots,J_m)}(I_1,\ldots,I_\ell) = \xi^g_{(J_1,\ldots,J_m)}(I'_1,\ldots,I'_\ell)$$

- UO cardinal RPC, uniformity on the output shares **only**.

$$\xi^g_{(J_1,\ldots,J_m)}(I_1,\ldots,I_\ell) = \xi^g_{(J'_1,\ldots,J'_m)}(I_1,\ldots,I_\ell)$$

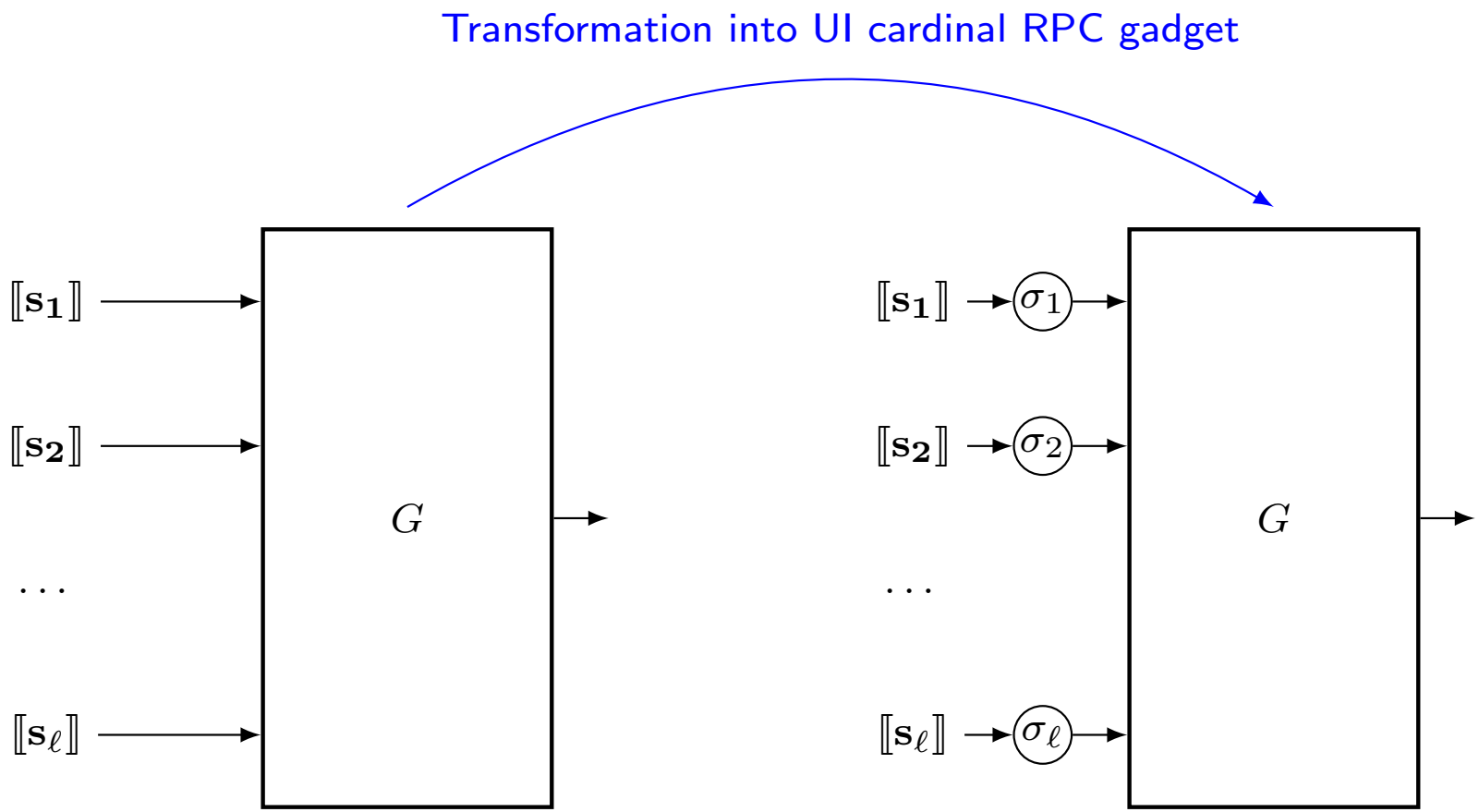
Utility of UI/UO

UI : Tighter bounds in case of **copy**.



UO : Gives **tighter** compositions.

Turn gadget UI cardinal RPC



Turn gadget UO cardinal RPC

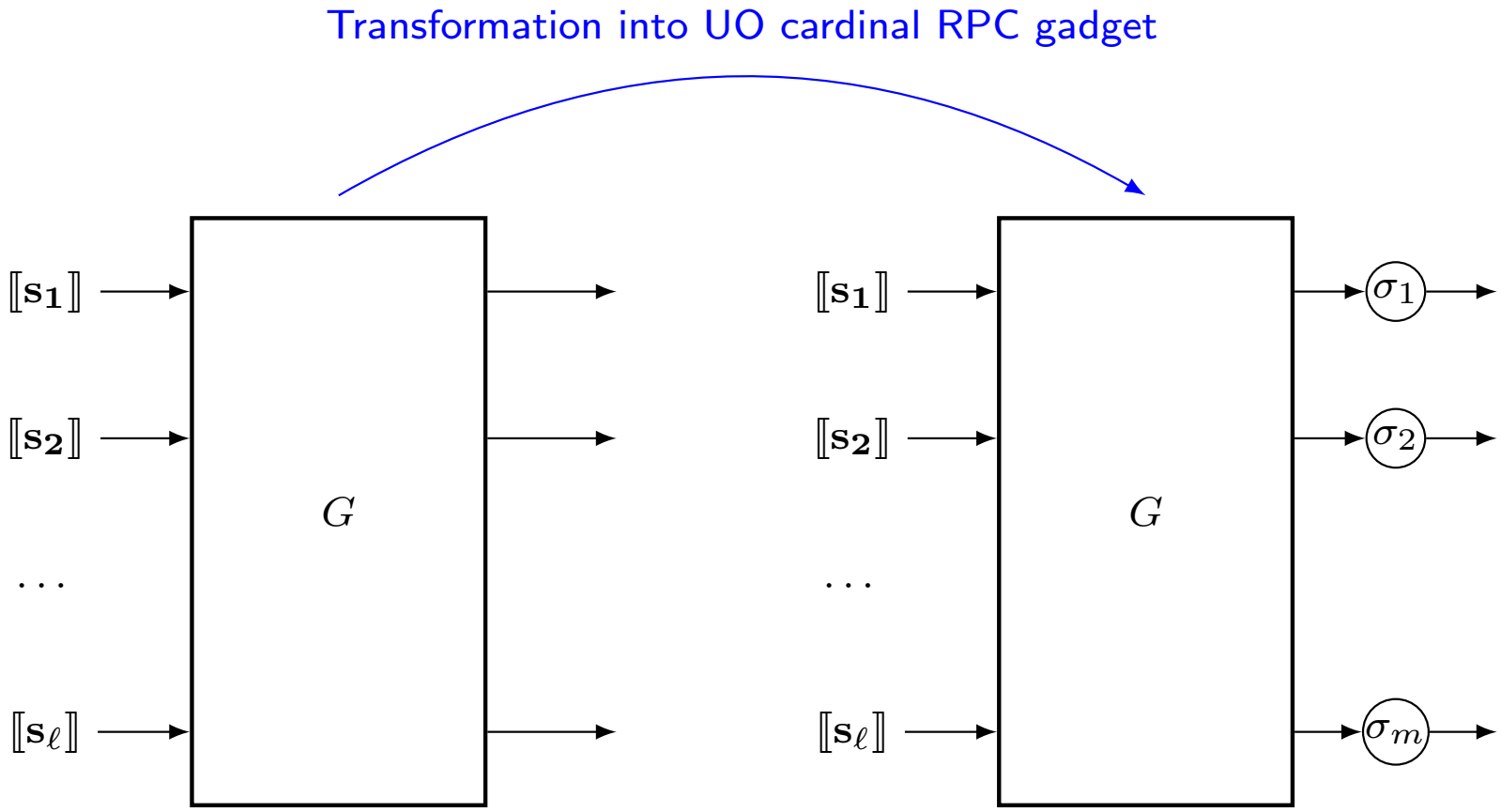
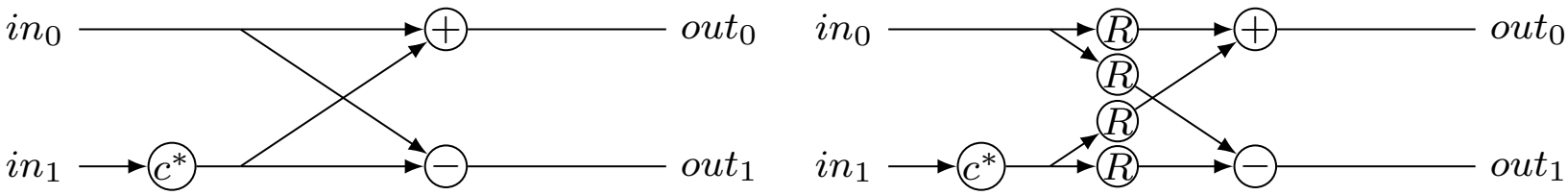
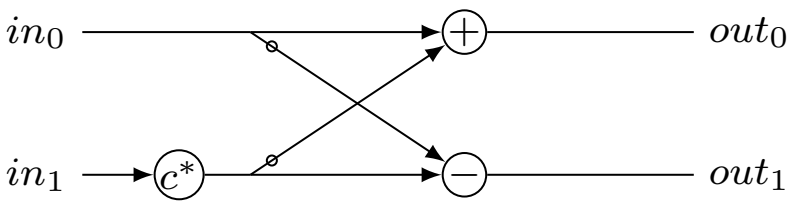


Illustration : 1-stage butterfly network

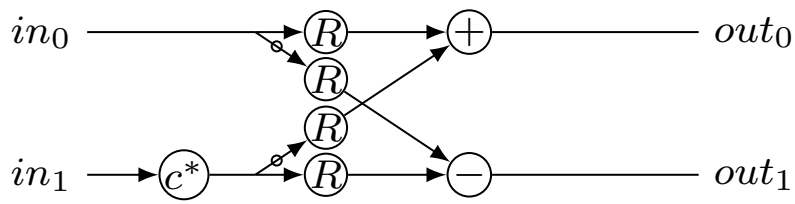


(a) Version 1

(b) Version 2



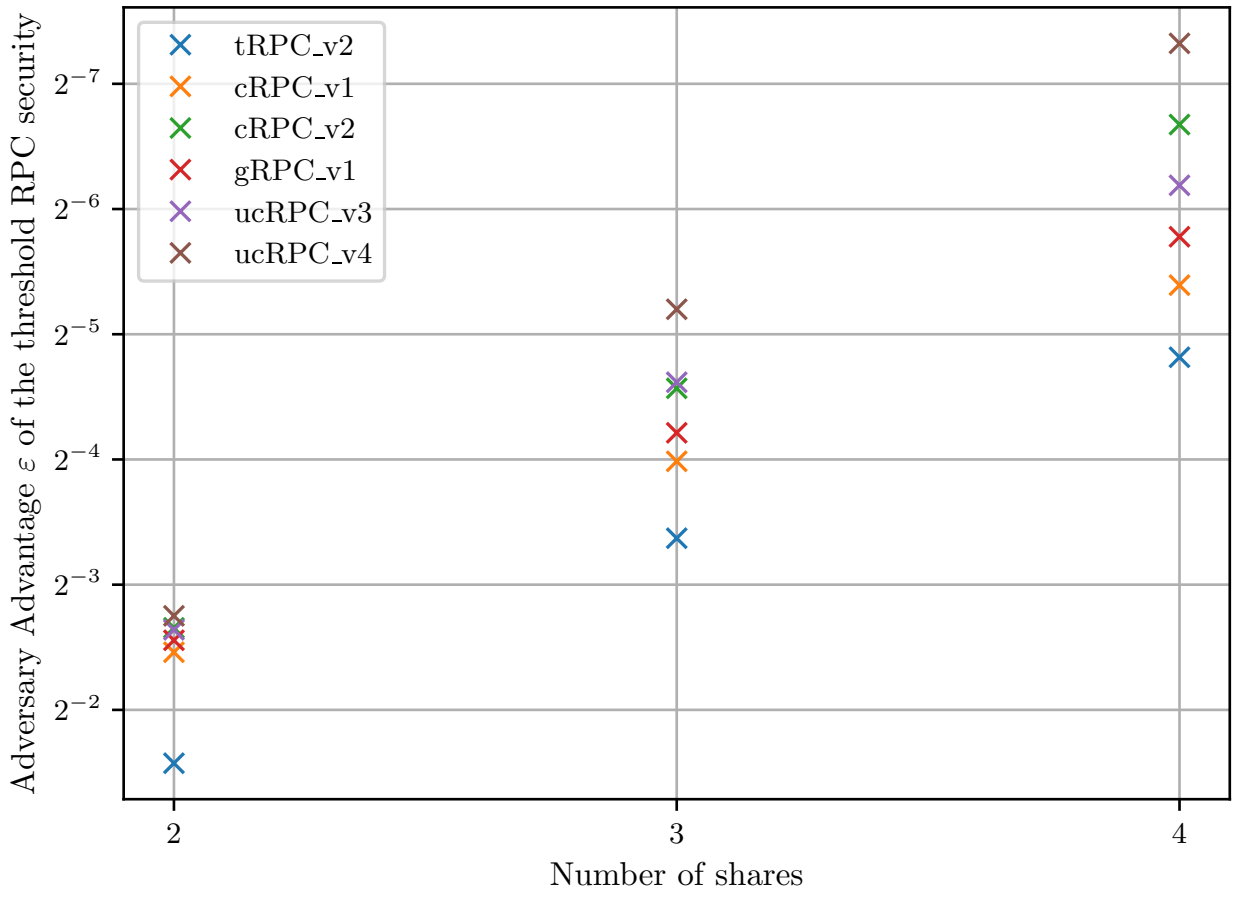
(c) Version 3



(d) Version 4

Figure: 1-stage butterfly network.

Illustration : Butterfly Network

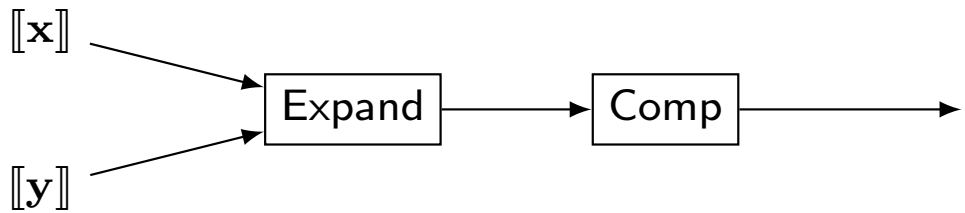


$$p = 2^{-5}$$

First contribution

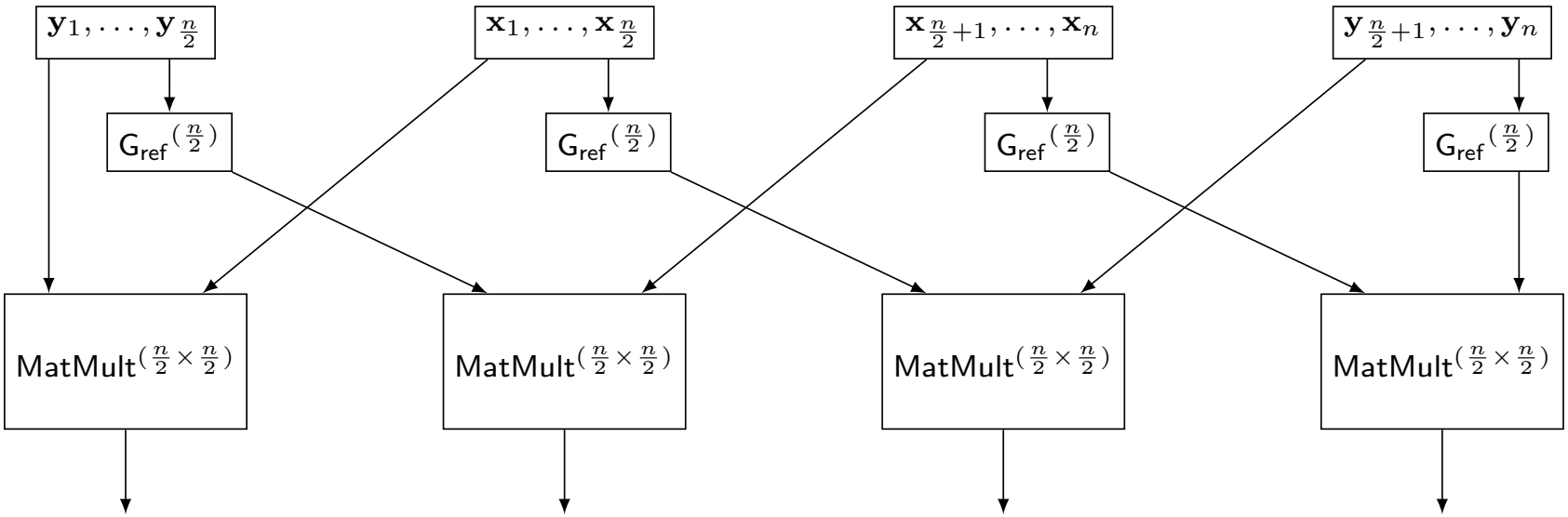
- Decomposition of the uniformly cardinal RPC property into UI cardinal and UO cardinal RPC.
- A simple technique to obtain UI/UO cardinal RPC envelopes from arbitrary gadgets by adding permutations.
- Illustration of the advantages and drawbacks of different composability notions on a 1-stage butterfly network.

Multiplication gadget

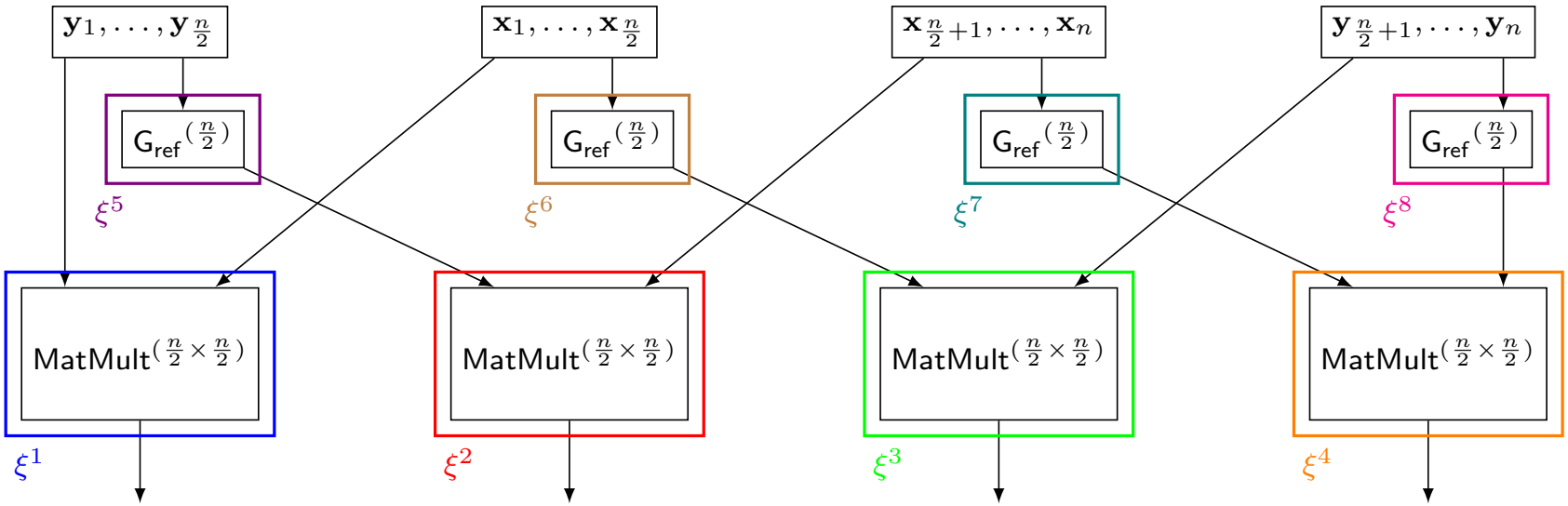


- Expand : From two n-sharing, compute the cross product $x_i \cdot y_j$ to obtain a n^2 sharing.
- Comp : Transform a n^2-sharing into a n-sharing.

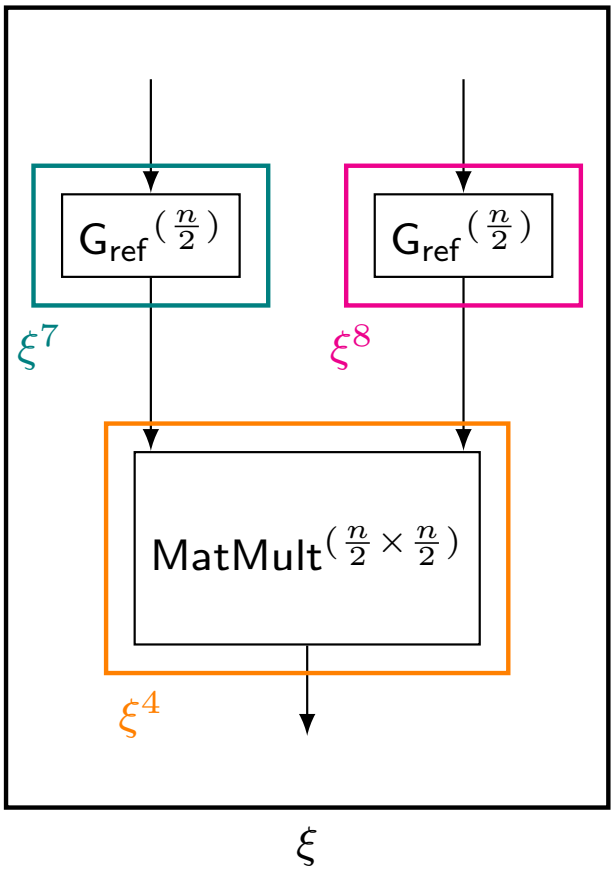
Expand : MatMultRef



Expand : MatMultRef

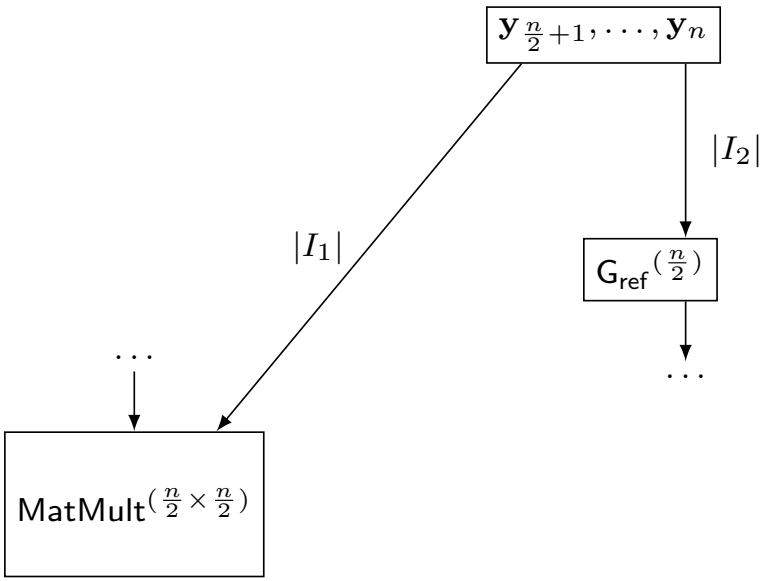


Expand : MatMultRef- Branch



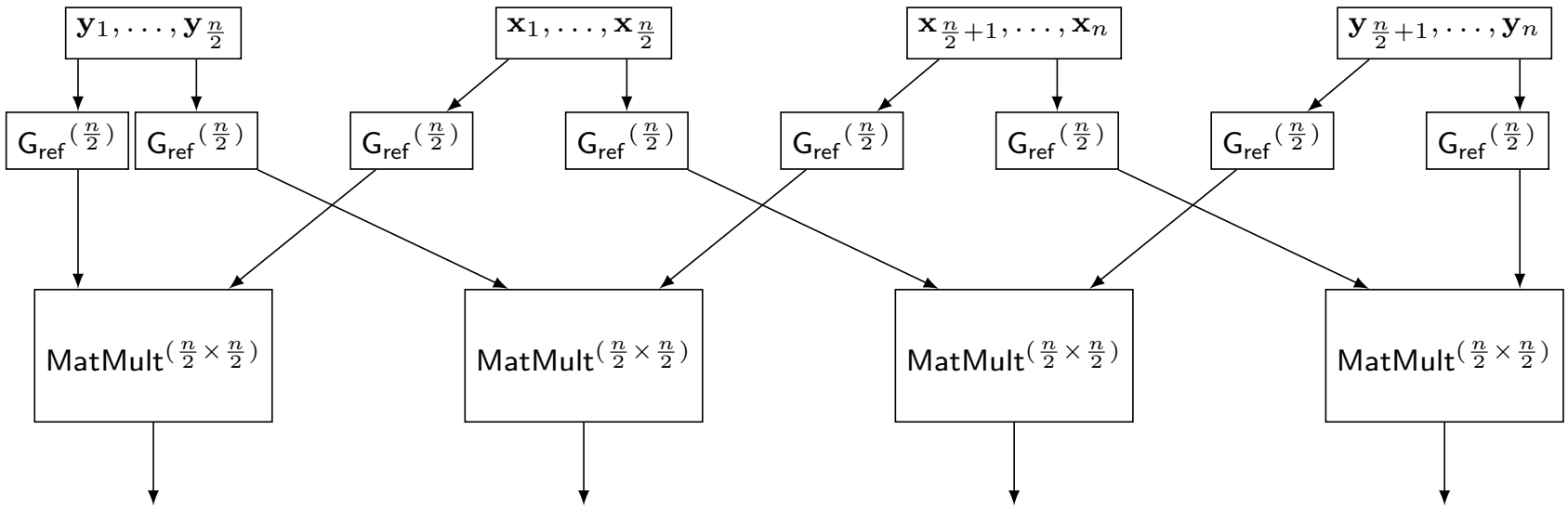
$$\xi_j(i_1,i_2) = \sum_{\ell_1=0}^{\frac{n}{2}} \sum_{\ell_2=0}^{\frac{n}{2}} {\xi^7}_{\ell_1}(i_1)\cdot{\xi^8}_{\ell_2}(i_2)\cdot{\xi^4}_{j}(\ell_1,\ell_2)$$

Expand : MatMultRef- Drawbacks

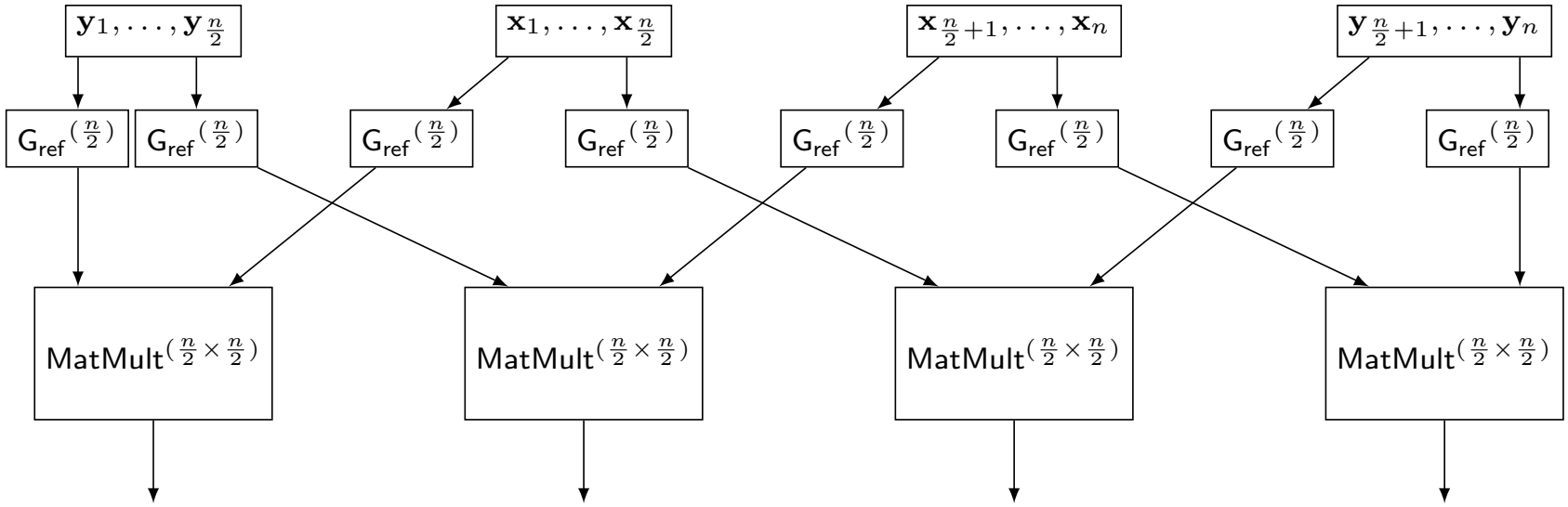


- $|I_1|$ required shares for the simulation of the first branch.
- $|I_2|$ required shares for the simulation of the second branch.
- **No information** on $|I_1 \cup I_2|$.
- Must consider the worst case: $|I_1 \cup I_2| = \min(\frac{n}{2}, |I_1| + |I_2|)$.
- **Looser Bounds**

Expand : MatMultSym

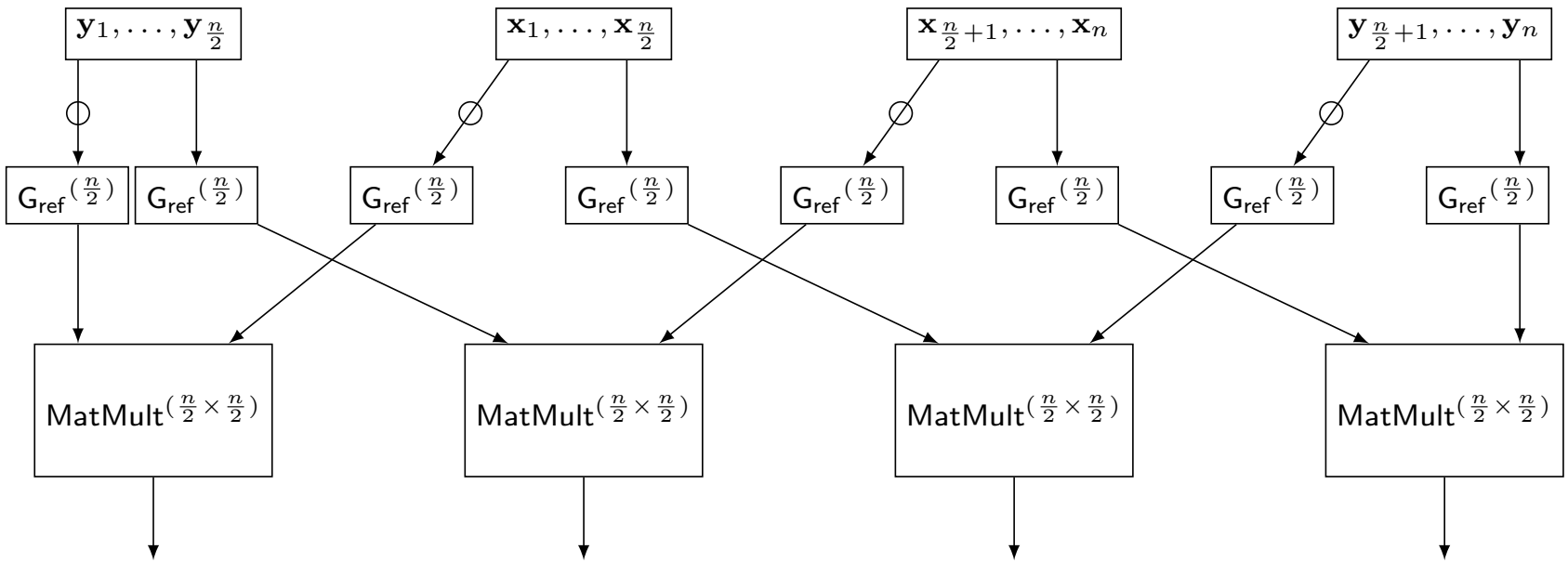


Expand : MatMultSym

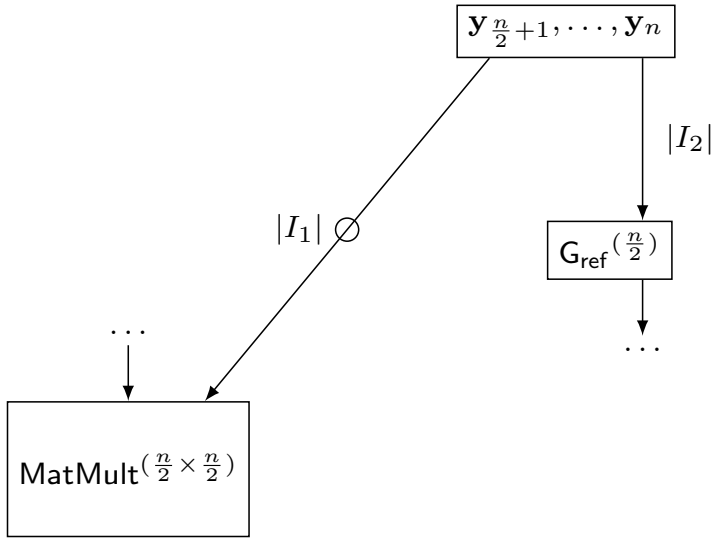


- Same principle for the computations of the envelopes.
- Same drawbacks
- The addition of refresh gadgets upgrades the security.

Expand : UnifMatMult



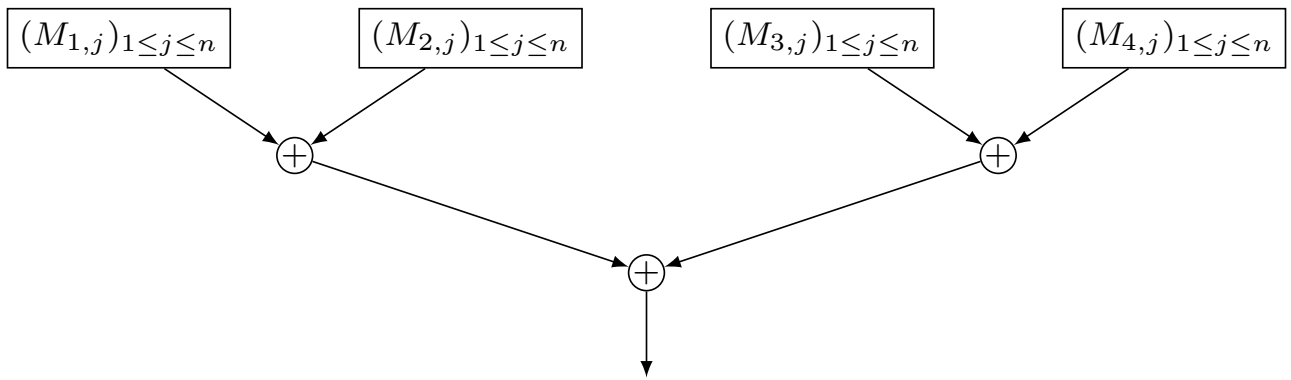
Expand : UnifMatMult



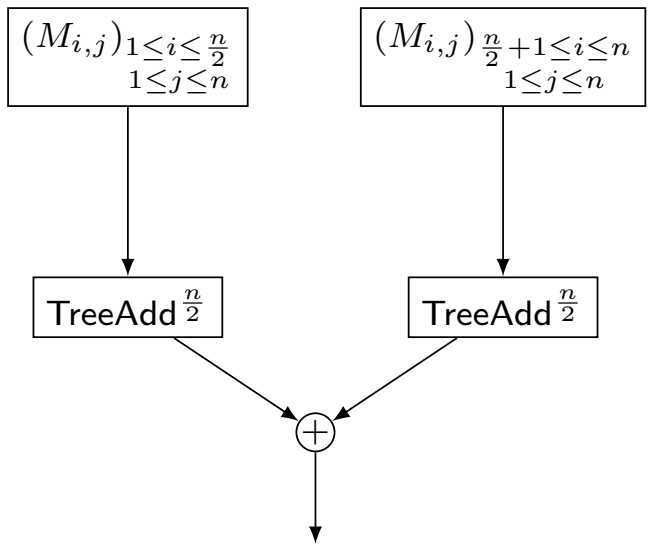
- **Informations** on $|I_1 \cup I_2|$.
- $\mathbb{P}(|I_1 \cup I_2| = \ell, |I_1| = \ell_1, |I_2| = \ell_2) = \frac{\binom{\ell_1}{\ell_1+\ell_2-\ell} \cdot \binom{\frac{n}{2}-\ell_1}{\ell-\ell_1}}{\binom{\frac{n}{2}}{\ell_2}}$
- **Tighter Bounds**

Comp : TreeAdd

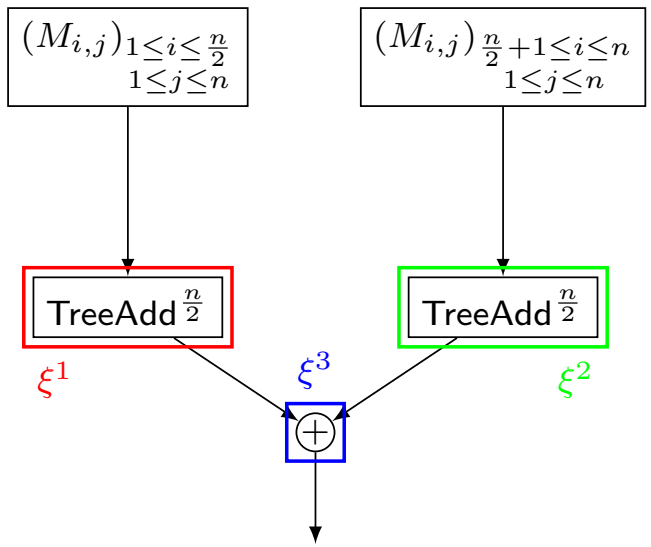
- Transform a n^2 sharing into a n-sharing
- Tree of n-share gadget addition.
- Example with $n = 4$, and a $(M_{i,j})_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}}$ the n^2-sharing :



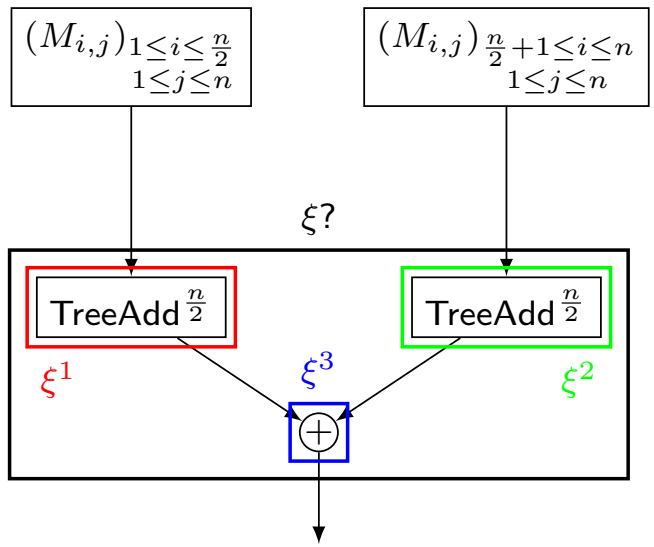
Comp : TreeAdd



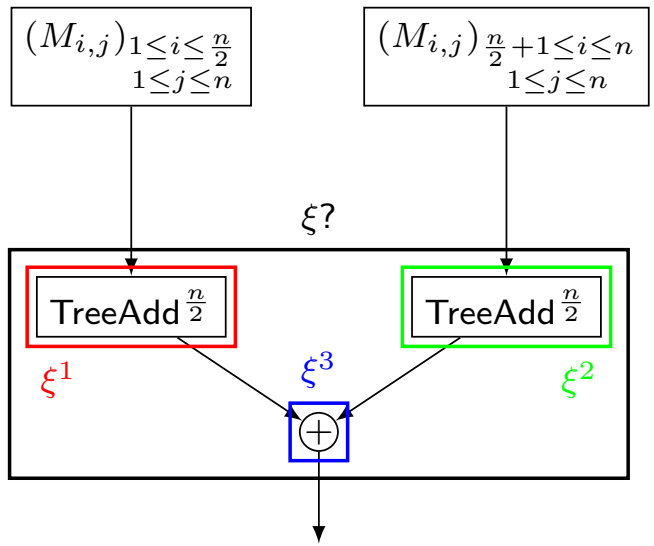
Comp : TreeAdd



Comp : TreeAdd

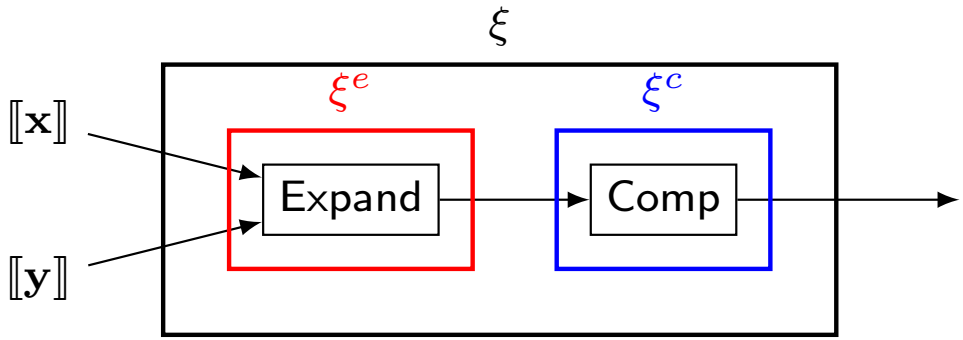


Comp : TreeAdd



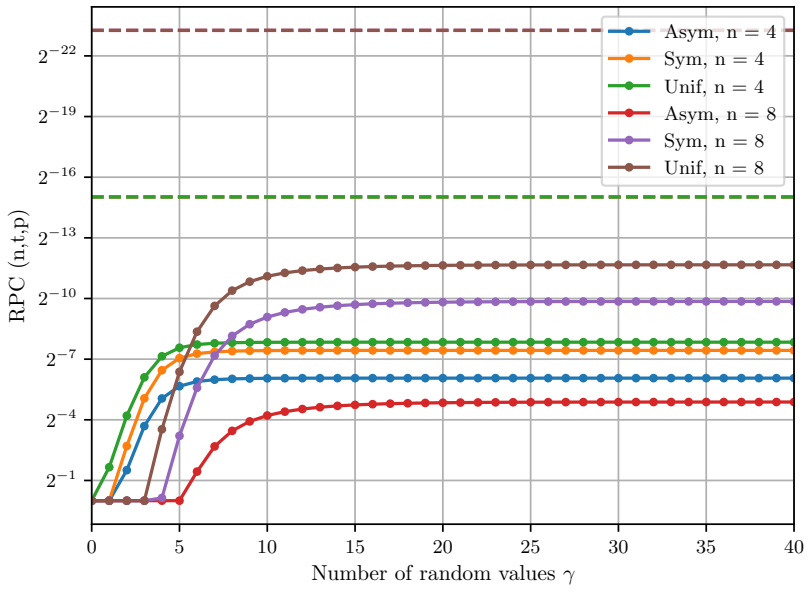
$$\xi_j(i) = \sum_{i_1+i_2=i} \sum_{\ell_1=0}^{n} \sum_{\ell_2=0}^{n} {\xi^1}_{\ell_1}(i_1) \cdot {\xi^2}_{\ell_2}(i_2) \cdot {\xi^3}_j(\ell_1, \ell_2)$$

Multiplication gadget

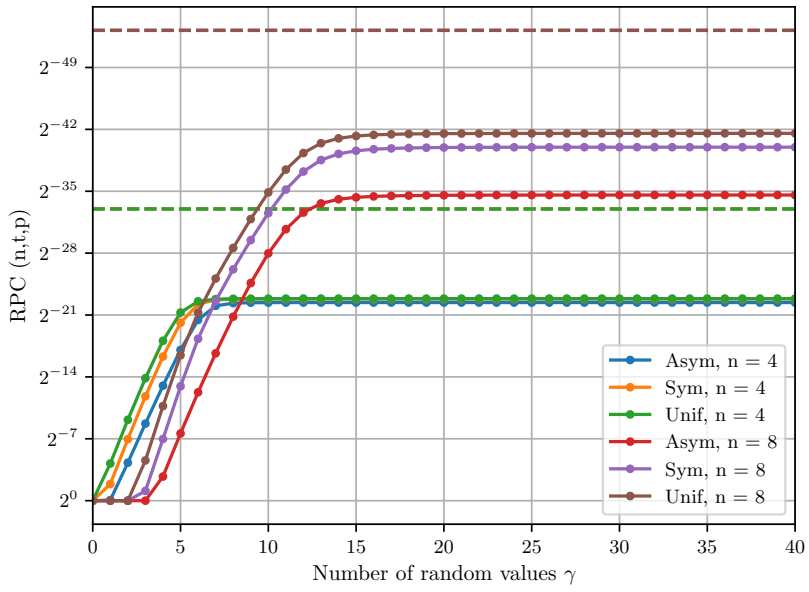


$$\xi_j(i_{\mathbf{x}}, i_{\mathbf{y}}) = \sum_{\ell=0}^{n^2} {\xi^e}_\ell(i_{\mathbf{x}}, i_{\mathbf{y}}) \cdot {\xi^c}_j(\ell)$$

Comparison between different variants



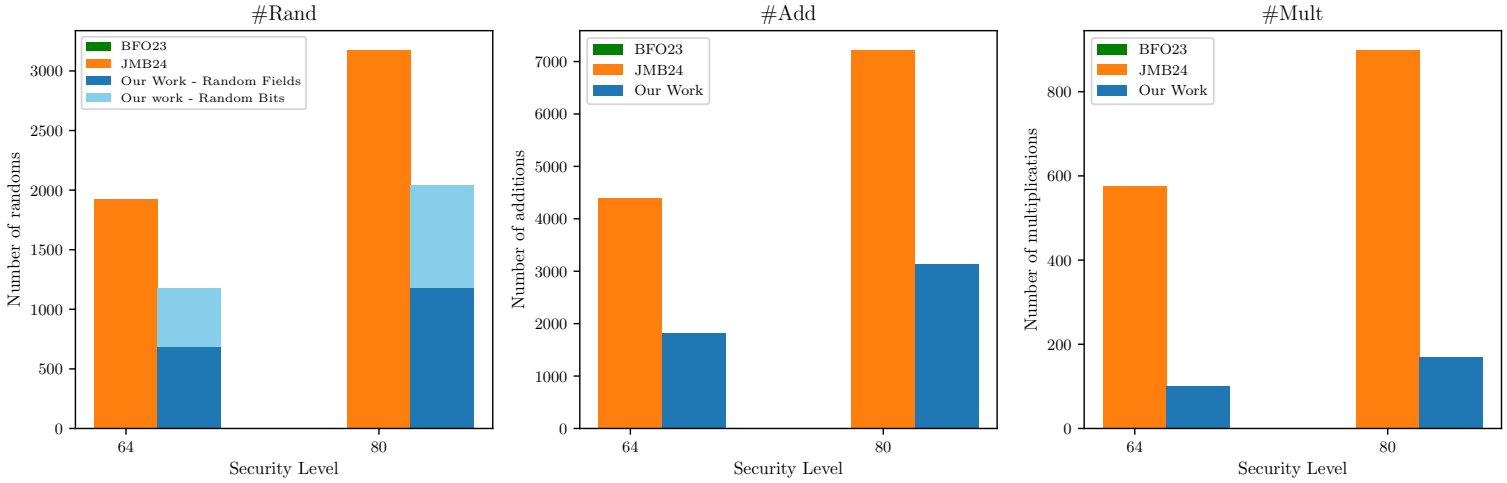
(a) $p = 2^{-6}$



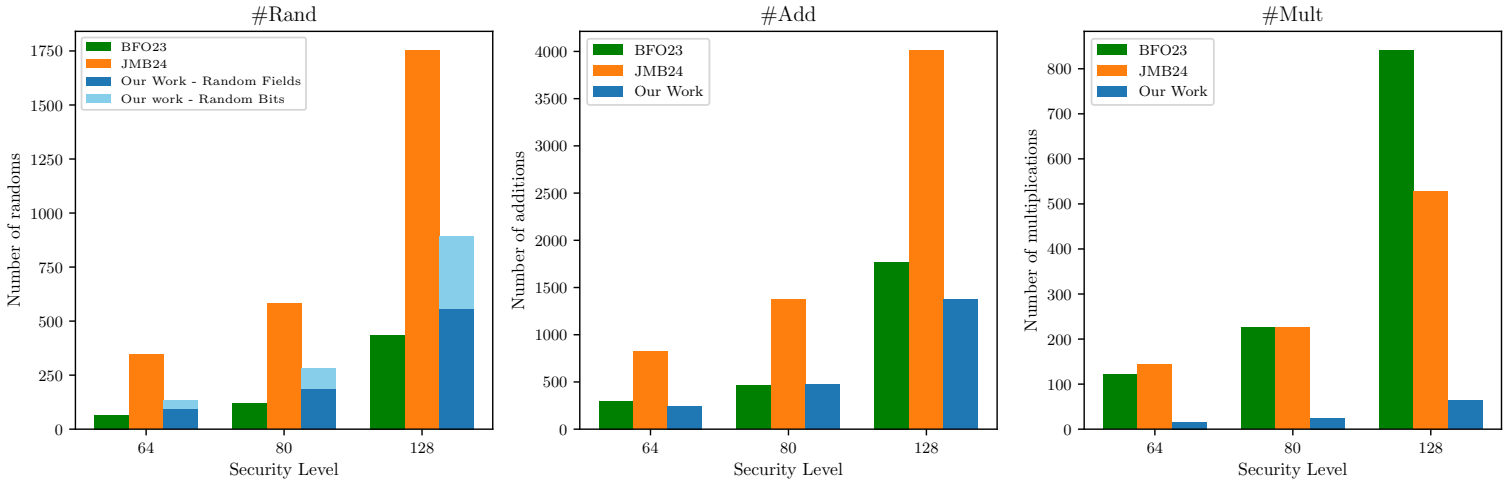
(b) $p = 2^{-12}$

- Asym $\rightarrow$ Multplication gadget with MatMultRef (i.e. refresh half branch).
- Sym $\rightarrow$ Multplication gadget with MatMultSym (i.e. refresh all branch).
- Unif $\rightarrow$ Multplication gadget with UnifMatMult (i.e. with permutation).

Comparison between different works



(a) $p = 2^{-10}$



(b) $p = 2^{-20}$

Second contribution

- Describe a multiplication gadget and computes tight envelopes for cardinal RPC security.
- Compare our multiplication gadget with existing constructions and show that our approach is competitive.

Compiler

- Linear Gadgets : $\mathsf{Add}^\gamma, \mathsf{cMult}^\gamma, \mathsf{cAdd}^\gamma \rightarrow$ Known cardinal RPC envelopes.
- Squaring Gadget : In case $\mathbb{K} = \mathbb{F}_{2^k}$, $\mathsf{Sq}^\gamma \rightarrow$ Known cardinal RPC security envelopes.
- Non Linear Gadget : Multiplication gadget used with UnifMatMult $\rightarrow$ Now known cardinal RPC security envelopes.

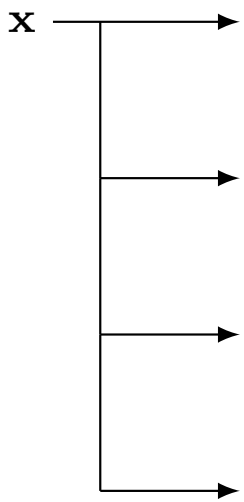
Copy

- Linear Gadgets : $\mathsf{Add}^\gamma, \mathsf{cMult}^\gamma, \mathsf{cAdd}^\gamma \rightarrow$ Known cardinal RPC envelopes.
- Squaring Gadget : In case $\mathbb{K} = \mathbb{F}_{2^k}$, $\mathsf{Sq}^\gamma \rightarrow$ Known cardinal RPC security envelopes.
- Non Linear Gadget : Multiplication gadget used with UnifMatMult $\rightarrow$ Now known cardinal RPC security envelopes.

What to do during duplication of values?
Recursive procedure.

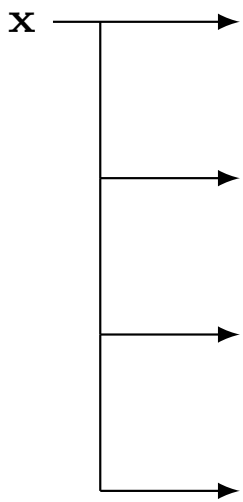
Copy - Example

Fan-out 4 *setting*

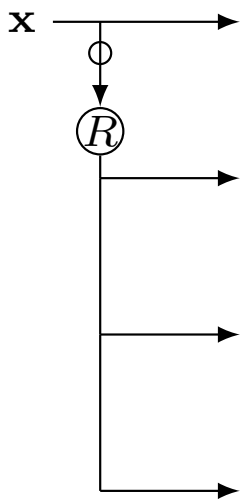


Copy - Example

Fan-out 4 *setting*

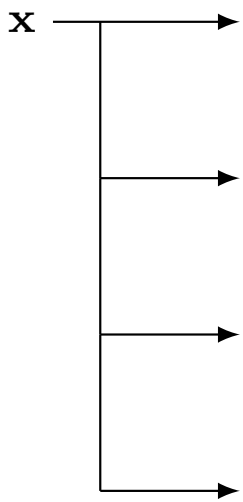


Fan-out 3 *setting*

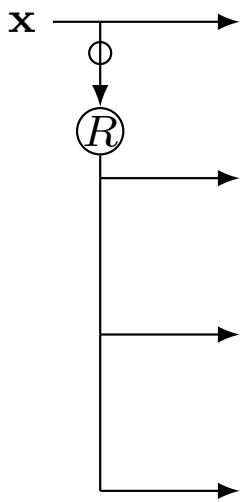


Copy - Example

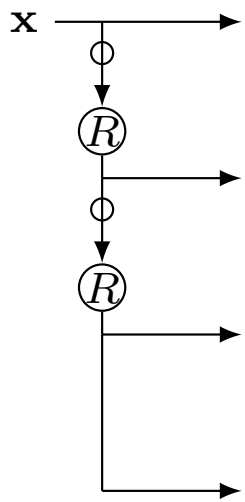
Fan-out 4 *setting*



Fan-out 3 *setting*

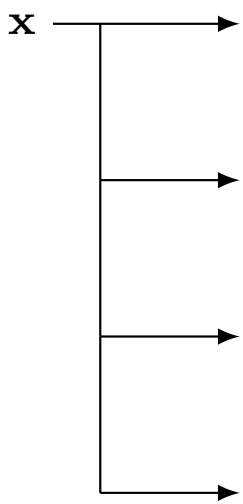


Fan-out 2 *setting*

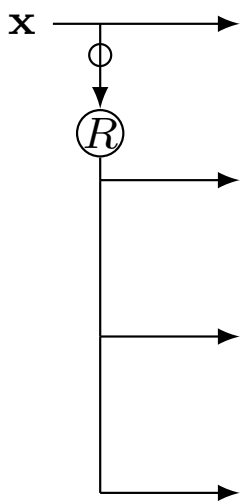


Copy - Example

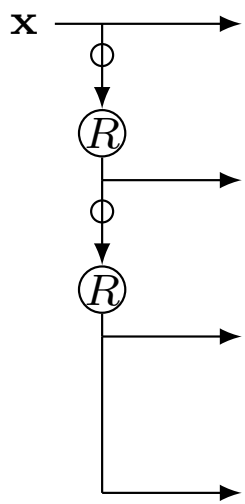
Fan-out 4 *setting*



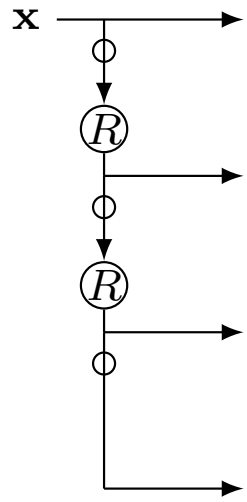
Fan-out 3 *setting*



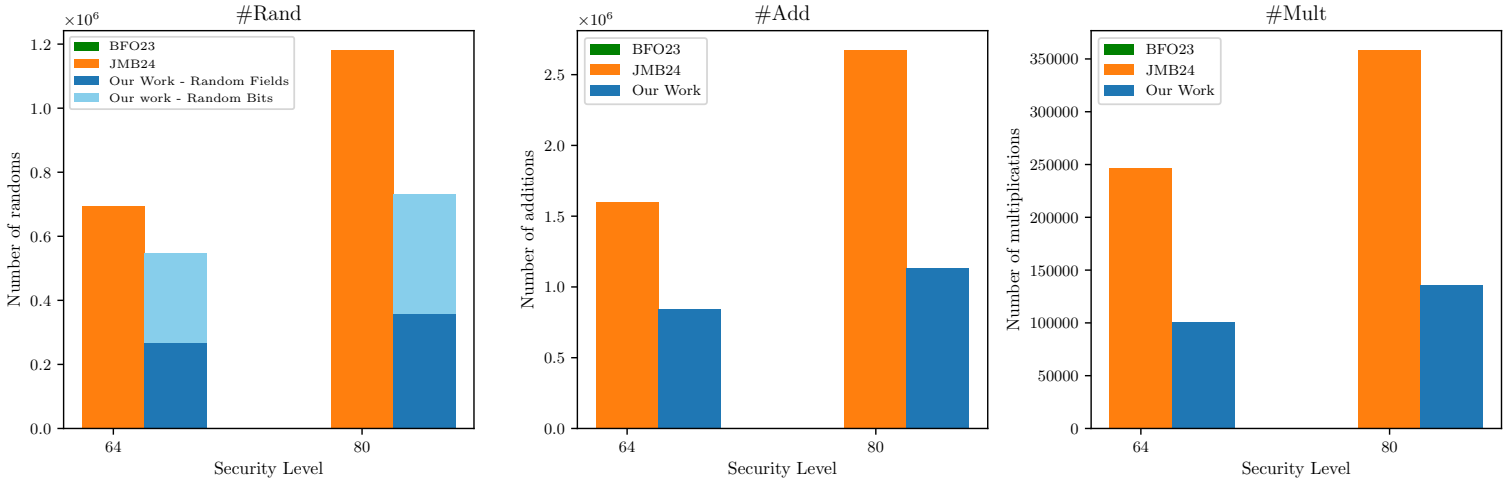
Fan-out 2 *setting*



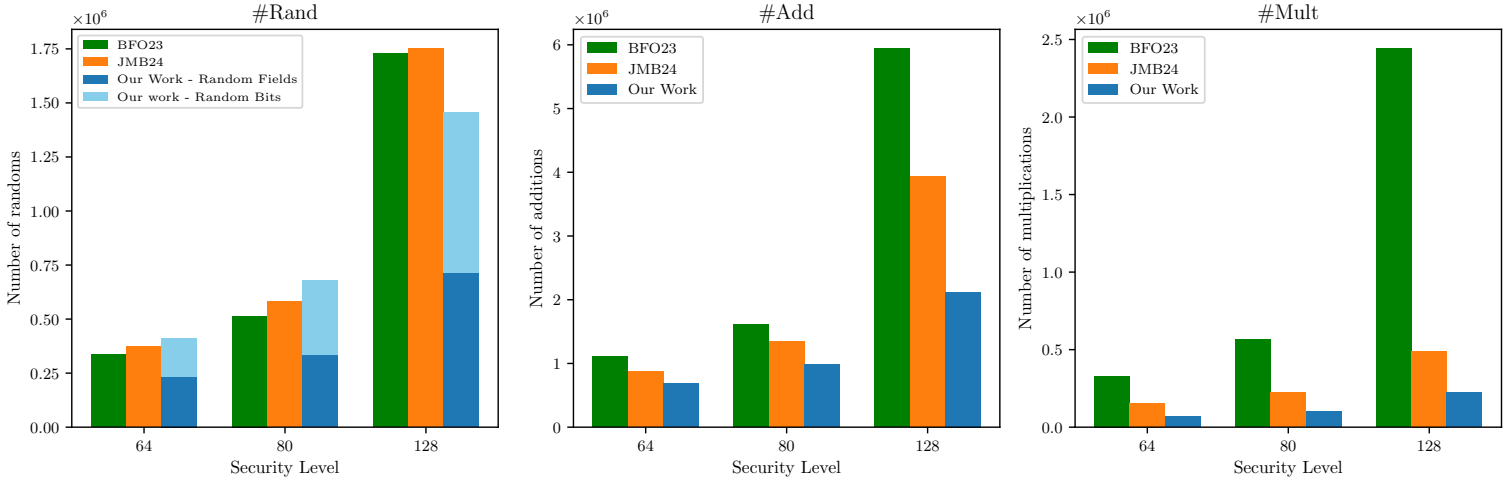
Final Circuit



AES Results



(a) $p = 2^{-16}$



(b) $p = 2^{-20}$

Conclusion

- Describe a procedure for constructing uniformly cardinal RPC envelopes for any gadget.
- Introduce a new multiplication gadget that achieves performance competitive with state-of-the-art multiplication gadgets.
- Present a new complete RP compiler that integrates this gadget and offers better performance than current leading solutions.
- Demonstrate the application of the compiler by using AES as a case study.

Thank you for your attention!