

Decomposition of Large Look-Up Tables for Fast Homomorphic Evaluation

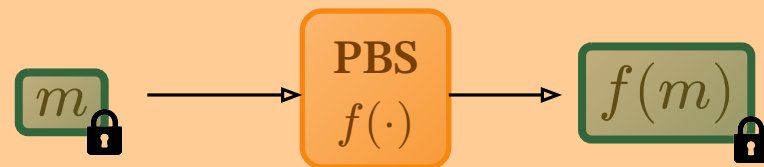
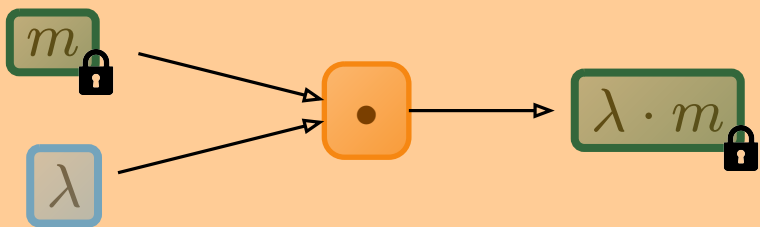
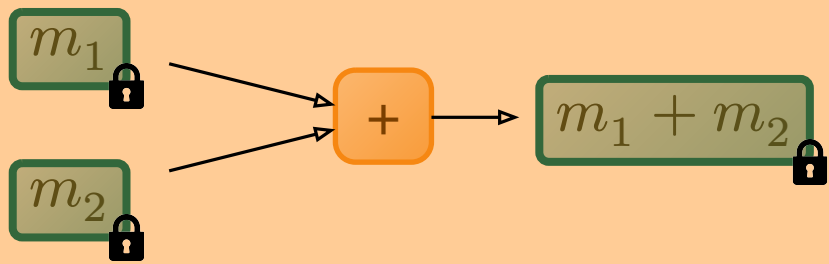
Sonia Belaïd, **Nicolas Bon**, Matthieu Rivain

CryptoExperts

Part 1:

Basics on TFHE scheme

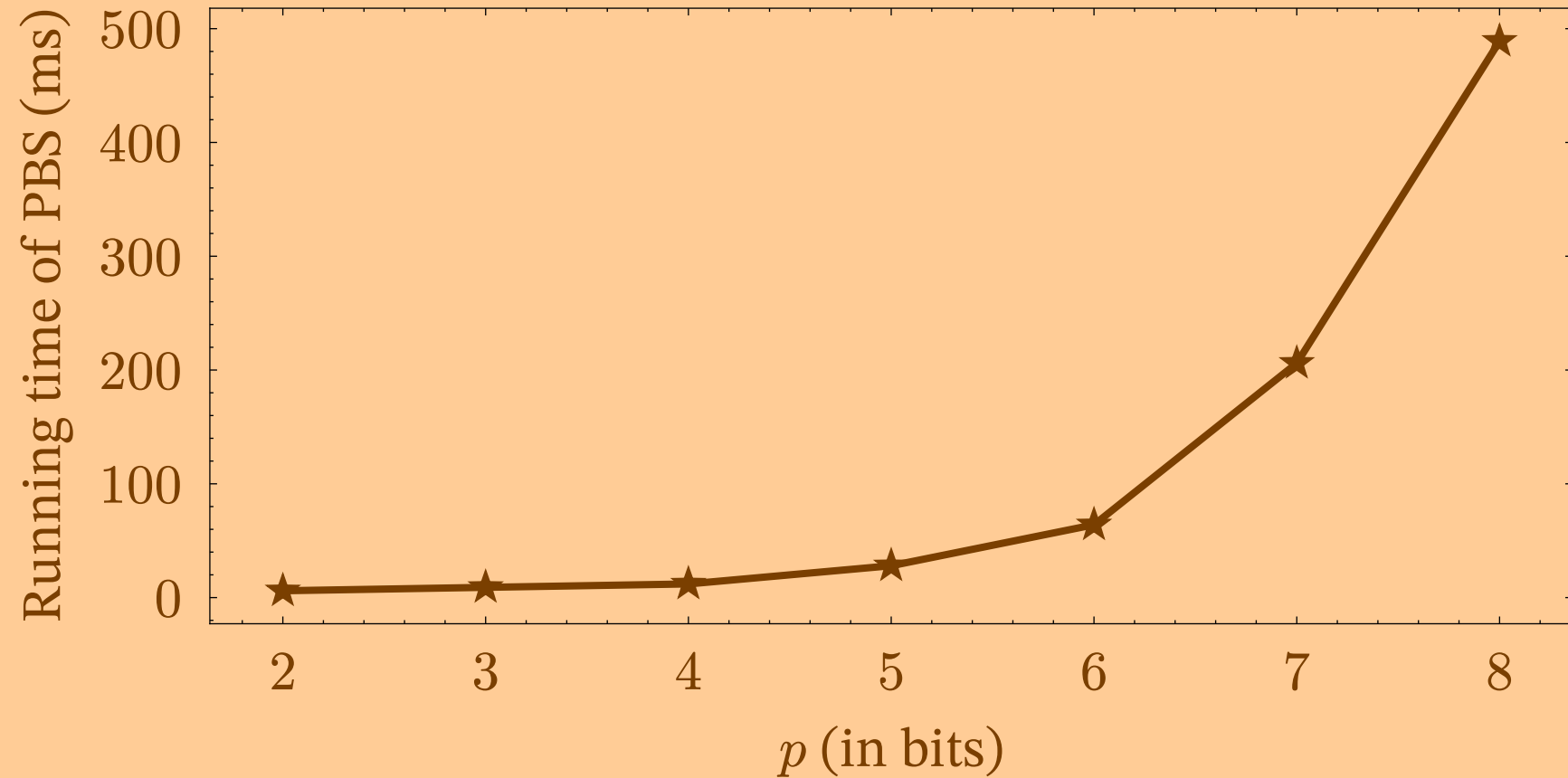
TFHE: the building blocks



Speed 🏎️ , Noise ⬆️

Speed 🐌 , Noise ⬆️

PBS performance degrades with the size of the input



Part 2:

Circuit compilation

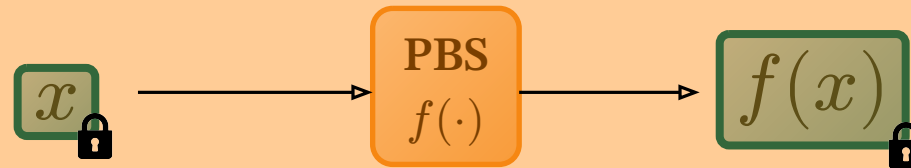
$$f : \mathbb{Z}_p \mapsto \mathbb{Z}_p$$

$$x \rightarrow y$$

Question

Using TFHE basic operations, how do we compute f homomorphically?

Naive version

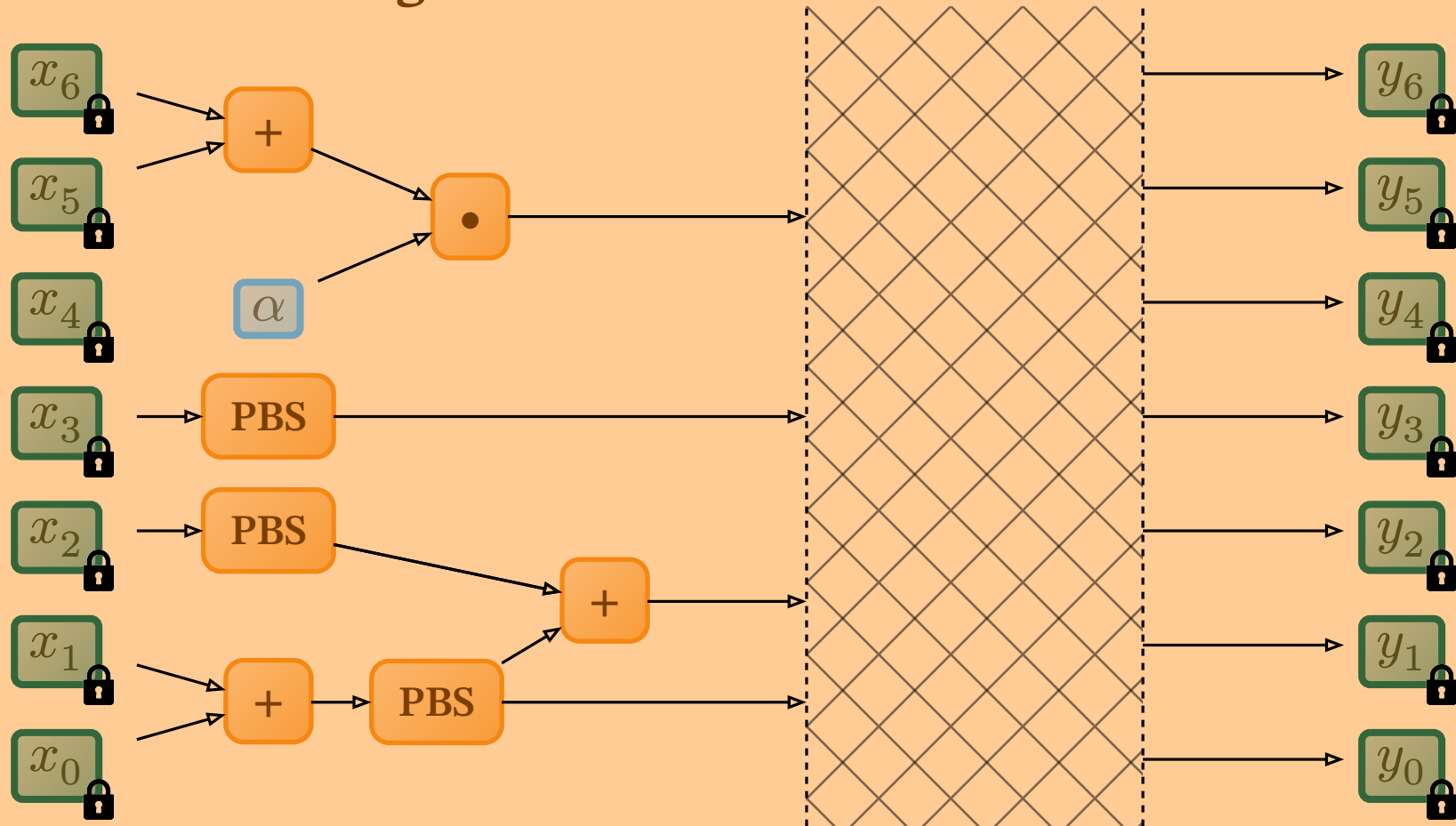


Impractical when the size of the message space p gets too large

A better version using chunks



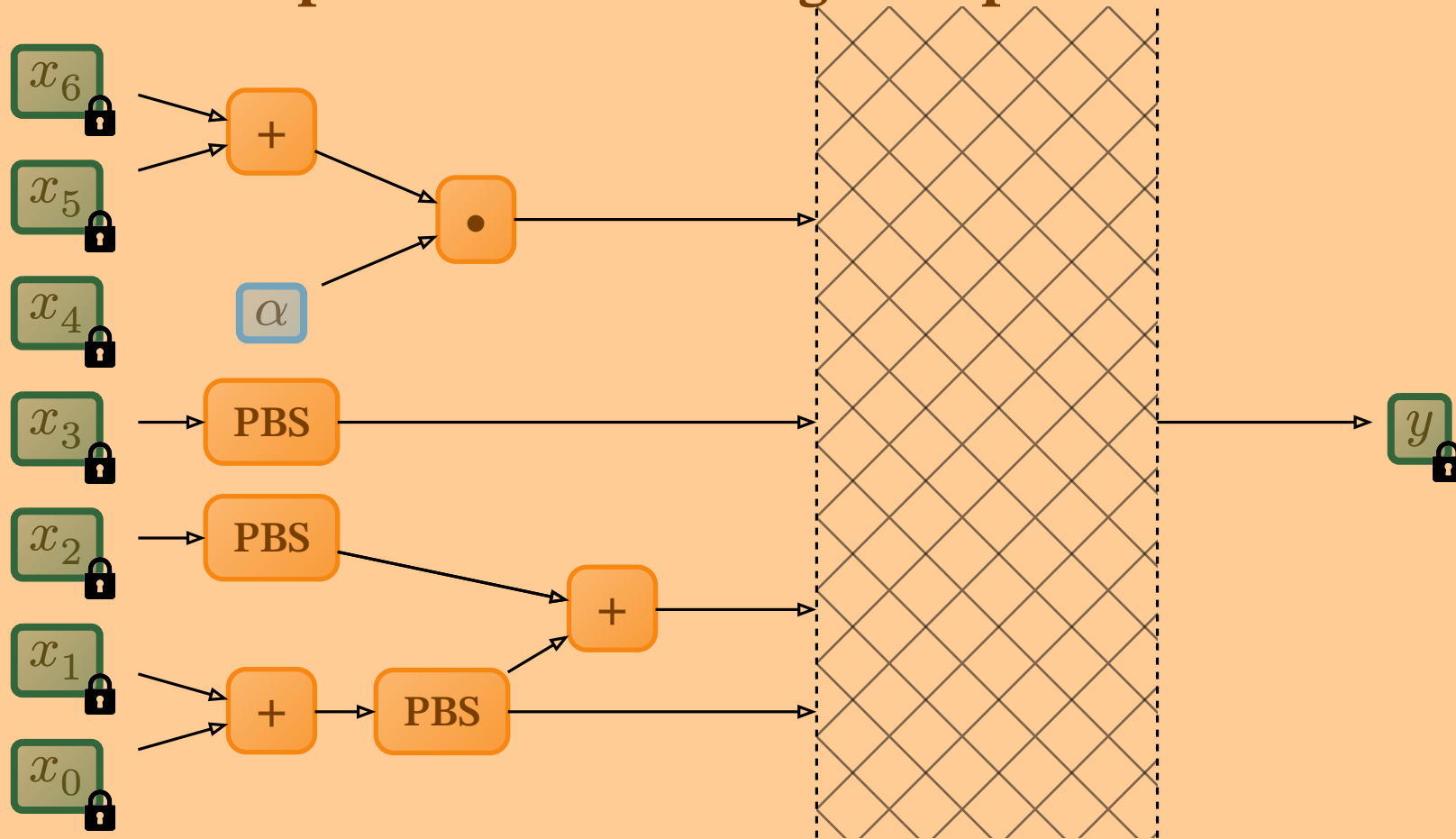
A better version using chunks



Question

Given a function f , how to compile the most efficient circuit possible in the homomorphic domain?

Reduction of the problem to one single output chunk:



This question is already studied in the CHES community, but targeting different metrics.

An usual goal is to minimize the number of **AND** gates.

This question is already studied in the CHES community, but targeting different metrics.

~~An usual goal is to minimize the number of **AND** gates.~~

Instead, we want to minimize the number of !

Part 3:

Our technique

The Dream Scenario

$$\boxed{y} = \sum_{i=1}^n \beta_i \cdot \boxed{x_i}$$

Only linear computation → NO PBS REQUIRED !

The Dream Scenario

$$\boxed{y} = \sum_{i=1}^n \beta_i \cdot \boxed{x_i}$$

Question

In this case, how would we compute $\vec{\beta}$ from the definition of f ?

This was linear algebra all along

We start from the truth table of f

This was linear algebra all along

We start from the truth table of f

x_1	x_2	x_3	x_4	...
0	0	0	0	...
1	0	0	0	...
0	1	0	0	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

y
1
1
0
$\vdots$

This was linear algebra all along

$$\begin{array}{c} \updownarrow s^n \\ \left(\begin{array}{cccc} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ x_1^{(3)} & x_2^{(3)} & \dots & x_n^{(3)} \\ x_1^{(4)} & x_2^{(4)} & \dots & x_n^{(4)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(s^n)} & x_2^{(s^n)} & \dots & x_n^{(s^n)} \end{array} \right) \cdot \left(\begin{array}{c} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{array} \right) = \left(\begin{array}{c} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{s^n} \end{array} \right) \\ \leftarrow n \rightarrow \end{array}$$

This was linear algebra all along

$$\begin{array}{c} \updownarrow s^n \\ \left(\begin{array}{cccc} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ x_1^{(3)} & x_2^{(3)} & \dots & x_n^{(3)} \\ x_1^{(4)} & x_2^{(4)} & \dots & x_n^{(4)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(s^n)} & x_2^{(s^n)} & \dots & x_n^{(s^n)} \end{array} \right) \cdot \boxed{\begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix}} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{s^n} \end{pmatrix} \\ \leftarrow n \rightarrow \end{array}$$

β is pre-computed from f with **Gaussian elimination**.

→ The inverted matrix can be re-used for **any** f of same size

This was linear algebra all along

$$\begin{array}{c} \updownarrow s^n \\ \left(\begin{array}{cccc} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ x_1^{(3)} & x_2^{(3)} & \dots & x_n^{(3)} \\ x_1^{(4)} & x_2^{(4)} & \dots & x_n^{(4)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(s^n)} & x_2^{(s^n)} & \dots & x_n^{(s^n)} \end{array} \right) \cdot \boxed{\begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix}} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{s^n} \end{pmatrix} \\ \leftarrow n \rightarrow \end{array}$$



Problem: in practice $n \ll s^n$. So the rank of the matrix is not sufficient to find a solution.

This was linear algebra all along

$$\begin{array}{c} \updownarrow s^n \end{array} \left(\begin{array}{cccc|ccc} x_1^{(1)} & x_2^{(1)} & \dots & x_{s^n}^{(1)} & x_{n+1}^{(1)} & \dots & x_{s^n}^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_{s^n}^{(2)} & x_{n+1}^{(2)} & \dots & x_{s^n}^{(2)} \\ x_1^{(3)} & x_2^{(3)} & \dots & x_{s^n}^{(3)} & x_{n+1}^{(3)} & \dots & x_{s^n}^{(3)} \\ x_1^{(4)} & x_2^{(4)} & \dots & x_{s^n}^{(4)} & x_{n+1}^{(4)} & \dots & x_{s^n}^{(4)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_1^{(s^n)} & x_2^{(s^n)} & \dots & x_{s^n}^{(s^n)} & x_{n+1}^{(s^n)} & \dots & x_{s^n}^{(s^n)} \end{array} \right) \cdot \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{s^n} \end{pmatrix}$$

$\leftarrow s^n \rightarrow$

The solution



We extend the matrix with synthetic variables!

Question

How to generate the synthetic variables?

Question

How to generate the synthetic variables?

First idea: we sample random functions $\varphi_{n+1}, \dots, \varphi_{s^n}$.

$$x_{n+1} = \varphi_{n+1}(x_1, \dots, x_n)$$

PBS

Question

How to generate the synthetic variables?

First idea: we sample random functions $\varphi_{n+1}, \dots, \varphi_{s^n}$.

$$x_{n+1} = \varphi_{n+1}(x_1, \dots, x_n)$$

PBS

$$x_{n+2} = \varphi_{n+2}(x_1, \dots, x_{n+1})$$

PBS

Question

How to generate the synthetic variables?

First idea: we sample random functions $\varphi_{n+1}, \dots, \varphi_{s^n}$.

$$x_{n+1} = \varphi_{n+1}(x_1, \dots, x_n)$$

PBS

$$x_{n+2} = \varphi_{n+2}(x_1, \dots, x_{n+1})$$

PBS

$\vdots$

$\vdots$

$$x_{s^n} = \varphi_{s^n}(x_1, \dots, x_{s^n-1})$$

PBS

Question

How to generate the synthetic variables?

First idea: we sample random functions $\varphi_{n+1}, \dots, \varphi_{s^n}$.

$$x_{n+1} = \varphi_{n+1}(x_1, \dots, x_n) \quad \text{PBS}$$

$$x_{n+2} = \varphi_{n+2}(x_1, \dots, x_{n+1}) \quad \text{PBS}$$

$$\vdots$$
$$\vdots$$

$$x_{s^n} = \varphi_{s^n}(x_1, \dots, x_{s^n-1}) \quad \text{PBS}$$



This costs one **PBS** per function. Not efficient in practice.

Second idea: we use multiplication to use fewer PBS:

Second idea: we use multiplication to use fewer **PBS**:

A ciphertext-ciphertext product takes two PBS:

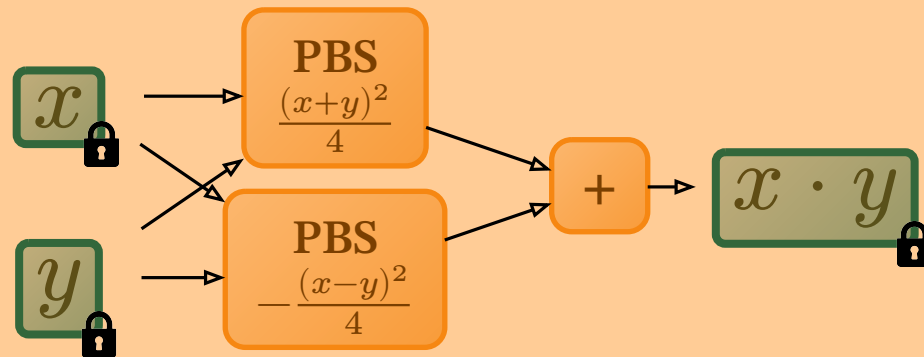


Second idea: we use multiplication to use fewer **PBS**:

A ciphertext-ciphertext product takes two PBS:



$$x \cdot y = \frac{(x+y)^2}{4} - \frac{(x-y)^2}{4}$$



Second idea: we use multiplication to use fewer **PBS**:

A ciphertext-ciphertext product takes two PBS:



Construction of basis:

$$x_{n+1} = \varphi_{n+1}(x_1, \dots, x_n) \quad \text{PBS}$$

$$x_{n+2} = \varphi_{n+2}(x_1, \dots, x_{n+1}) \quad \text{PBS}$$

$\vdots$

$$x_{n+\lambda} = \varphi_{n+\lambda}(x_1, \dots, x_{n+\lambda-1}) \quad \text{PBS}$$

Smaller by a factor $\sqrt{}$

Second idea: we use multiplication to use fewer **PBS**:

A ciphertext-ciphertext product takes two PBS:



Construction of basis:

$$x_{n+1} = \varphi_{n+1}(x_1, \dots, x_n) \quad \text{PBS}$$

$$x_{n+2} = \varphi_{n+2}(x_1, \dots, x_{n+1}) \quad \text{PBS}$$

$\vdots$

$$x_{n+\lambda} = \varphi_{n+\lambda}(x_1, \dots, x_{n+\lambda-1}) \quad \text{PBS}$$

Smaller by a factor $\sqrt{}$

Products:

$$\left(\sum_{j=1}^{n+\lambda} \beta_{i,j} \cdot x_j \right) \cdot \left(\sum_{k=1}^{n+\lambda} d_{i,k} \cdot x_k \right)$$

The equation shows the product of two sums. Each term in the sums is represented by a green box with a dot and a lock icon, indicating ciphertexts.

produces $n + \lambda$ degrees of freedom in 2^{PBS} .

Full formula:

$$f(\mathbf{x}) = \sum_{i=1}^t \left(\sum_{j=1}^{n+\lambda} \beta_{i,j} \cdot x_j \right) \cdot \left(\sum_{k=1}^{n+\lambda} d_{i,k} \cdot x_k \right) + \sum_{k=1}^{n+\lambda} \beta_{t,j} \cdot x_j$$

Cost of evaluation of the decomposition

$$\lambda + 2t \text{ PBS}$$

Bounds on the parameters (necessary to get a full-rank matrix)

$$s^n \leq (t + 1)(n + \lambda)$$

Ring vs Field



We need to work in a **field** for Gauss elimination to work!

Ring vs Field



We need to work in a **field** for Gauss elimination to work!

Embedding

We embed $\mathbb{Z}_s$ into $\mathbb{F}_p$ (with $s \leq p$)

Ring vs Field



We need to work in a **field** for Gauss elimination to work!

Embedding

We embed $\mathbb{Z}_s$ into $\mathbb{F}_p$ (with $s \leq p$)

This enable further optimizations, because the space of solutions for β increases.

Generalization to several outputs

Naive method

Each output can be processed independently

Generalization to several outputs

Naive method

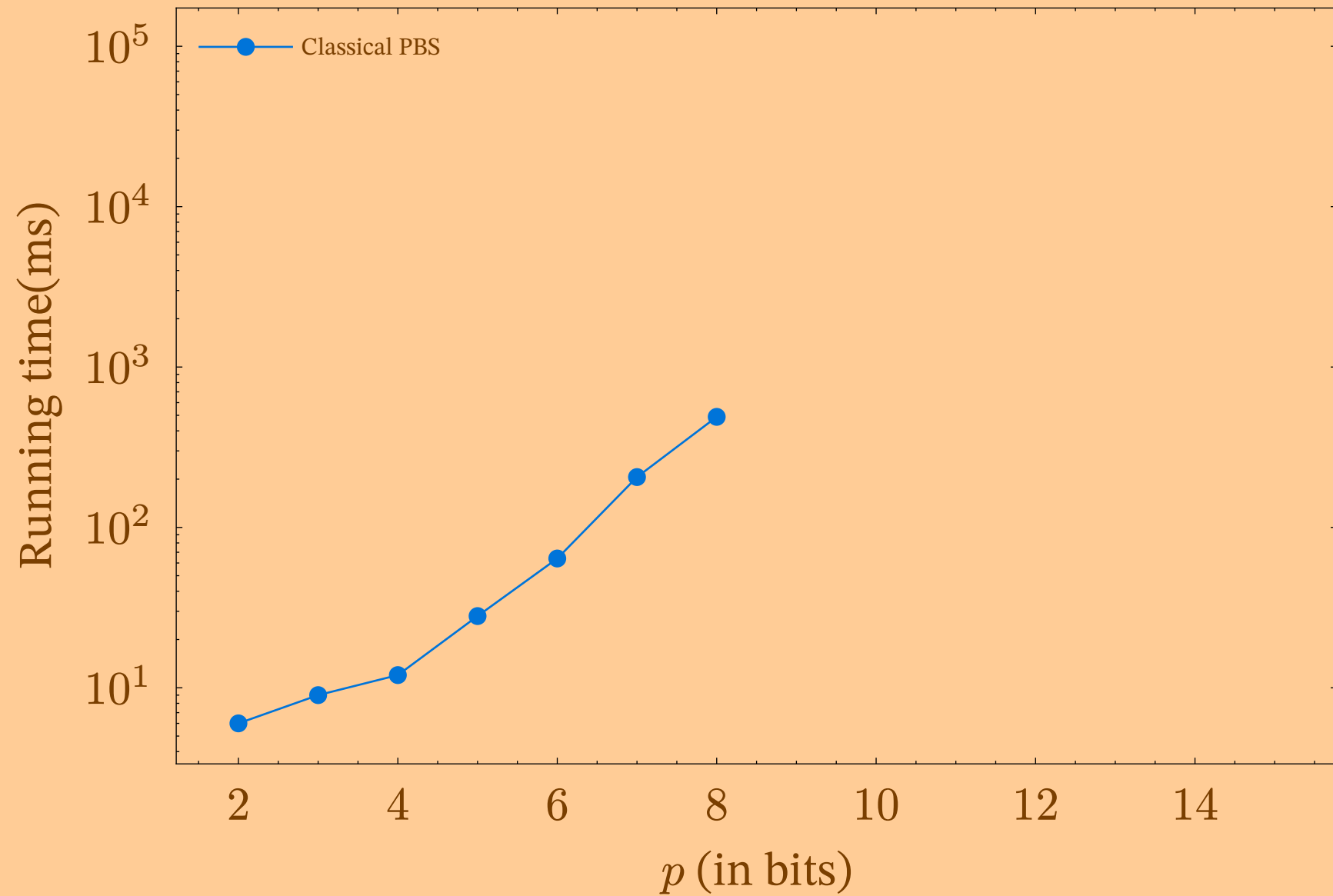
Each output can be processed independently

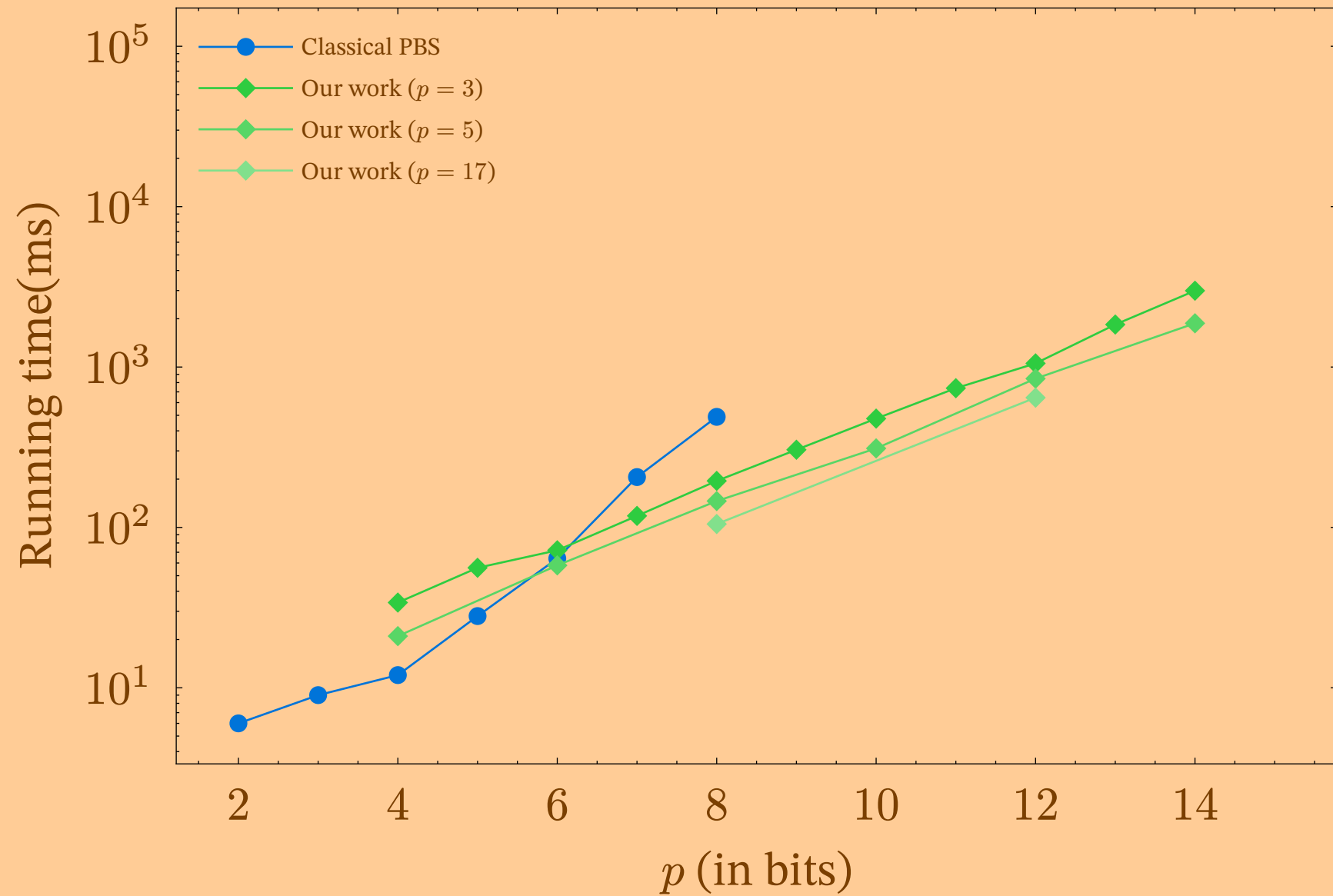
A better idea

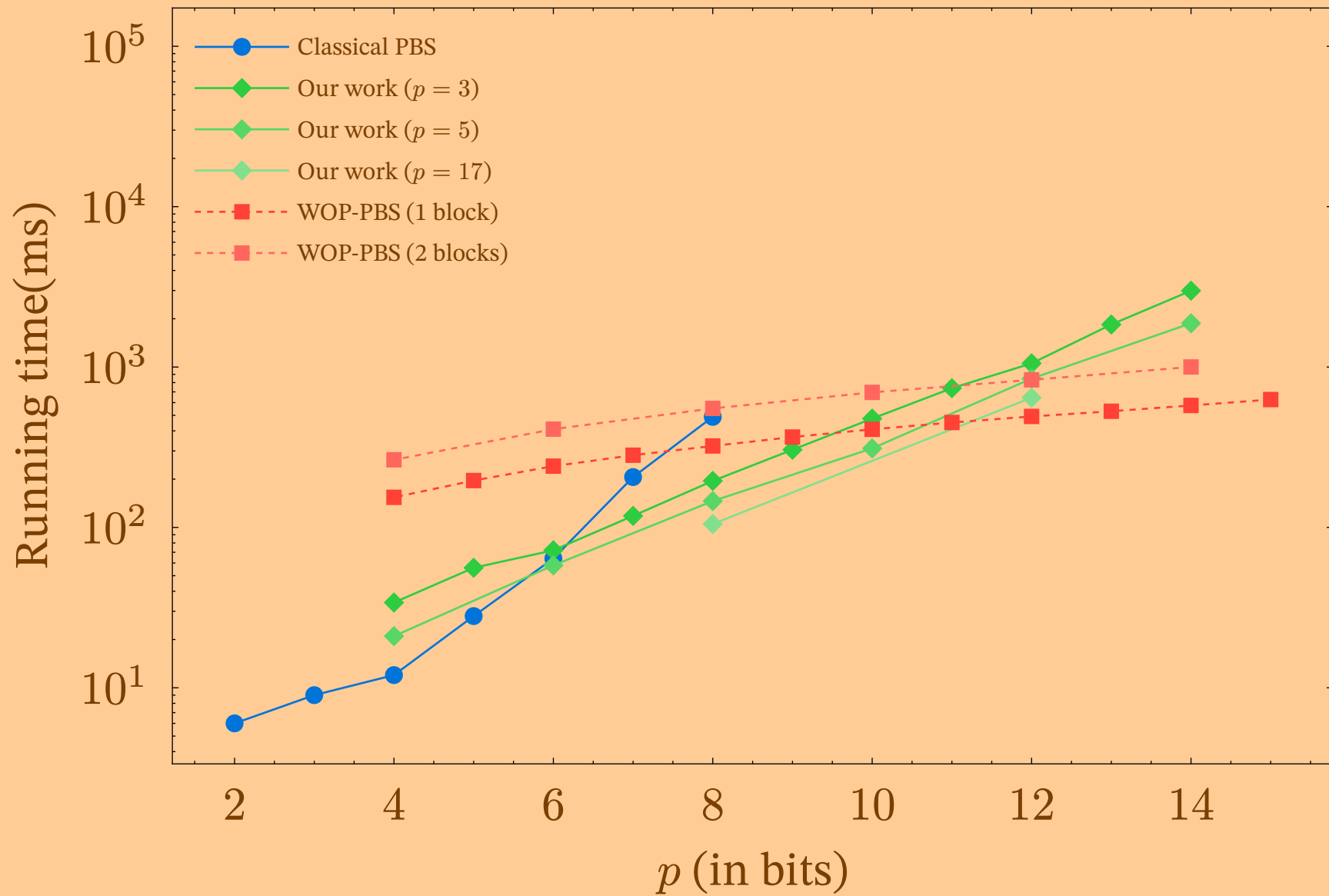
Output $n + 1$ can re-use the basis of Output n , thus reducing the number of **PBS** at each step.

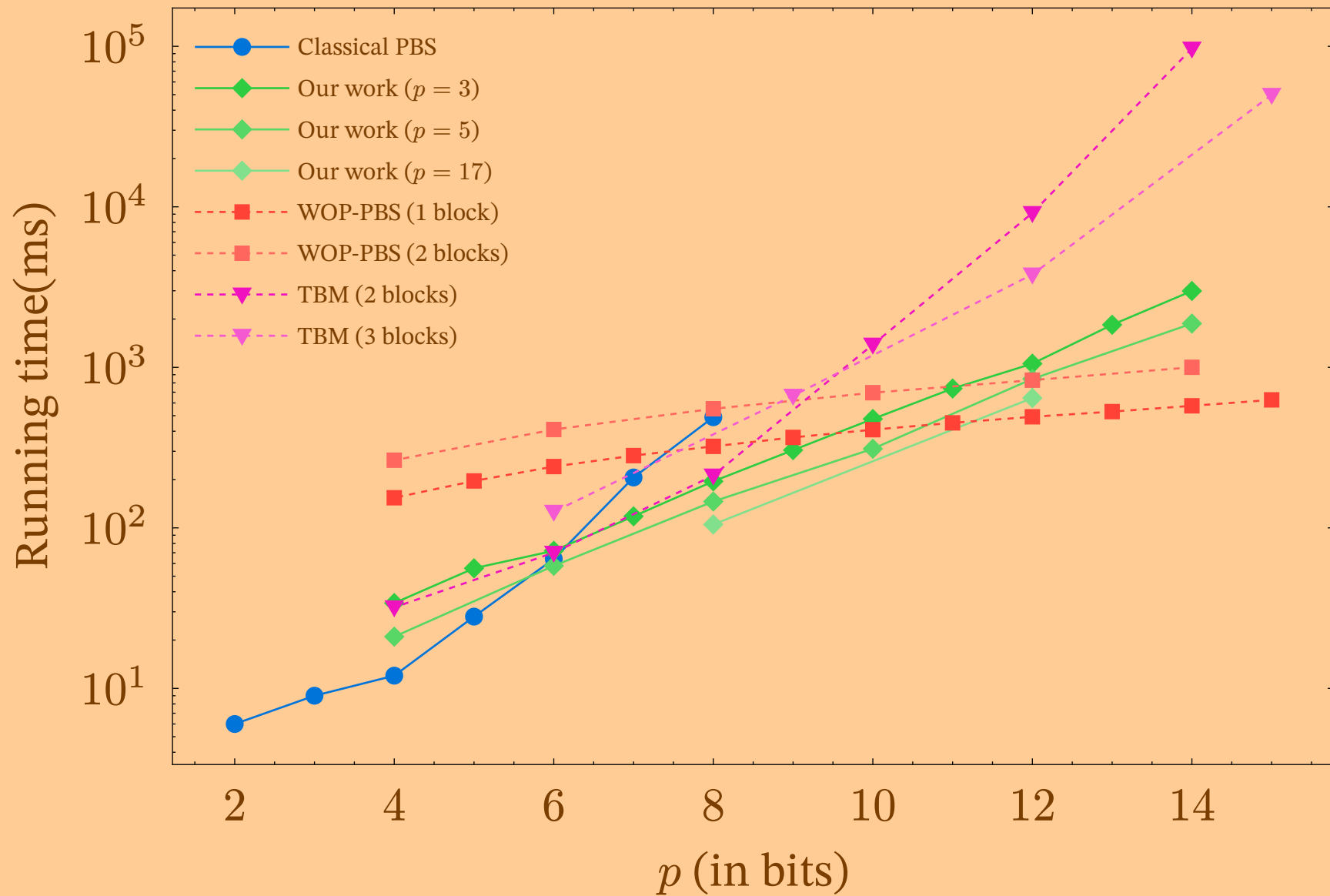
Part 4:

Experimental Results









Conclusion

Conclusion

- Improvement of TFHE homomorphic function evaluation at medium to large precision
- Conceptually simple, only basic building blocks used
- The heavy pre-computation can be re-used on any LUT of same size.

Conclusion

- Improvement of TFHE homomorphic function evaluation at medium to large precision
- Conceptually simple, only basic building blocks used
- The heavy pre-computation can be re-used on any LUT of same size.

Thank you!