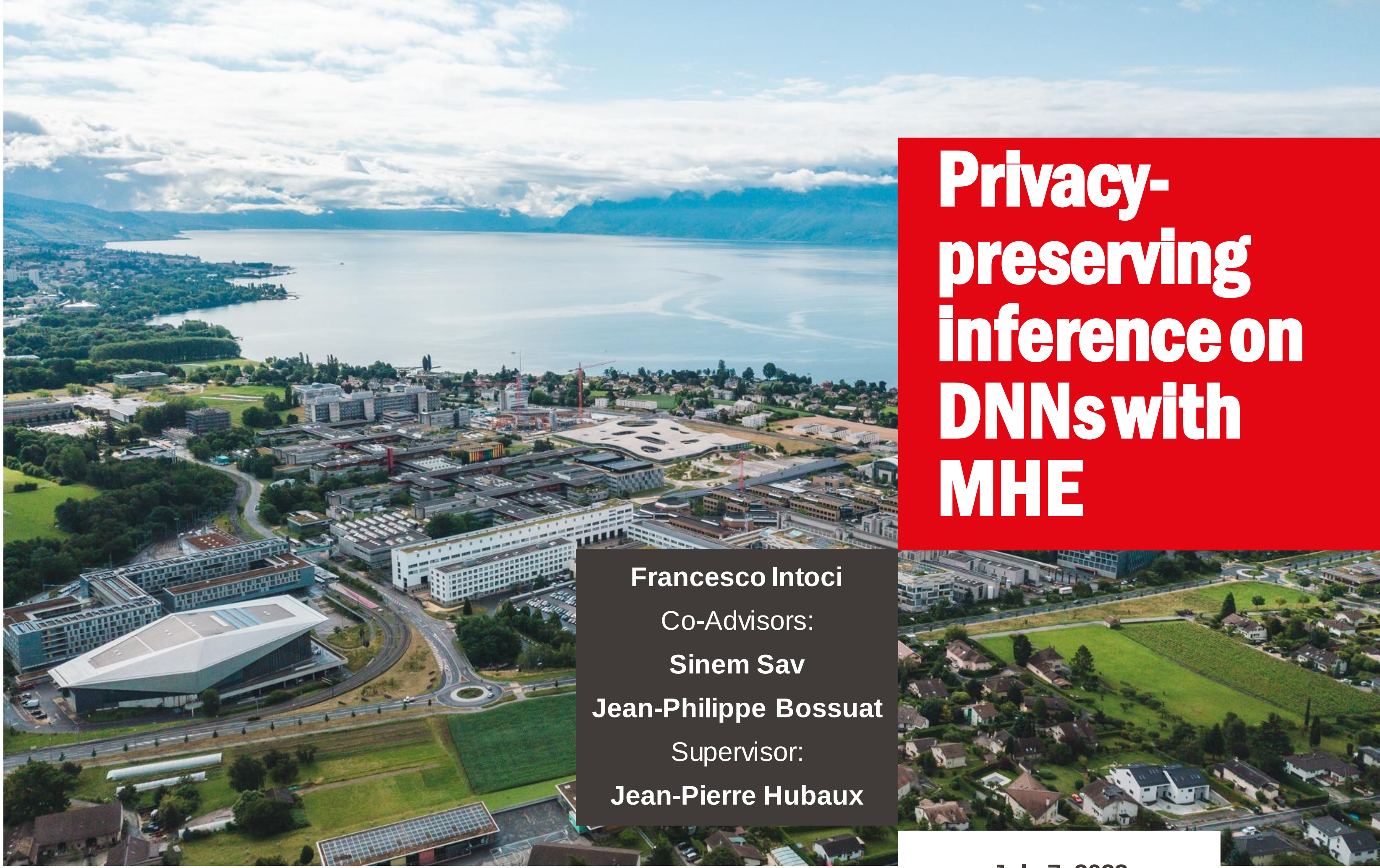


Privacy-preserving inference on DNNs with MHE

Francesco Intoci

Co-Advisors:

Sinem Sav

Jean-Philippe Bossuat

Supervisor:

Jean-Pierre Hubaux

LABORATORY FOR DATA SECURITY

École polytechnique fédérale de Lausanne

July 7, 2022

Introduction

LABORATORY FOR DATA SECURITY

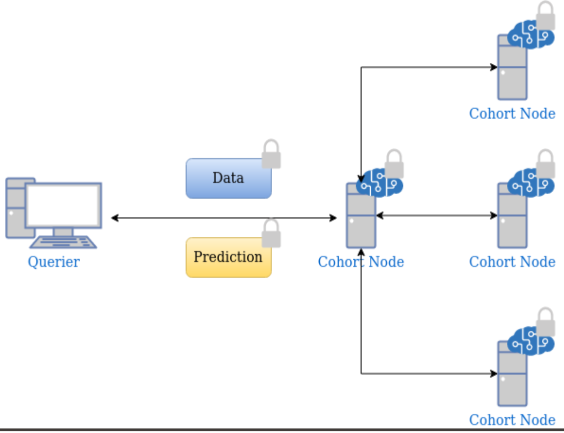
- Enable efficient privacy-preserving inference on Deep Neural Networks
- Different scenarios

Model in plaintext – data encrypted

Model owner offers PaaS

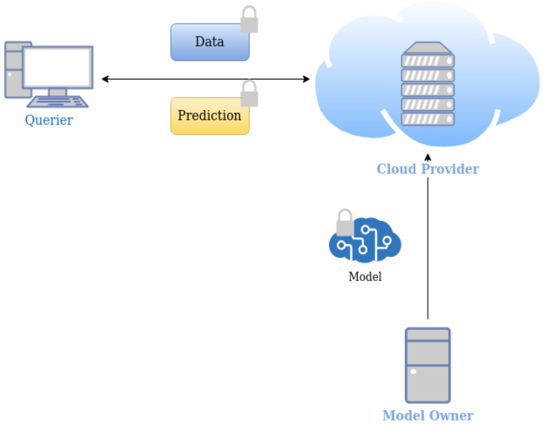
Model encrypted – data encrypted

Model is trained under encryption by a cohort of nodes. Several model holders [1]



Model encrypted – data encrypted

Model owner offloads computation to untrusted cloud provider



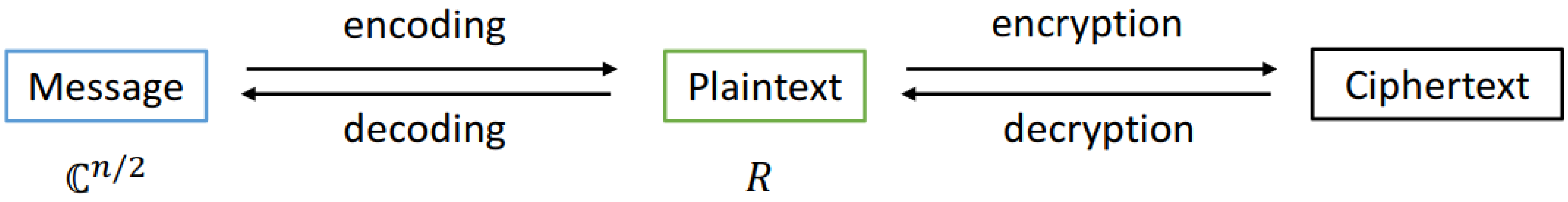
[1] POSEIDON: Privacy-Preserving Federated Neural Network Learning

Challenges

1. How to design an efficient and consistent **data packing** for network evaluation?
2. How to efficiently compute **convolutions on encrypted data** in a layer agnostic way?
3. How to **approximate non linear operations** in the network, and how does this impact the training phase and accuracy of models?
4. How to compute homomorphic **matrix multiplication** in an efficient way?
5. How to handle **data size which does not fit** into one ciphertext?

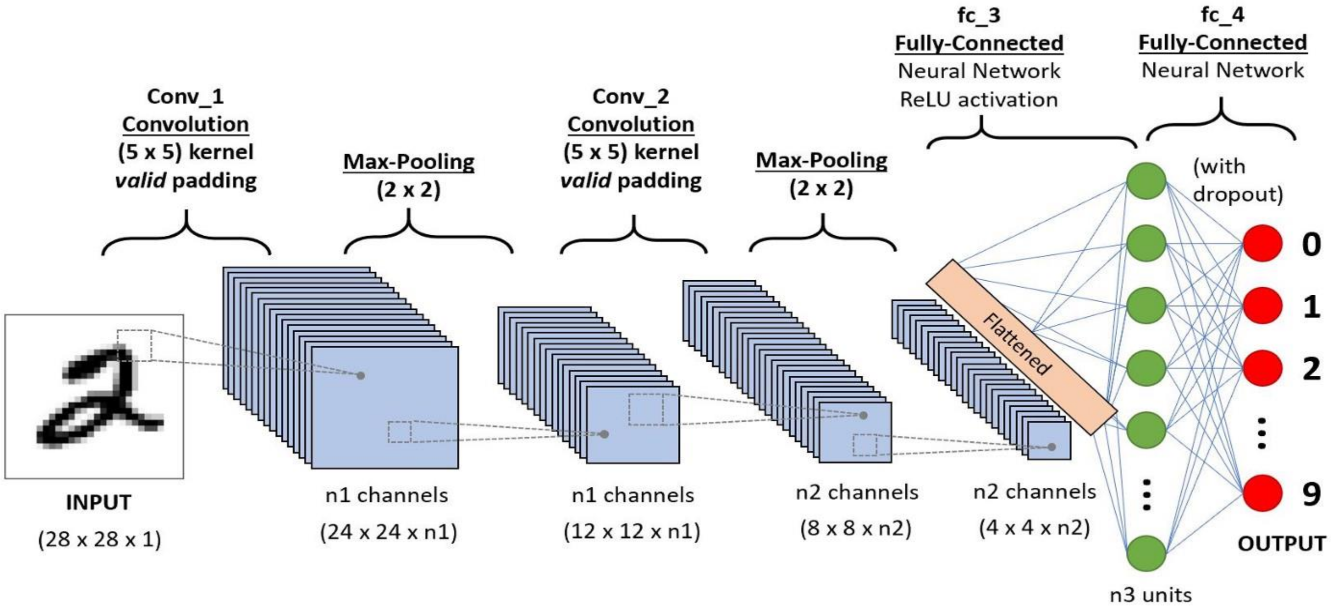
Background – CKKS scheme

- Based on RLWE hardness
- Leveled encryption schemes
- Suited for computation over floating point numbers



From: https://yongsoosong.github.io

Background – Convolutional Neural Network



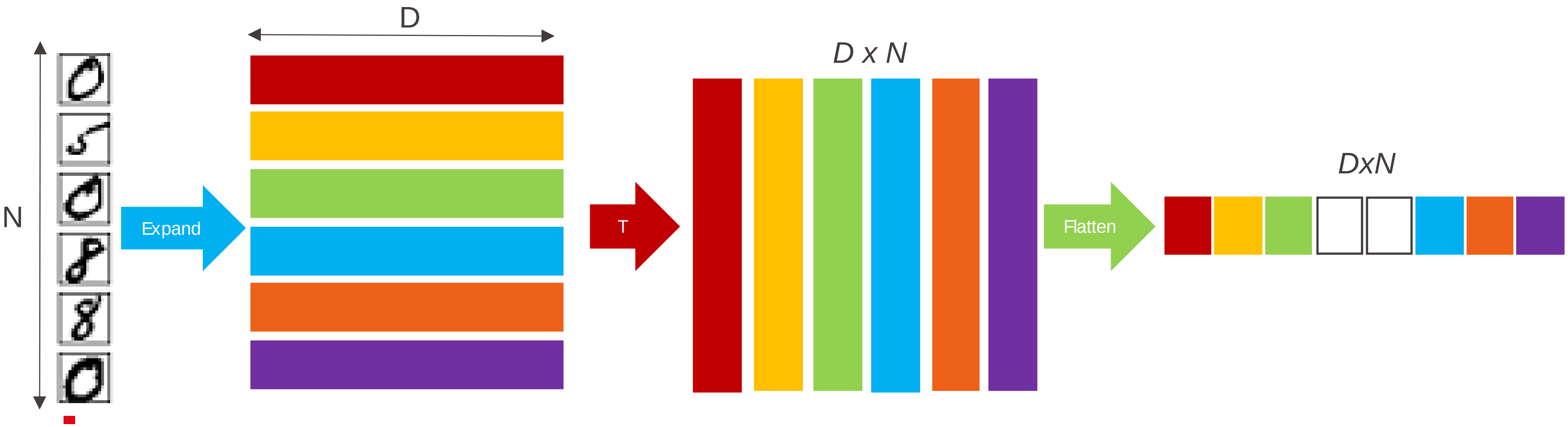
From: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Challenges

1. How to design an efficient and consistent data packing for network evaluation?
2. How to efficiently compute convolutions on encrypted data in a layer agnostic way?
3. How to approximate non linear operations in the network, and how does this impact the training phase and accuracy of models?
4. How to compute homomorphic matrix multiplication in an efficient way?
5. How to handle data size which does not fit into one ciphertext?

Methods - Data packing

- Querier holds batch of *N* images:
  - Pre-processing tasks (normalization,padding, etc...)
  - Images are represented as vectors of size *D* (feature number)
  - The *NxD* matrix is transposed
  - The *NxD* matrix is row-flattened
- Benefits:
  - Very simple
  - Fully compatible with matrix multiplication approach



Challenges

1. How to design an efficient and consistent data packing for network evaluation?
2. How to efficiently compute convolutions on encrypted data in a layer agnostic way?
3. How to approximate non linear operations in the network, and how does this impact the training phase and accuracy of models?
4. How to compute homomorphic matrix multiplication in an efficient way?
5. How to handle data size which does not fit into one ciphertext?

Methods - Convolutional layer «linearization»

- Transform the convolutional layer into a sparse layer and use matrix multiplication (Toeplitz matrix representation)

- Example: transformation of a filter operating on a 3x3 matrix with stride 1

$$\begin{bmatrix} k_1 & k_2 \\ k_3 & k_4 \end{bmatrix} = \begin{bmatrix} k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 & 0 \\ 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 \\ 0 & 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 \end{bmatrix}$$

Methods - Convolutional layer «linearization» cont'd

- Generalized for *f* kernels of KxK size, with *M* input channels

$$\begin{bmatrix} \langle m(k_{1,ch1}) | \dots | m(k_{1,chM}) \rangle \\ \langle m(k_{2,ch1}) | \dots | m(k_{2,chM}) \rangle \\ \langle m(k_{f,ch1}) | \dots | m(k_{f,chM}) \rangle \end{bmatrix}$$

- Benefits:
  - Can represent any convolution
  - Self-consistent for subsequent convolutional or linear layers

Challenges

1. How to design an efficient and consistent data packing for network evaluation?
2. How to efficiently compute convolutions on encrypted data in a layer agnostic way?
3. How to approximate non linear operations in the network, and how does this impact the training phase and accuracy of models?
4. How to compute homomorphic matrix multiplication in an efficient way?
5. How to handle data size which does not fit into one ciphertext?

Methods - Non linear operations in network

- Approximate non-linear activation functions with polynomials using Minimax approximation
  - Polynomial activations introduced new challenges in training phase:
    - Weight explosion
    - Slow convergence
    - Partially solved with fine-tuned weight initialization and learning rate
  - Polynomial activations might introduce errors during encrypted inference with respect to the original function:
    - For each layer, record the interval of intermediate results and approximate the function only on the interval
      - Lower degree of approximation needed
      - Higher accuracy
- Average Pooling and Sum Pooling in place of Max Pooling

Challenges

1. How to design an efficient and consistent data packing for network evaluation?
2. How to efficiently compute convolutions on encrypted data in a layer agnostic way?
3. How to approximate non linear operations in the network, and how does this impact the training phase and accuracy of models?
4. How to compute homomorphic matrix multiplication in an efficient way?
5. How to handle data size which does not fit into one ciphertext?

Methods - Matrix Multiplication

- Input matrix is transposed and row-flattened (same as data packing)
- Weight matrix in diagonal form
- Element wise multiplications, rotations and additions

$$\begin{bmatrix} a_{00} a_{01} a_{02} \\ a_{10} a_{11} a_{12} \\ a_{20} a_{21} a_{22} \end{bmatrix} \times \begin{bmatrix} w_{00} w_{01} w_{02} \\ w_{10} w_{11} w_{12} \\ w_{20} w_{21} w_{22} \end{bmatrix} = \begin{bmatrix} b_{00} b_{01} b_{02} \\ b_{10} b_{11} b_{12} \\ b_{20} b_{21} b_{22} \end{bmatrix}$$

Transpose and row-flatten

$$\begin{aligned} & [w_{00} w_{00} w_{00} w_{11} w_{11} w_{11} w_{22} w_{22} w_{22}] \odot \boxed{[a_{00} a_{10} a_{20} a_{01} a_{11} a_{21} a_{02} a_{12} a_{22}]} \\ + & [w_{10} w_{10} w_{10} w_{21} w_{21} w_{21} w_{02} w_{02} w_{02}] \odot [a_{01} a_{11} a_{21} a_{02} a_{12} a_{22} a_{00} a_{10} a_{20}] \\ + & [w_{20} w_{20} w_{20} w_{01} w_{01} w_{01} w_{12} w_{12} w_{12}] \odot [a_{02} a_{12} a_{22} a_{00} a_{10} a_{20} a_{01} a_{11} a_{21}] \\ = & \boxed{[b_{00} b_{10} b_{20} b_{01} b_{11} b_{21} b_{02} b_{12} b_{22}]} \end{aligned}$$

Same format

$$\sum_{j=0}^{d-1} \mathrm{Diag}_j(W) \odot \mathrm{Rotate}_{dj}(\mathrm{Flatten}(A^T)) \rightarrow O(d) \text{ rotations}$$

- Benefits:
  - Linear complexity
  - Fully compatible with data packing
  - Easily Parallelizable

Matrix Multiplication – cont'd

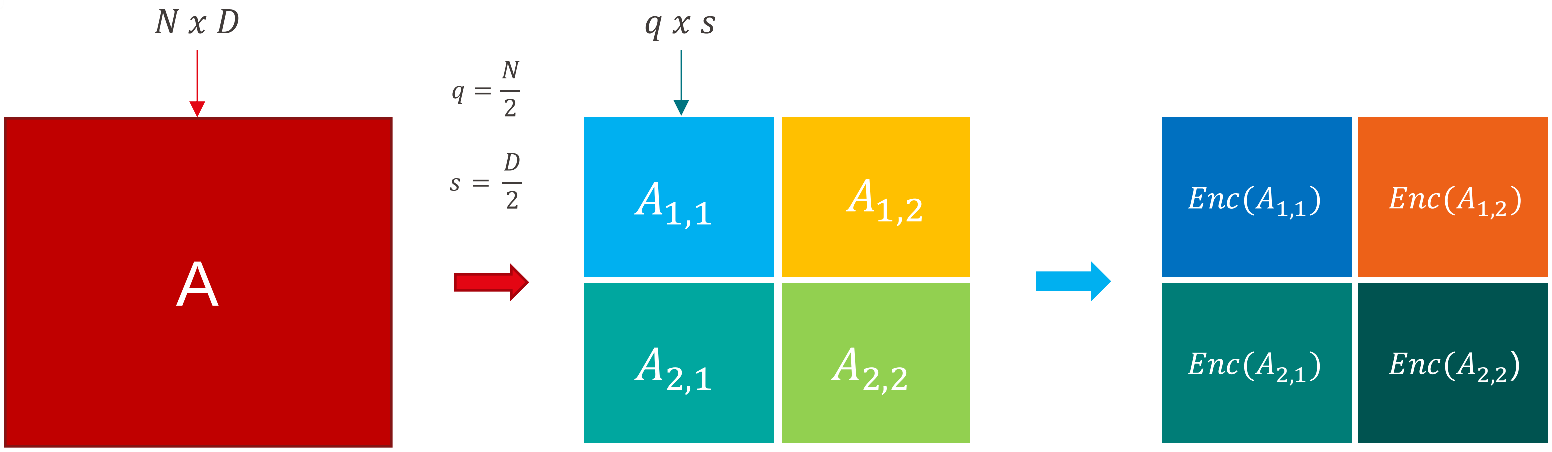
- Optimized version for $O\left(\frac{d}{2}\right)$
- Half the (intermediate) size of the matrices by packing real values in complex form
- Later remove the imaginary part with 1 addition and 1 rotation

$$\begin{pmatrix} a_{1,1} & \dots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \dots & a_{n,m} \end{pmatrix} \times \begin{pmatrix} b_{1,1} & \dots & b_{1,h} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \dots & b_{m,h} \end{pmatrix} \rightarrow \begin{pmatrix} a_{1,1} - ia_{1,2} & \dots & a_{1,m-1} - ia_{1,m} \\ \vdots & \ddots & \vdots \\ a_{n,1} - ia_{n,2} & \dots & a_{n,m-1} - ia_{n,m} \end{pmatrix} \times \begin{pmatrix} b_{1,1} + ib_{2,1} & \dots & b_{1,m} + ib_{2,m} \\ \vdots & \ddots & \vdots \\ b_{n-1,1} + ib_{n,1} & \dots & b_{n-1,m-1} + ib_{n,m} \end{pmatrix}$$

Challenges

1. How to design an efficient and consistent data packing for network evaluation?
2. How to efficiently compute convolutions on encrypted data in a layer agnostic way?
3. How to approximate non linear operations in the network, and how does this impact the training phase and accuracy of models?
4. How to compute homomorphic matrix multiplication in an efficient way?
5. How to handle data size which does not fit into one ciphertext?

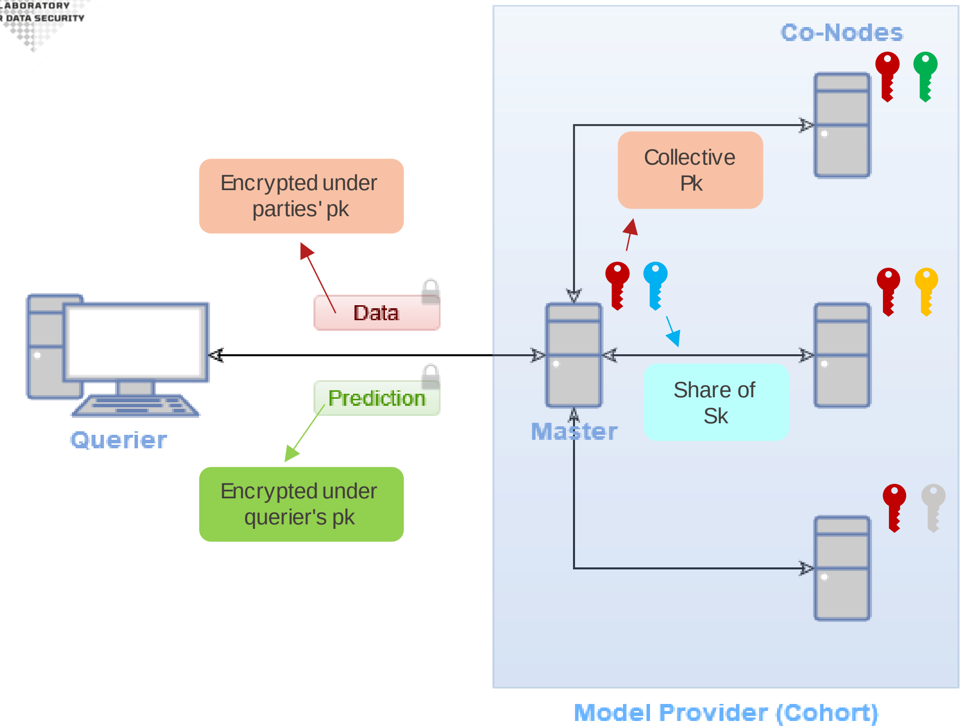
Methods – Block Matrix Arithmetics



- Each block (sub-matrix) is encrypted/encoded indipendently
- Operations can be carried out on each block using block matrix arithmetics

- Benefits:
  - Smaller ciphertexts/plaintexts
  - Easily parallelizable
  - Flexible: block splits can be adjusted to maximize latency or throughput

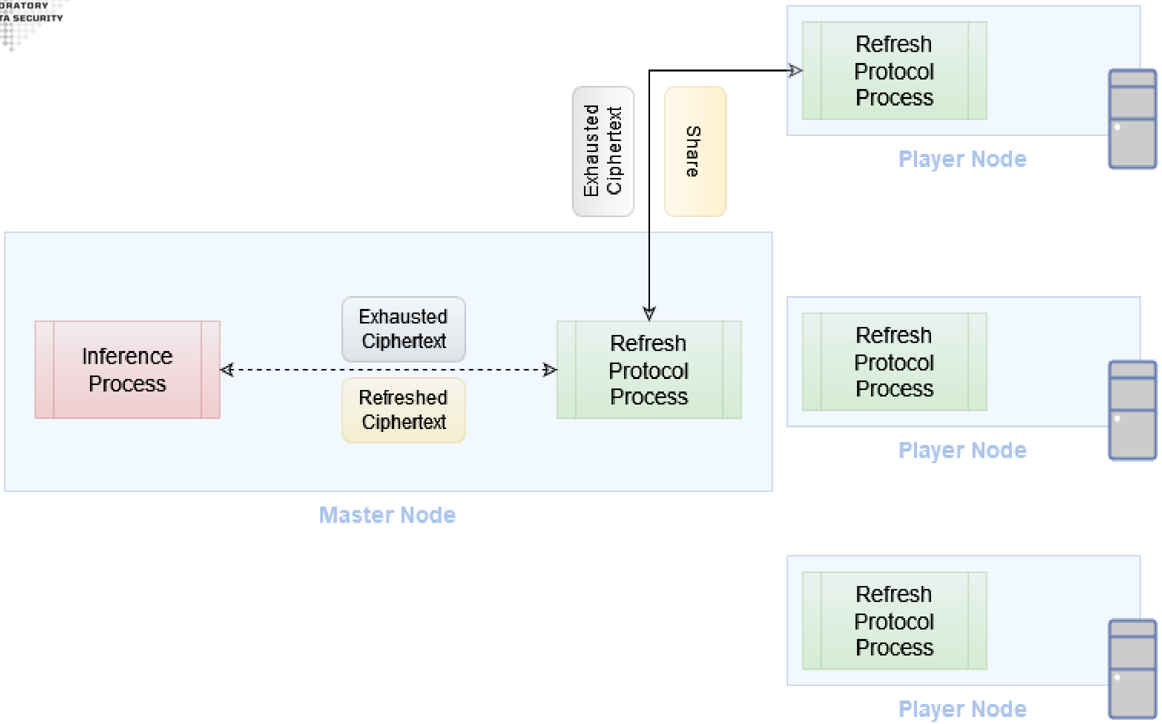
Methods – Distributed protocols for MHE



- Assumption: model is trained encrypted by a cohort of nodes.
- Very efficient: 1 Round Protocols!
- Reference: Mouchet et al.[2]

- [2] Multiparty Homomorphic Encryption from Ring-Learning-With-Errors

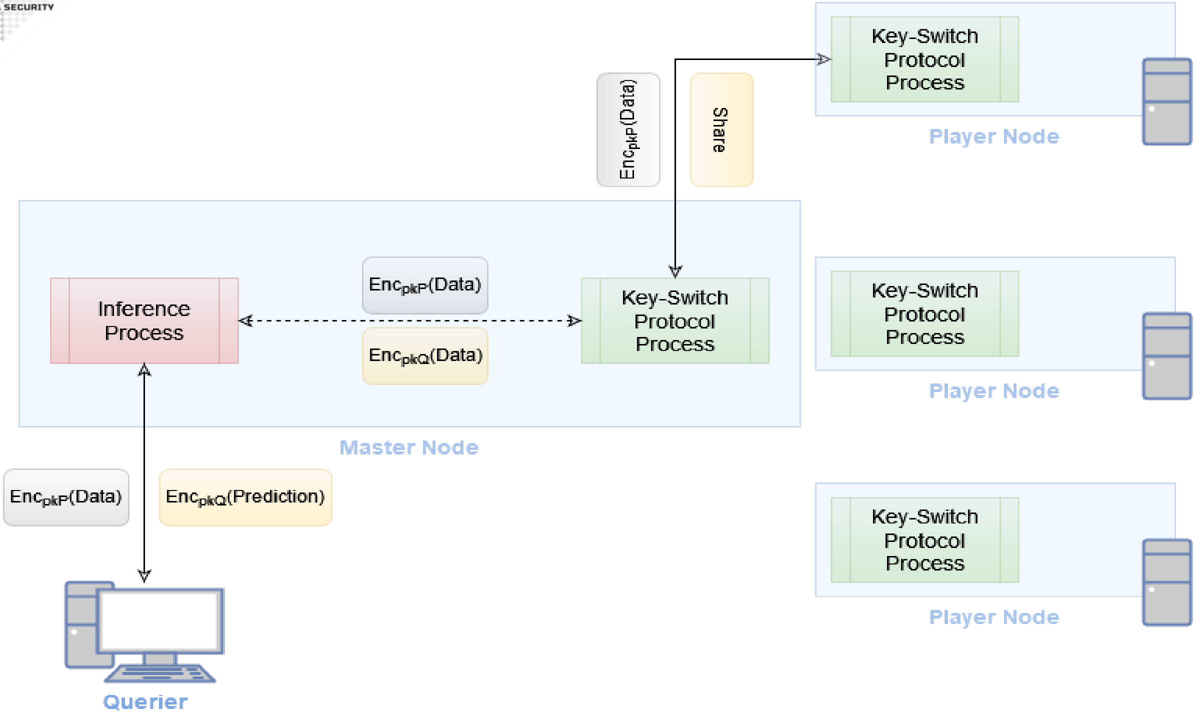
Methods – Distributed protocols for MHE - Refresh



- Distributed Bootstrapping generates a freshly encrypted ciphertext from a "low-level" ciphertext

- [2] Multiparty Homomorphic Encryption from Ring-Learning-With-Errors

Methods – Distributed protocols for MHE – Collective Key Switch



- Collective Key Switch Protocol is invoked to generate an encryption of the result under querier's public key

- [2] Multiparty Homomorphic Encryption from Ring-Learning-With-Errors

Experimental setup

LABORATORY FOR DATA SECURITY

- Hardware specifications:
  - 2x Intel Xeon E5-2680 2.5 GHz (48 threads)
  - 16 GB RAM DDR4
- Parameters for CKKS scheme:

Set	LogN (bits)	LogQP (bits)	Levels	Scale (bits)
CryptoNets	14	329	7	30
NN20	15	874	18	35
NN50	15	629	11	35
NN50-light	14	436	10	31
NN-Central-Btp	15	768	13	25

Experiment 1 - CryptoNets

- We benchmarked our framework on the Microsoft CryptoNets[3] model
  - Training and inference on MNIST dataset
  - Scenario is “model in clear – data encrypted”

Model	Batching	Batch	Latency(s)	Throughput(im/s)
CryptoNets	Y	4096	250	16.54
Faster Cryptonets [5]	N	1	39.1	0.02
LoLa [4]	N	1	2.2	0.5
MiniONN [6]	N	1	1.28	0.78
Ours	Y	83	10.5	7.9

[3] CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy
[4] Low Latency Privacy Preserving Inference
[5] Faster CryptoNets: Leveraging Sparsity for Real-World Encrypted Inference
[6] Oblivious Neural Network Predictions via MiniONN transformations

Experiment 2 – ZAMA NN

- We further tested our framework on deeper and more complex networks, using the models proposed by ZAMA[8]
  - Training and inference on MNIST dataset
  - Model is encrypted
  - Two models tested → 20 and 50 layers
  - Two bootstrapping approaches → centralized and distributed

[8] Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks

Experiment 2 – ZAMA NN – 20 layers

Setting	Batch	Latency (s)	Throughput (im/s)	Communication (MB/cipher)
1PC	292	639.6	0.45	-
MPC-5	292	365	0.85	21
MPC-10	292	417	0.70	21
Baseline-PC	1	115.52	0.008	-
Baseline-AWS	1	17.96	0.055	-

[4] Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks

Experiment 2 – ZAMA NN – 50 layers

Parameters with smaller ring size!

Setting	Batch	Latency (s)	Throughput (im/s)	Communication (MB/cipher)
1PC	292	1343	0.21	-
MPC-3	292	811	0.36	21
MPC-5	146	759	0.19	21
MPC-10	146	1007	0.14	21
Baseline-PC	1	233.55	0.004	-
Baseline-AWS	1	37.69	0.026	-

[4] Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks

EPFL

Final Remarks

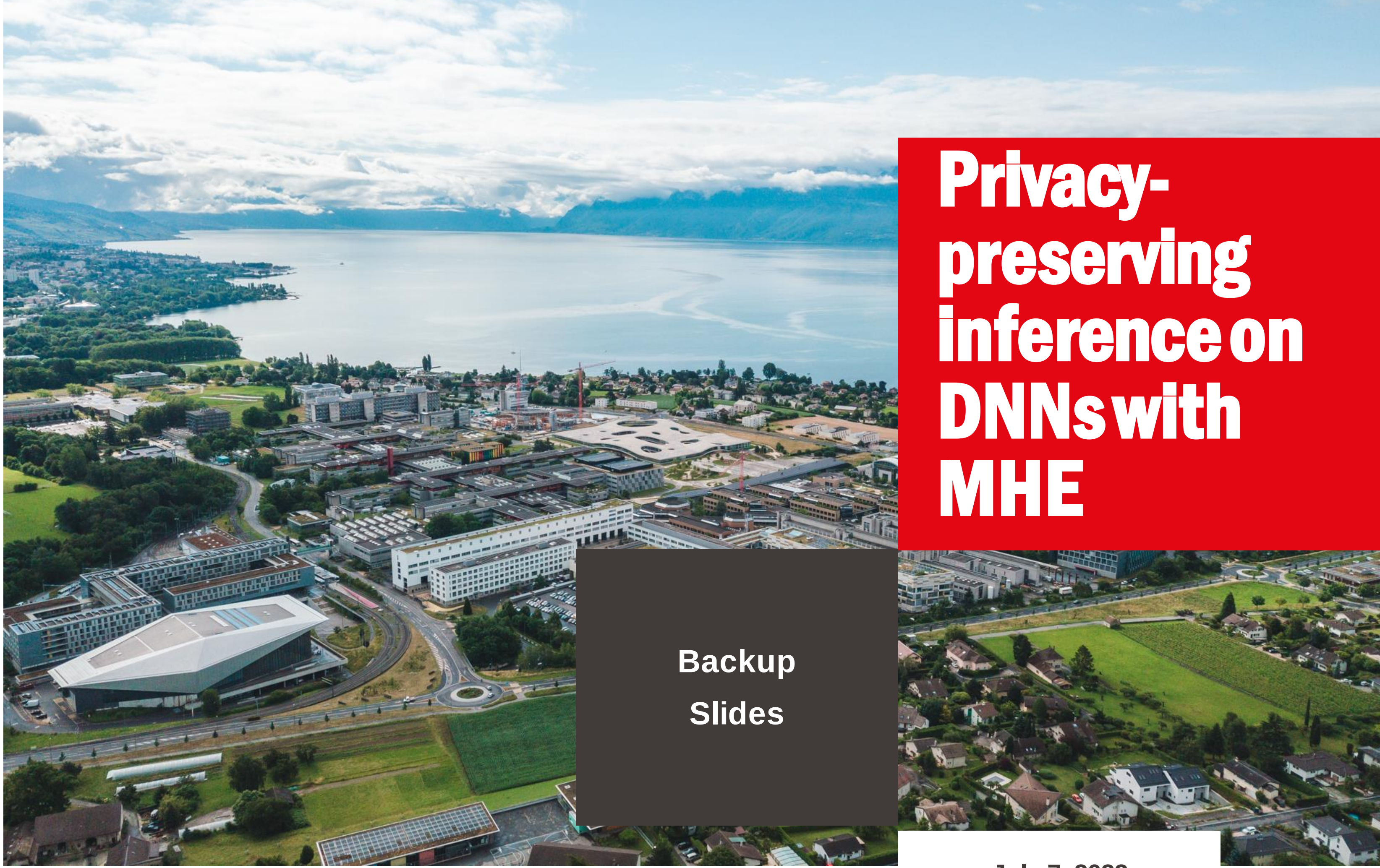
LABORATORY FOR DATA SECURITY

- Results are promising...
  - CKKS allows for data packing and SIMD operations for high throughput
- ...but optimization is hard!
  - Finding optimal data packing approach is crucial for performance
    - Data packing and block matrices splitting greatly influence latency and throughput
  - Dealing with non-linearities is complex
    - Introducing polynomial activations during training might cause slow or no convergence
    - Polynomial approximations must be fine-tuned (tradeoff between accuracy – efficiency)
  - <u>CKKS parametrization is hard!</u>
    - A bad choice for the parameters of the scheme might completely disrupt performance and accuracy!

EPFL

LABORATORY FOR DATA SECURITY

Privacy-preserving inference on DNNs with MHE

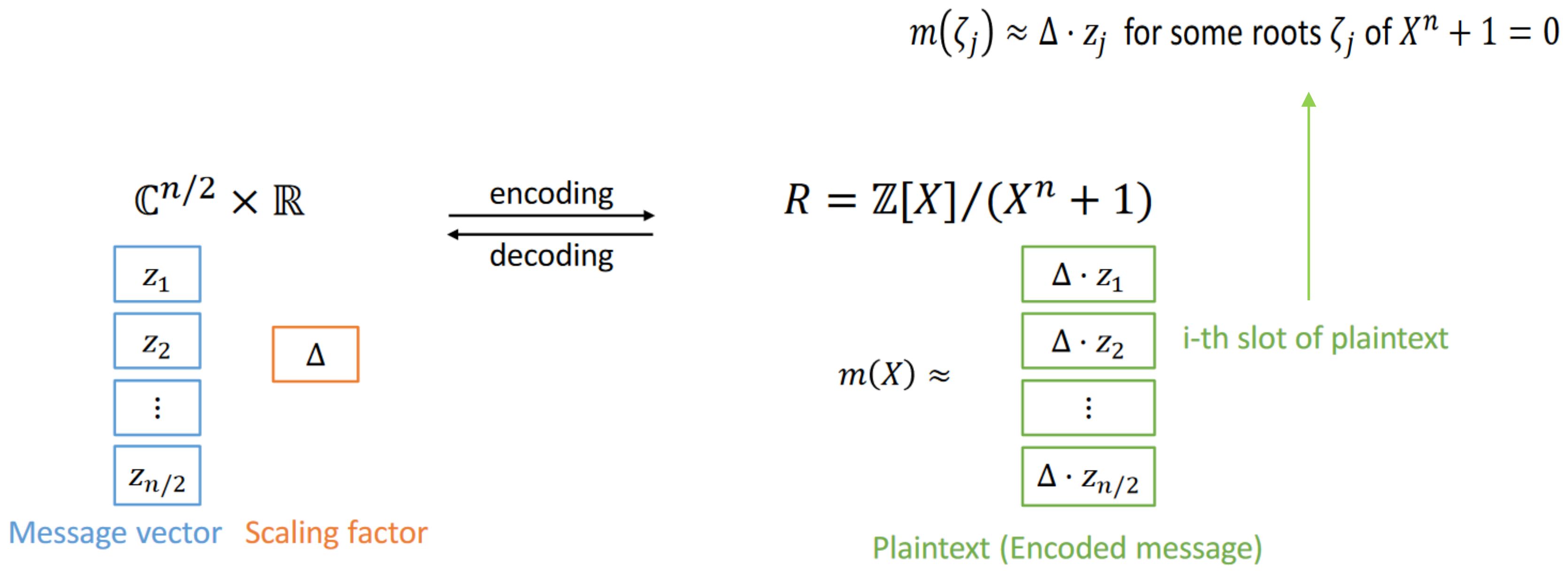


Backup Slides

July 7, 2022

École polytechnique fédérale de Lausanne

Background – CKKS scheme – Encoding & Decoding



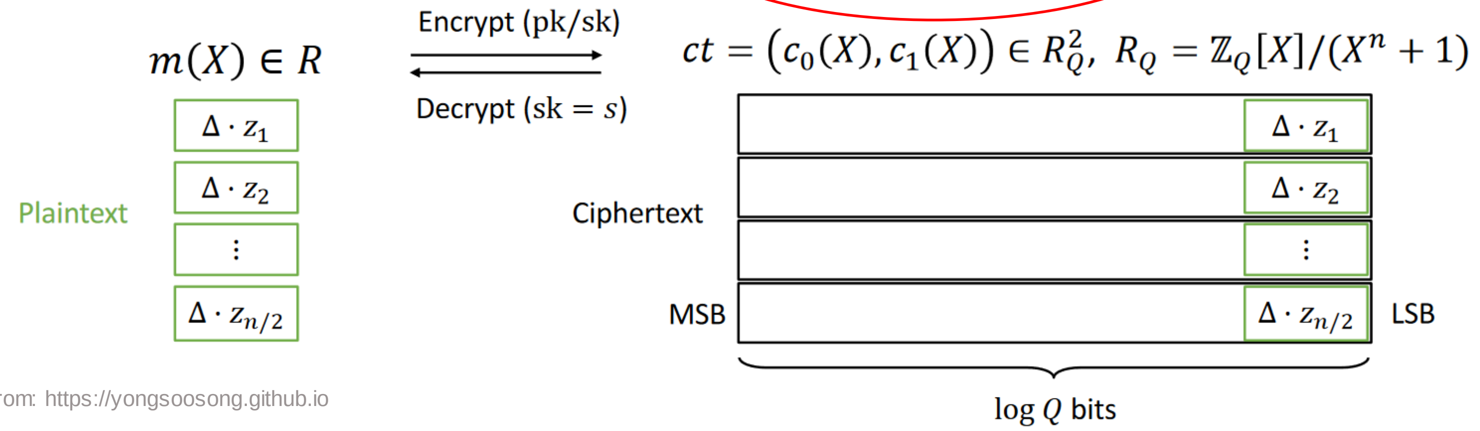
From: https://yongsoosong.github.io

Background – CKKS scheme – Encryption & Decryption

Modulus Chain

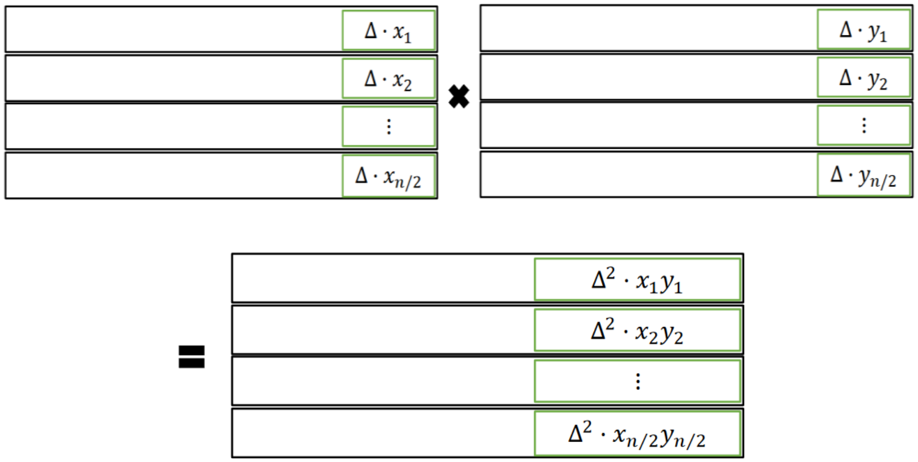
Ring structure $R_q = \mathbb{Z}_q[x]/(x^n + 1)$.

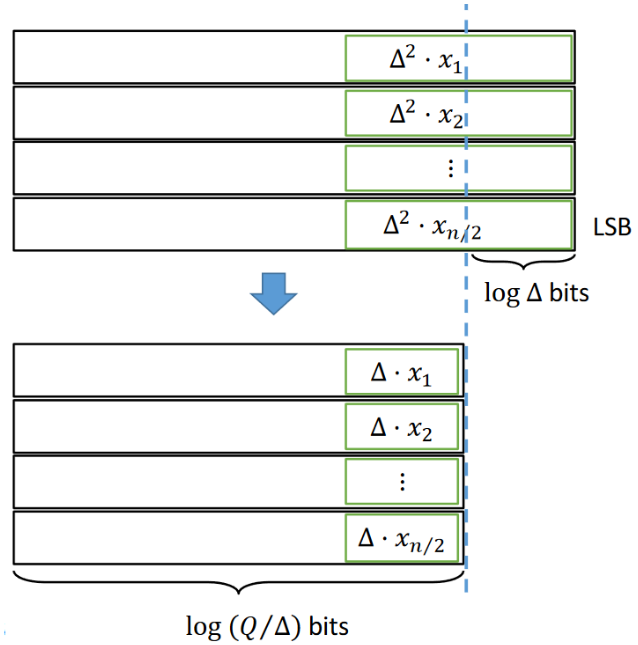
Residue Number System (RNS) : $\mathbb{Z}_q \cong \mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_L}$.



From: https://yongsoosong.github.io

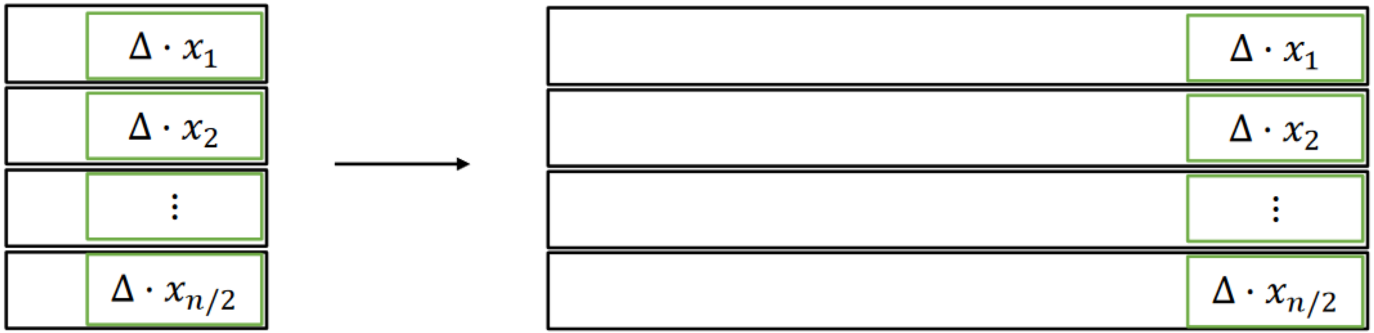
Background – CKKS scheme – Multiplication & Rescaling





From: https://yongsoosong.github.io

Background – CKKS scheme – Bootstrapping



Level l_0

Level $l_0 < l \leq L$

From: https://yongsoosong.github.io