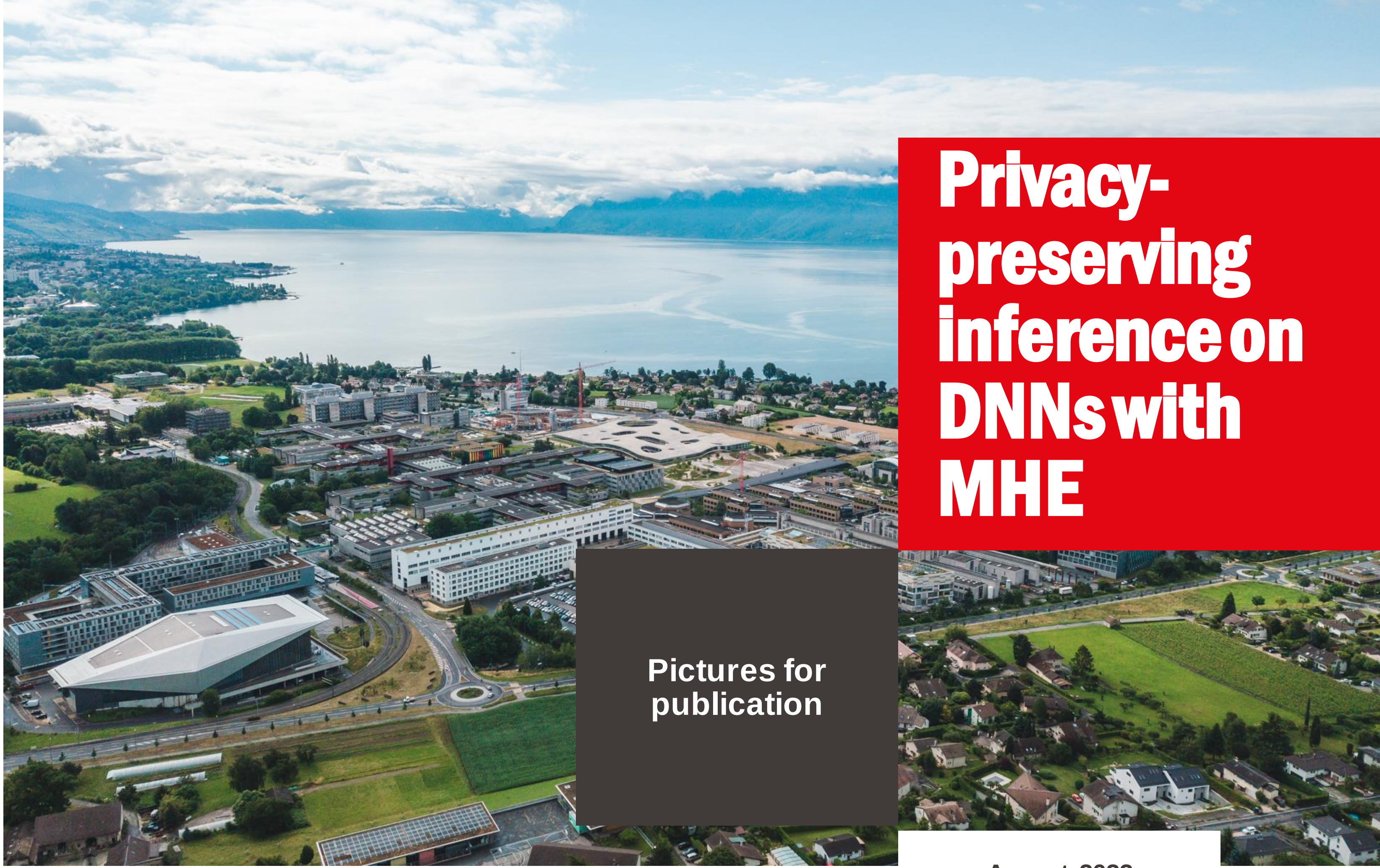
EPFL

LDS
LABORATORY
FOR DATA SECURITY



Privacy-preserving inference on DNNs with MHE

Pictures for publication

August 2022

École
polytechnique
fédérale
de Lausanne

EPFL

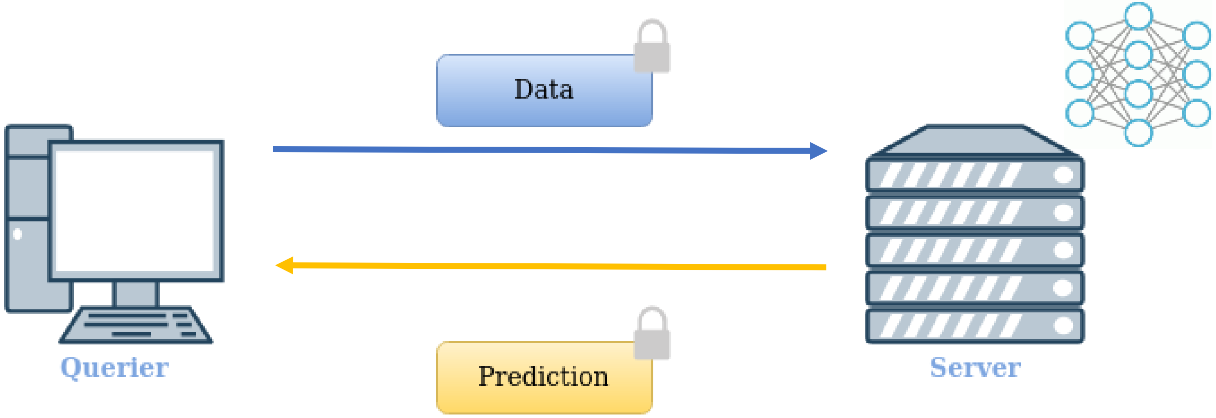
Experiments

LABORATORY FOR DATA SECURITY

- We can present 4 experiments:
  1. Cryptonet: Model in clear – data encrypted (usual HE inference)
  2. Cryptonet: Model encrypted – data in clear (model is exported to client. Model owner offers an oblivious decryption service)
  3. NN20: Model encrypted – data encrypted (Poseidon setting with MHE training and distributed protocols)
  4. NN50: Model in clear – data encrypted (usual HE inference with Centralized Bootstrapping)

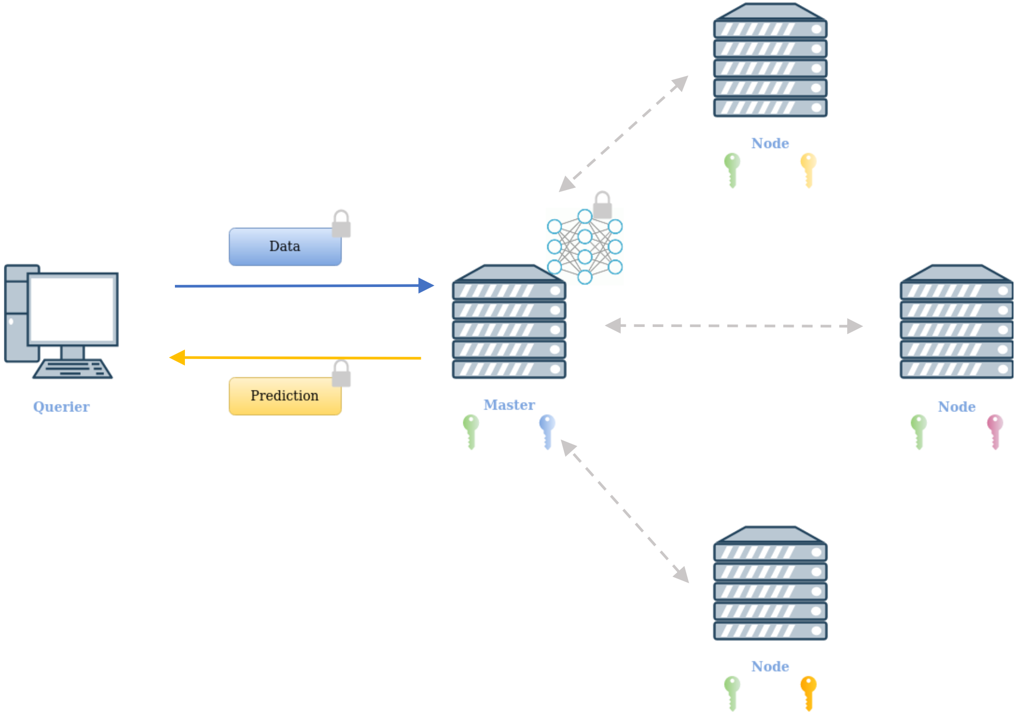
[1] POSEIDON: Privacy-Preserving Federated Neural Network Learning

Experiments – Scenario 1



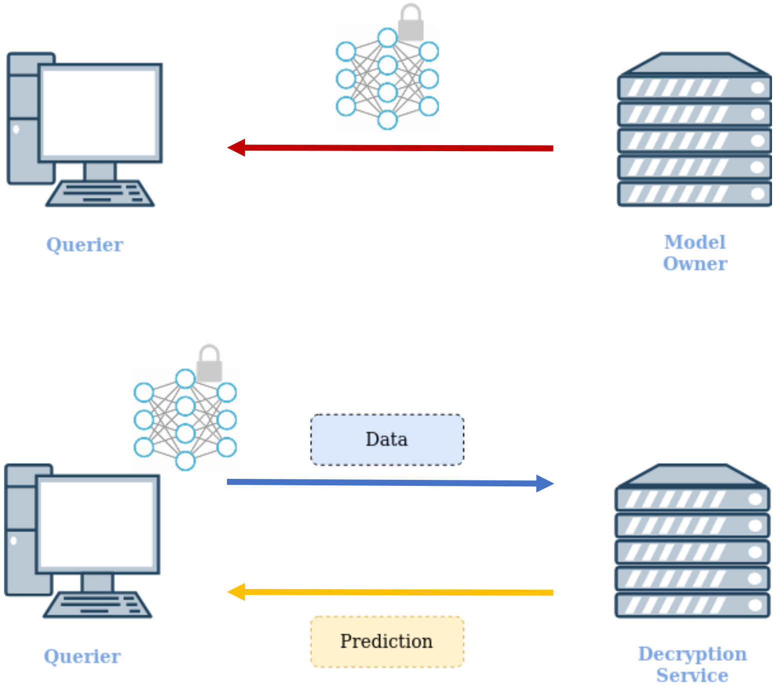
[1] POSEIDON: Privacy-Preserving Federated Neural Network Learning

Experiments – Scenario 2



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

Experiments – Scenario 3



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

Methods - Matrix Multiplication

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

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

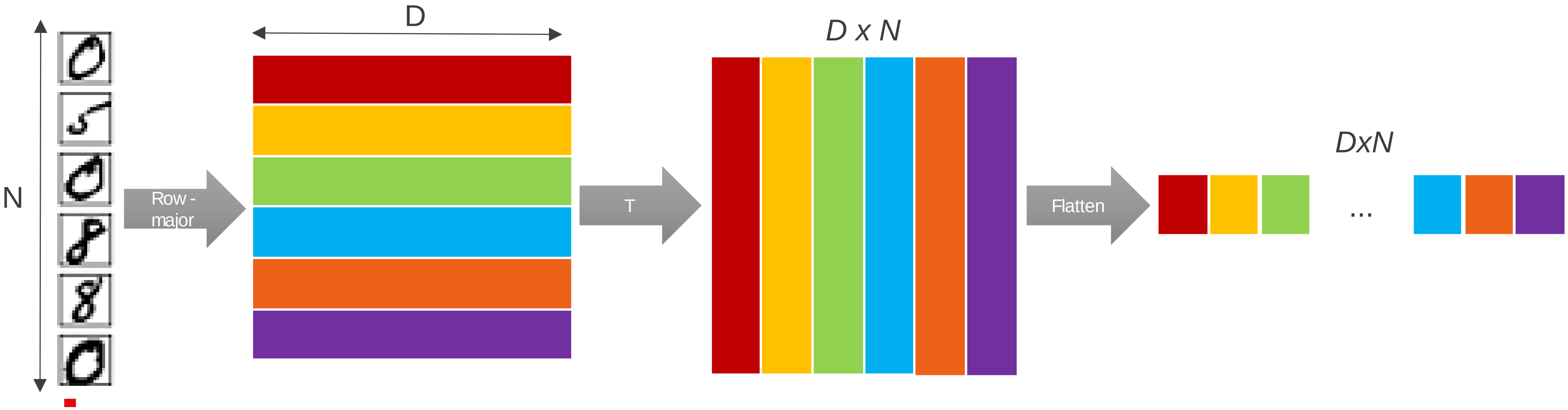
Same format

$$= \boxed{[b_{00} b_{10} b_{20} b_{01} b_{11} b_{21} b_{02} b_{12} b_{22}]}$$

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

Methods – Input matrix packing

- Querier holds batch of *N* images:
  - Pre-processing tasks (normalization,padding, etc...)
  - Images are represented as vectors of size *D* (row-major ordering)
  - The *NxD* matrix is transposed
  - The *NxD* matrix is row-flattened



Methods – Input matrix packing – complex trick

Input ciphertext gets packed with complex trick and then replicated for the multiplication (red)

$$\begin{bmatrix} a_{00} & \cdots & a_{0d} \\ \vdots & \ddots & \vdots \\ a_{n0} & \cdots & a_{nd} \end{bmatrix} \rightarrow [a_{00} \quad a_{10} \quad \dots \quad a_{n0} \quad a_{0d} \quad \dots \quad a_{nd}]$$

$$\downarrow$$

$$\text{Slots} \left\{ \begin{array}{l} [a_{00} + ia_{01} \quad \dots \quad a_{n0} + ia_{n1} \quad \dots \quad a_{0d-1} + ia_{0d} \quad \dots \quad a_{0d} + ia_{00} \quad \dots \quad a_{nd} + ia_{n0}], \\ [a_{00} + ia_{01} \quad \dots \quad a_{n0} + ia_{n1} \quad \dots \quad a_{0d-1} + ia_{0d} \quad \dots \quad a_{0d} + ia_{00} \quad \dots \quad a_{nd} + ia_{n0}], \\ [a_{00} + ia_{01} \quad \dots \quad a_{n0} + ia_{n1} \quad \dots \quad a_{0d-1} + ia_{0d} \quad \dots \quad a_{0d} + i0 \quad \dots \quad a_{nd} + i0 \quad \dots 0] \end{array} \right\}$$

Methods – Weight matrix diagonal packing

Weight matrix gets diagonalized (generalized diagonals for non-square) and the diagonals are then packed together with complex trick. Additionally every element in the diagonal is replicated a number of times equal to the number of rows of the input matrix for multiplication correctness

$$\begin{bmatrix} w_{00} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \end{bmatrix} \rightarrow \begin{array}{l} \overbrace{}^{Rows(A)\,\times\,Cols(W)} \\ \overbrace{}^{Rows(A)} \\ d_0 = (w_{00} - iw_{10} \quad \cdots \quad w_{03} - iw_{13} \quad \ldots) \\ d_1 = (w_{20} - i0 \quad \cdots \quad w_{23} - i0 \quad \ldots) \end{array} \; (Rows(W) + 1)/2$$

Methods - Convolutional layer «linearization»

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

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

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

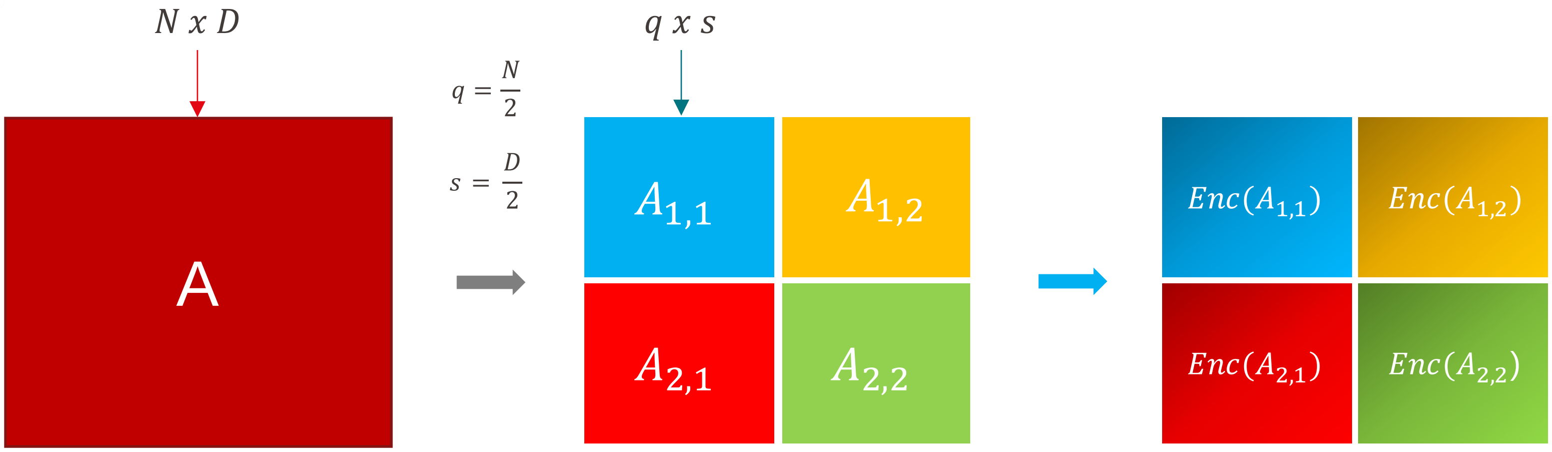
Methods - Convolutional layer «linearization» cont'd

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

$$\begin{bmatrix} T(k_1^{(ch_1)}) & \cdots & T(k_1^{(ch_M)}) \\ \vdots & \ddots & \vdots \\ T(k_f^{(ch_1)}) & \cdots & T(k_f^{(ch_M)}) \end{bmatrix}$$

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

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