RUB

RUHR-UNIVERSITÄT BOCHUM

Fast Homomorphic Evaluation of Deep Neural Networks

FHE–DiNN — Privacy-Preserving Image Classification in the Cloud

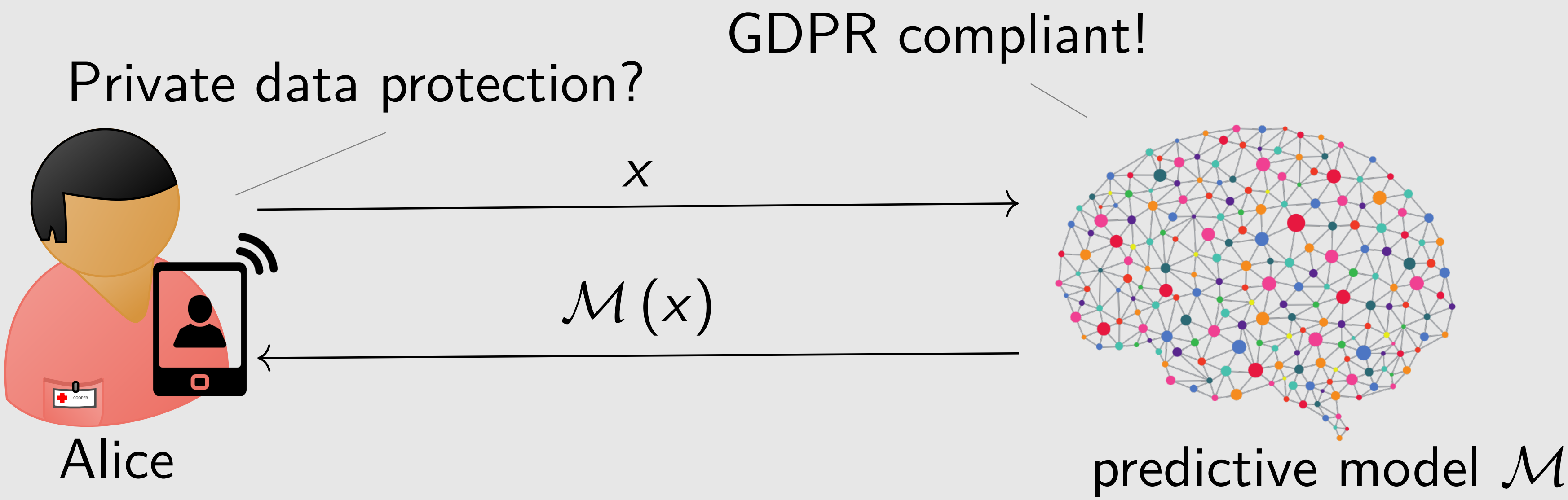
Realworld Cryptanalysis Seminar, RUB, 17.7.2018

Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier
Cryptology and IT-Security, Ruhr-Universität Bochum

Horst Görtz Institut für IT-Sicherheit

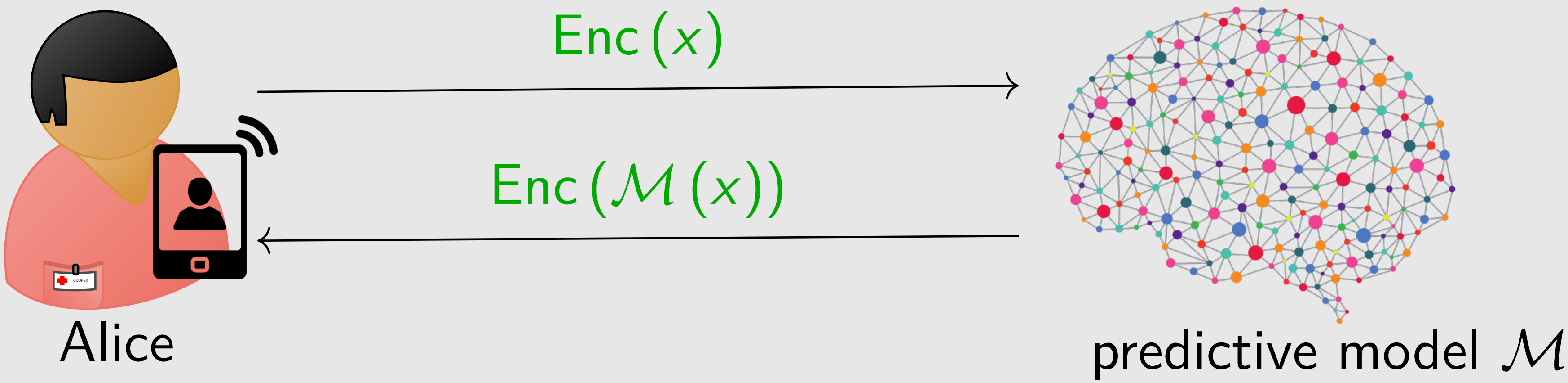
Machine Learning as a Service (MLaaS)

User submits x and recovers $\mathcal{M}(x)$; the prediction.



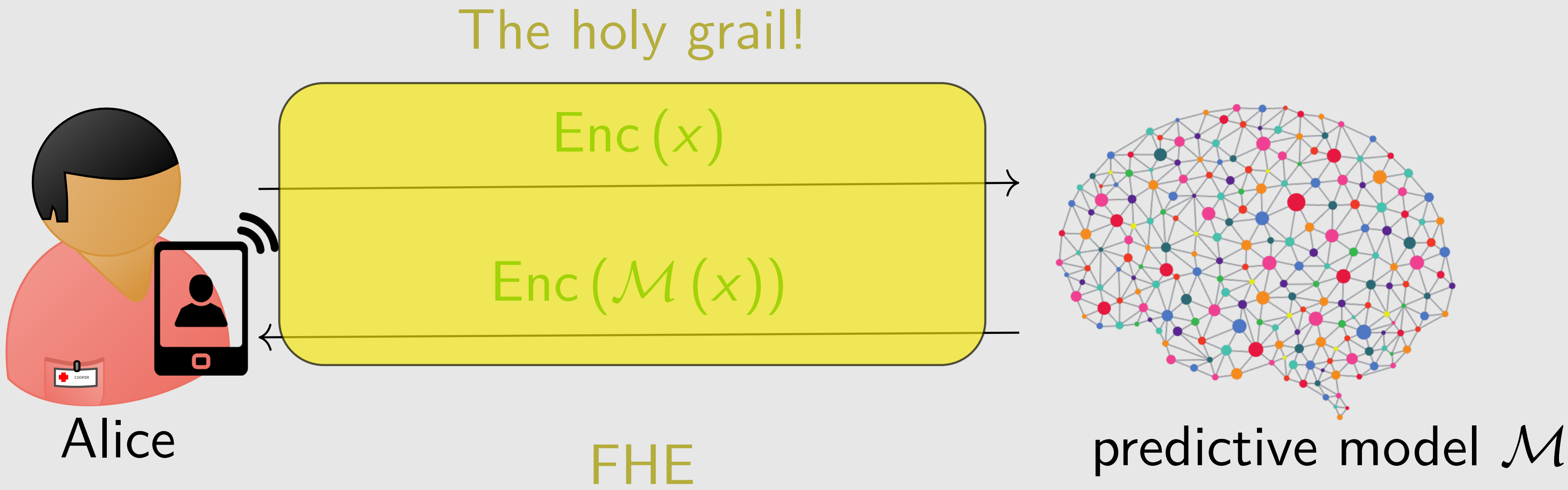
Machine Learning as a Service (MLaaS)

User submits $\mathrm{Enc}\,(x)$ and recovers $\mathrm{Enc}\,(\mathcal{M}\,(x))$; the encrypted prediction.



Machine Learning as a Service (MLaaS)

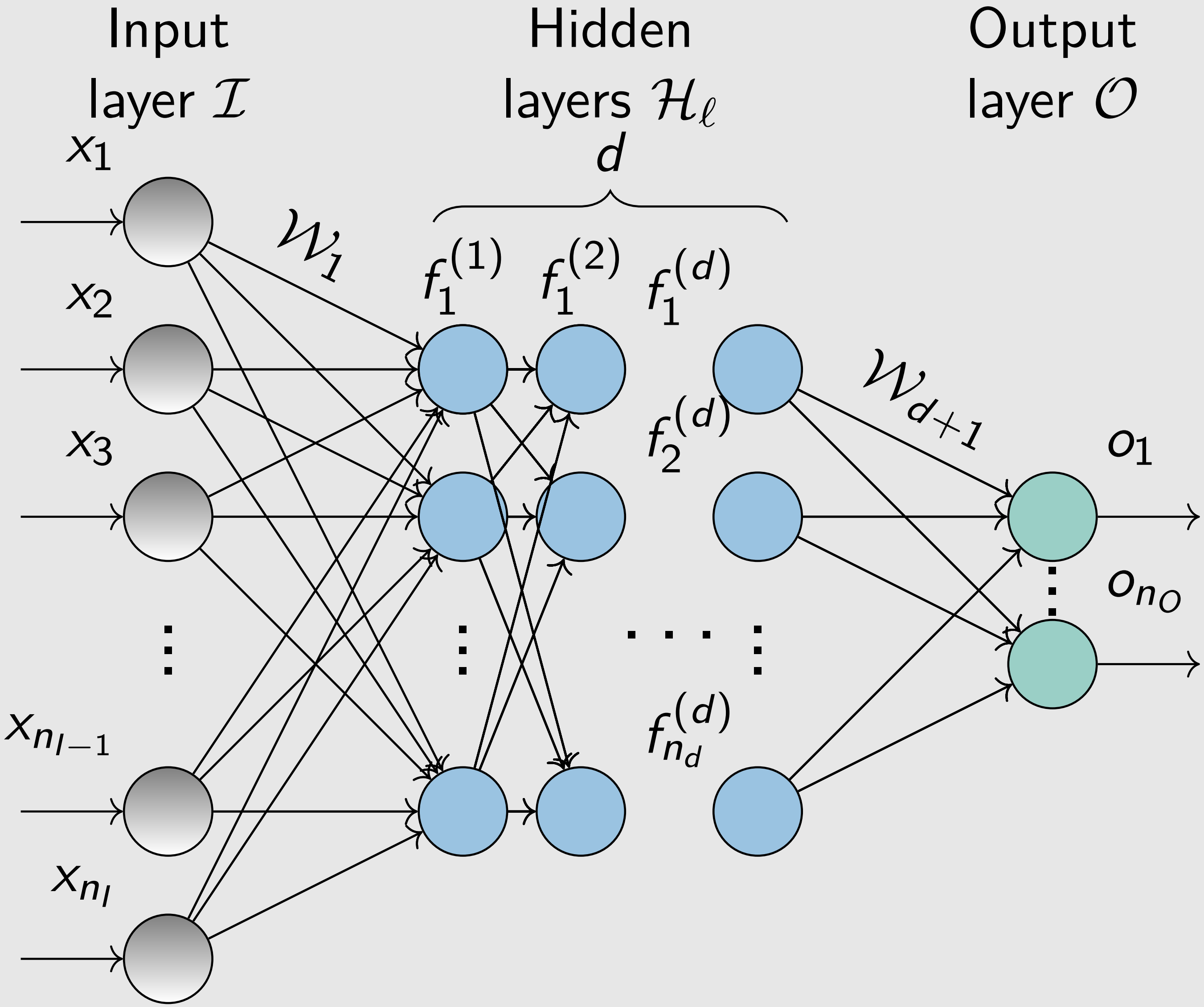
User submits $\mathrm{Enc}(x)$ and recovers $\mathrm{Enc}(\mathcal{M}(x))$; the encrypted prediction.



- ✓ Privacy: input & output data is encrypted (user has only key)
- ◆ Efficiency: is a central practical issue

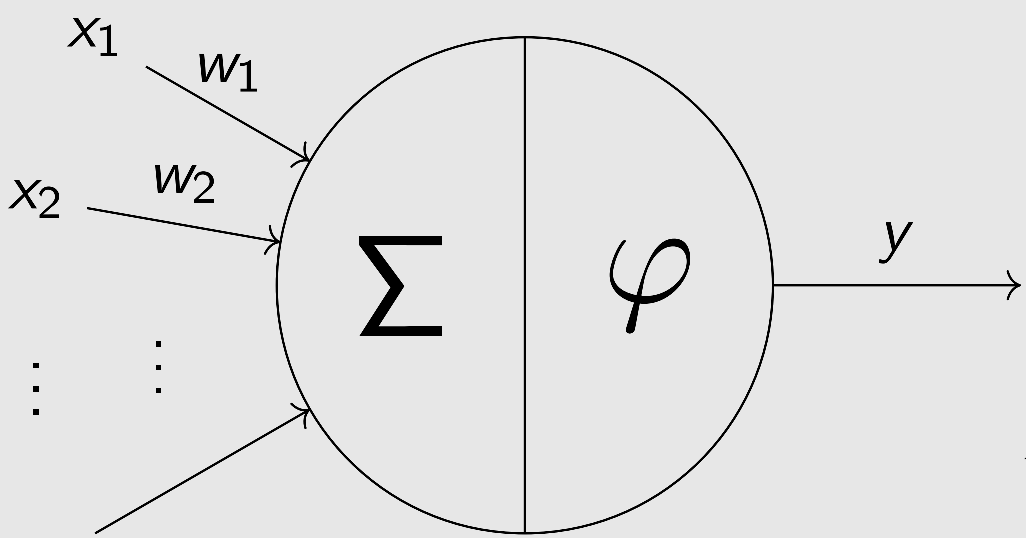
Goal: FHE–DiNN — fast homomorphic evaluation of neural networks.

Deep Neural Network



Close-up on Neuron

Computation for every neuron:



$x_i, w_i, y \in \mathbb{R}$

$$y = \varphi\left(\sum_i w_i\, x_i\right),$$

where φ is an activation function.

FHE-friendly Discretized Neural Networks

Goal: FHE-friendly model of neural network: $x_i, w_i, y \in \mathbb{Z}$.

Definition (DiNN)

A neural network whose layers have inputs in $\{-I, \ldots, I\} \subseteq \mathbb{Z}$, weights in $\{-W, \ldots, W\} \subseteq \mathbb{Z}$, for $I, W \in \mathbb{N}$, and each neuron's activation function maps the weighted sum to integer values in $\{-I, \ldots, I\} \subseteq \mathbb{Z}$.

- Not restrictive as it seems as, e.g., binarized NNs perform well;
- trade-off between size and performance;
- conversion is straight-forward easy.

Main Idea: Activation During FHE Bootstrapping

Combine necessary refreshing with desirable activation function:

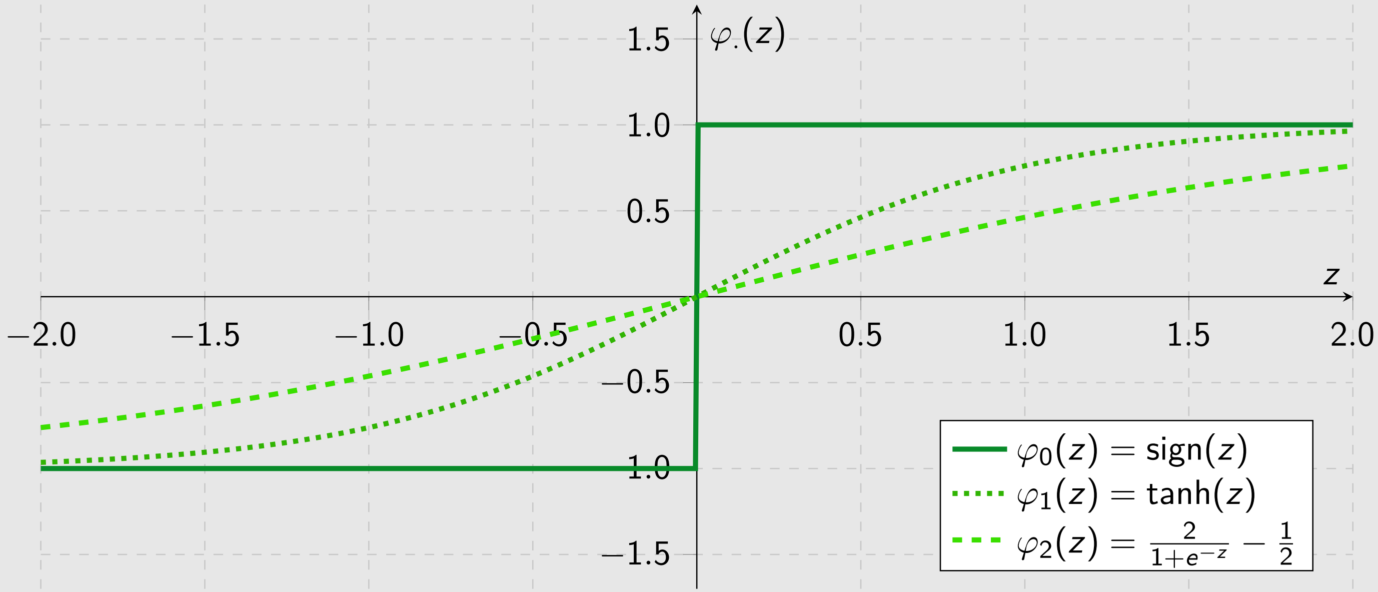
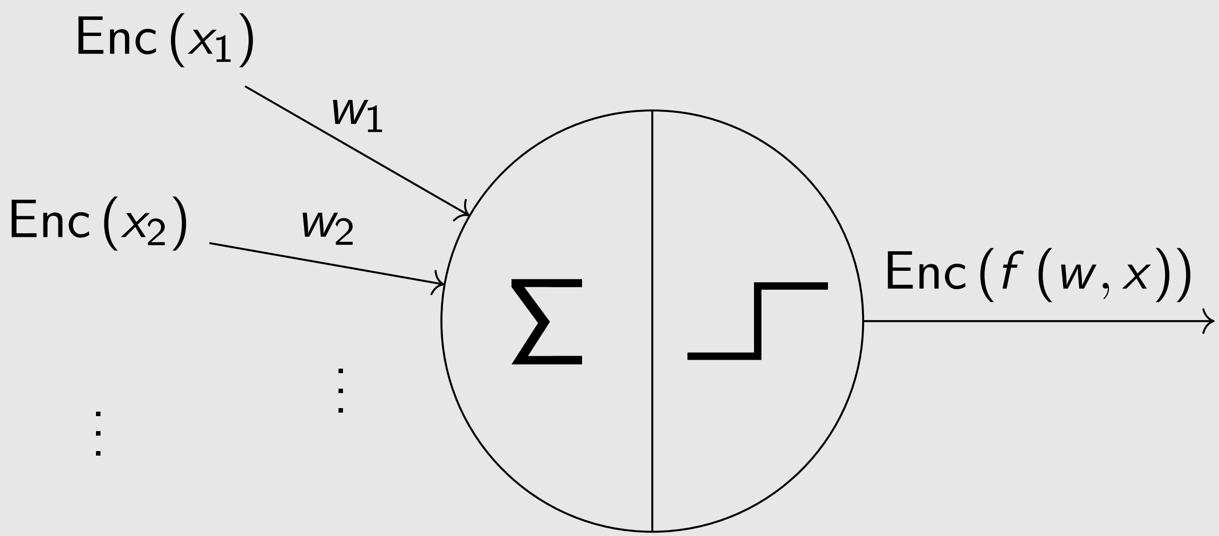


Figure: Several neural network activation functions and our choice φ_0.

$$\mathrm{Enc}(z) \rightarrow \mathrm{Enc}(f(z)) \rightarrow \ldots$$

Close-up on a single neuron: two steps



Each neuron computes $\mathrm{Enc}(f(w,x))$, *e.g.* $\mathrm{Enc}(\mathrm{sign}(\langle w, x\rangle))$:

1. Compute inner product $\sum_i w_i \mathrm{Enc}(x_i)$ (linear homomorphic)
2. Bootstrap encryption of activated result (fully homomorphic)

Adapting TFHE framework for fast bootstrapping

We use Torus Fully Homomorphic Encryption framework [1] on $\mathbb{T} := \mathbb{R}/\mathbb{Z}$.

Security assumption in TFHE

Hardness of Learning with Errors (LWE) on $\mathbb{T}$:

$$(\mathbf{a},\ \langle \mathbf{s}, \mathbf{a} \rangle + e \mod 1) \overset{c}{\approx} (\mathbf{a},\ \mathbf{u}) \in \mathbb{T}^{n+1},$$

where $e \leftarrow \chi_\alpha$, $\mathbf{s} \leftarrow_\$ \{0,1\}^n$, $\mathbf{a}, \mathbf{u} \leftarrow_\$ \mathbb{T}^n$ with error parameter α.

We also use other torus-based schemes allowing performance increase:

- TLWE (for encrypting polynomials $\mathbb{T}[X]$)
- TGSW ('matrix TLWE'; roughly equivalent to GSW construction)

[1] [CGGI16,CGGI17]

Refining TFHE

1. Combining implementations of `Bootstrapping` and `Activation`
2. Reducing bandwidth usage by `Packing` ciphertexts
3. Moving internal operation order, i.e., when to do a `Keyswitch`
4. `Reparametrizing` message space between neural network layers
5. Optimizing alternative implementation of `BlindRotate`

Goal `Packing`: encrypt polynomial $\mathbb{T}[X]$ instead of $\mathbb{T}$ scalars: $c(X) = \text{TLWE.Encrypt}\left(\sum_i x_i X^i\right)$.

Idea Redefine and pack (clear) weights in hidden layers: $w(X) := \sum_i w_i X^{-i}$.

Effect Constant term of $c(X) \cdot w(X) \in \mathbb{T}[X]$ is $\sum_i w_i x_i \in \mathbb{T}$.

Refining TFHE

1. Combining implementations of `Bootstrapping` and `Activation`
2. Reducing bandwidth usage by `Packing` ciphertexts
3. Moving internal operation order, i.e., when to do a `Keyswitch`
4. `Reparametrizing` message space between neural network layers
5. Optimizing alternative implementation of `BlindRotate`

Goal Reduce LWE dimension, ensuring security level, to optimize memory, efficiency, bootstrapping–key's size, final noise, and number of expensive external products.

Idea Bootstrapping := KeySwitch ► BlindRotate ► Extract

Effect Less noise; size $n < N$ is used only for bootstrapping

Refining TFHE

1. Combining implementations of `Bootstrapping` and `Activation`
2. Reducing bandwidth usage by `Packing` ciphertexts
3. Moving internal operation order, i.e., when to do a `Keyswitch`
4. `Reparametrizing` message space between neural network layers
5. Optimizing alternative implementation of `BlindRotate`

Goal Dynamically change the message space to reduce errors.

Idea For B_ℓ, an upper bound on the sum in layer $\ell + 1$, define:

$$\texttt{testVector}(X) = \frac{-1}{2B_\ell + 1} \sum_{i=0}^{N-1} X^i.$$

Effect Less slices, hence less inaccurate decisions when rounding.

Refining TFHE

1. Combining implementations of `Bootstrapping` and `Activation`
2. Reducing bandwidth usage by `Packing` ciphertexts
3. Moving internal operation order, i.e., when to do a `Keyswitch`
4. `Reparametrizing` message space between neural network layers
5. Optimizing alternative implementation of `BlindRotate`

We unfold[a] the loop for computing $X^{\langle \vec{s}, \vec{a} \rangle}$ in `BlindRotate`.

Goal Trade-off off-line pre-processing for on-line speed.

Idea Windowed processing & using algebraic keys-relations.

Effect Larger bootstrapping key traded for faster execution.

[a] Following [ZYL+17]

Digit Recognition & Classification

We showcase a solution to the problem of **blind** digit recognition.

Dataset: MNIST (60 000 images in training set + 10 000 in test set).

FHE–DiNN Demo: Input and Neural Network

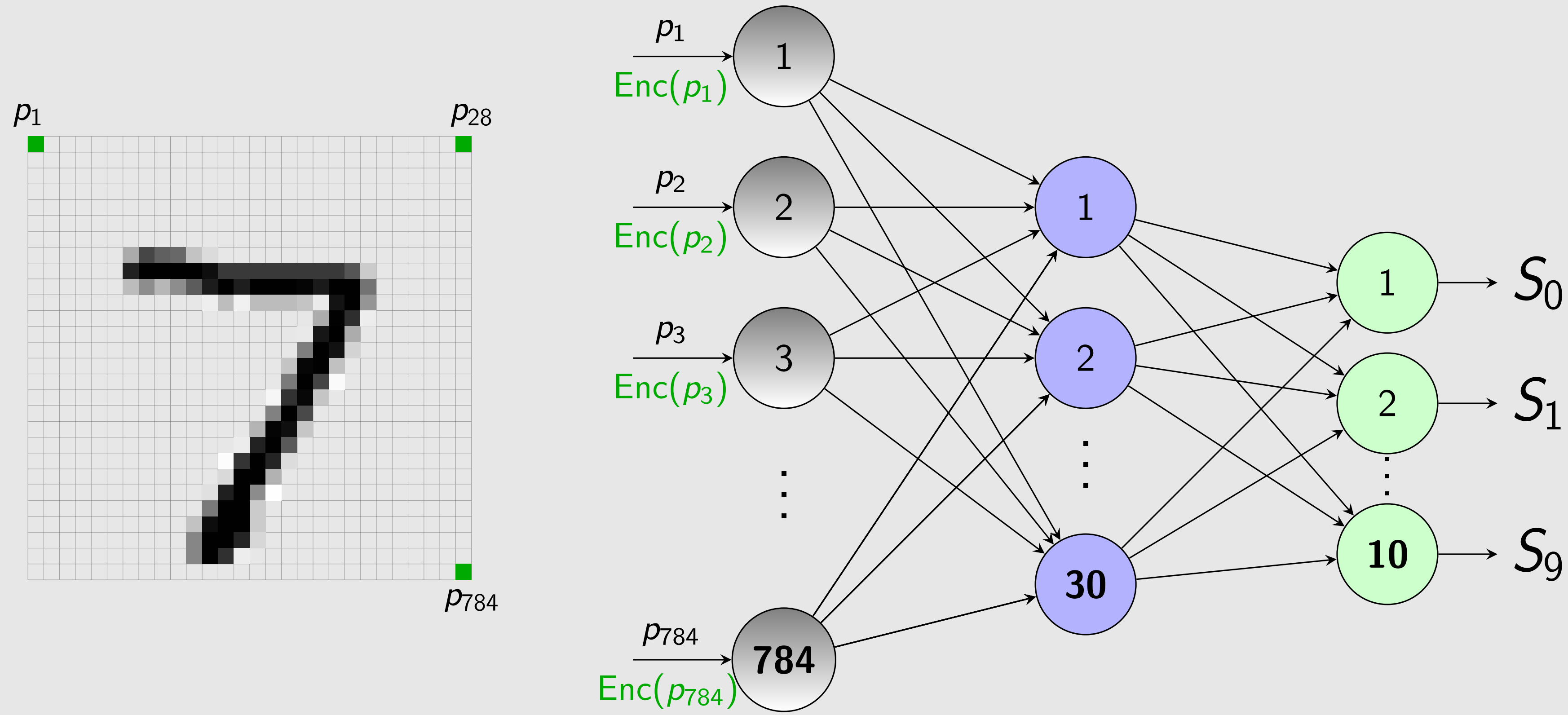
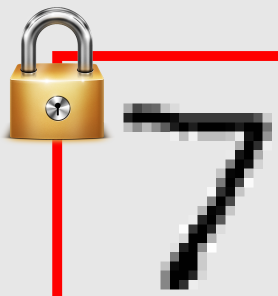


Figure: FHE–DiNN processes an MNIST image given a pre-trained neural network with 784:100:10–topology. Edges are labeled with weights, connect output with input nodes, which are functionally dependent on previous ones.

FHE–DiNN Demo: Algorithm Step-by-step

Evaluation of a server's DiNN, i.e. 100 neurons in one hidden layer.

Client



Alice

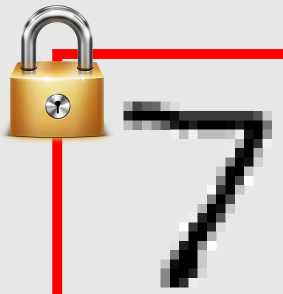
Server

predictive model $\mathcal{M}$

FHE–DiNN Demo: Algorithm Step-by-step

Evaluation of a server's DiNN, i.e. 100 neurons in one hidden layer.

Client | **Server**



$$\xrightarrow{\mathrm{Enc}\left(\sum_i p_i X^i\right)} \mathrm{TLWE}_N \xrightarrow{\cdot \sum_i w_i X^{-i}} 100\ \mathrm{TLWE}_N$$

$$100\ \mathrm{TLWE}_N \xrightarrow{\texttt{Extract}} 100\ N\text{-LWE} \xrightarrow{\texttt{KeySwitch}} 100\ n\text{-LWE} \xrightarrow{\text{Bootstrap2Sign}} 100\ N\text{-LWE} \xrightarrow{\texttt{Scores}} 10\ N\text{-LWE}$$

$$7 \xleftarrow{\text{argmax}} 10 \text{ scores} \xleftarrow{\texttt{Dec}} 10\ N\text{-LWE}$$

Experimental results

Performance on (clear) inputs x

	Original NN	DiNN + hard_sigmoid	DiNN + sign
30 neurons	94.76%	93.76% (-1 %)	93.55% (-1.21%)
100 neurons	96.75%	96.62% (-0.13%)	96.43% (-0.32%)

Performance on (encrypted) inputs $\mathsf{Enc}(x)$

	Acc.	Disagreements	Total wrong BS	when dis.	Time
30	93.71%	273 (105–121)	3383/300000	196/273	0.515 s
100	96.26%	127 (61–44)	9088/1000000	105/127	1.679 s
30 w	93.46%	270 (119–110)	2912/300000	164/270	0.491 s
100 w	96.35%	150 (66–58)	7452/1000000	99/150	1.640 s

window size w = 2

Conclusions and future directions

- Better DiNNs through refined conversion (+ retraining)
- GPU implementation for realistic timings
- Showcase more cognitive models (e.g., convolutional NNs, ...)
- Apply advanced crypto to other machine learning problems

Open Problems [1]

How to evaluate complex, non-linear functions such as $\max(x_1, x_2)$, hence $\mathrm{ReLU}(x) = \max(0, x)$, or vectorial $\mathrm{Softmax}(\vec{x})$ in FHE–DiNN?