Function privacy for GSW and efficient sign computation from TFHE

Florian Bourse

Orange Labs, Applied Crypto Group, Cesson-Sévigné, France

LATCA 2018 — Bertinoro

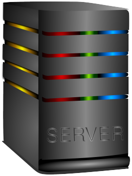
1. FHE Circuit Privacy Almost for Free [BPMW16]

2. Fast Homomorphic Evaluation of Deep Discretized Neural Networks [BMMP18]

Bourse, del Pino, Minelli, Wee

FHE Circuit Privacy Almost for Free

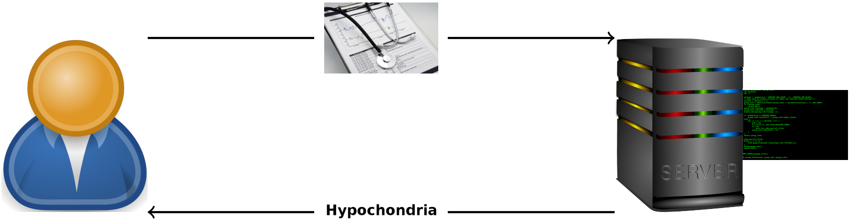
The incentive



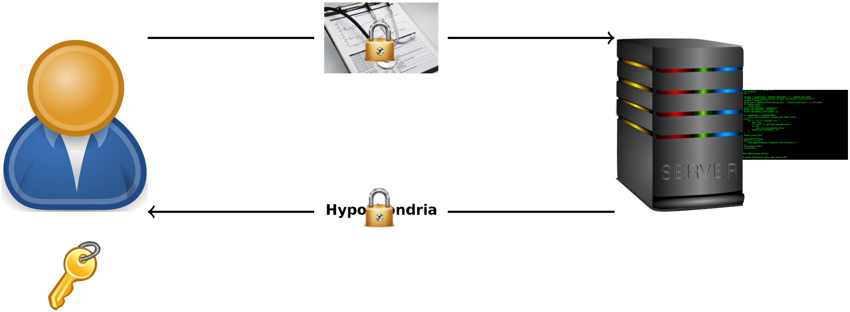
The incentive

The incentive

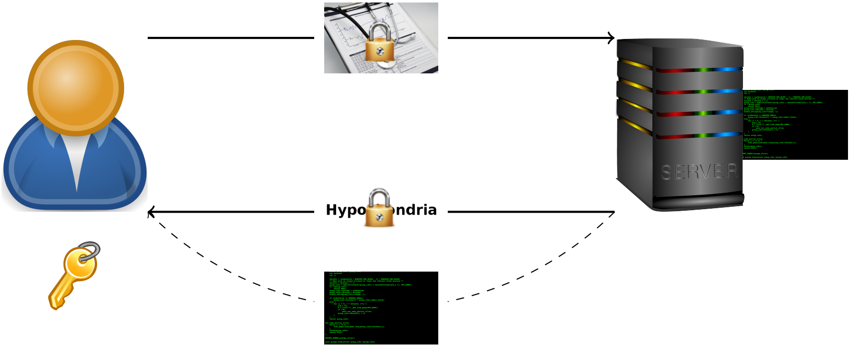
The incentive



The incentive



The incentive



The reminder

LWE

$$\begin{pmatrix} \mathbf{A} \\ \mathbf{s}\mathbf{A} + \mathbf{e} \end{pmatrix} \text{ looks uniform}$$

The reminder

GSW

$$\mathbf{G} = \mathbf{Id}_n \otimes \mathbf{g}, \quad \mathbf{g} = \left(1, 2, \ldots, 2^k\right)$$

$$\mathbf{C} = \begin{pmatrix} \mathbf{A} \\ \mathbf{s}\mathbf{A} + \mathbf{e} \end{pmatrix} + \mu \mathbf{G} \in \mathbb{Z}_q^{n \times m}$$

The reminder

GSW

$$\mathbf{G} = \mathbf{Id}_n \otimes \mathbf{g}, \quad \mathbf{g} = \left(1, 2, \ldots, 2^k\right)$$

$$\mathbf{C} = \begin{pmatrix} \mathbf{A} \\ \mathbf{sA} + \mathbf{e} \end{pmatrix} + \mu\mathbf{G} \in \mathbb{Z}_q^{n \times m}$$

Sum $\mathbf{C}_1 + \mathbf{C}_2$

The reminder

GSW

$$\mathbf{G} = \mathbf{Id}_n \otimes \mathbf{g}, \quad \mathbf{g} = \left(1, 2, \ldots, 2^k\right)$$

$$\mathbf{C} = \begin{pmatrix} \mathbf{A} \\ \mathbf{s}\mathbf{A} + \mathbf{e} \end{pmatrix} + \mu \mathbf{G} \in \mathbb{Z}_q^{n \times m}$$

Sum $\mathbf{C}_1 + \mathbf{C}_2$

Product $\mathbf{C}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2)$

where $\forall\, \mathbf{v} \in \mathbb{Z}_q^n$, $\mathbf{G}^{-1}(\mathbf{v}) \in \mathbb{Z}_q^m$ is *small* and s.t. $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{v}) = \mathbf{v}$

The genius idea

MUX: $\mu, v_0, v_1 \mapsto v_\mu$

$$\mathbf{V}_0 + \mathbf{C} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0)$$

The genius idea

MUX: $\mu, v_0, v_1 \mapsto v_\mu$

$$\mathbf{V}_0 + \mathbf{C} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0) = \mathbf{V}_\mu + \begin{pmatrix} \mathbf{A} \\ \mathbf{s}\mathbf{A} + \mathbf{e} \end{pmatrix} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0)$$

The genius idea

MUX: $\mu, v_0, v_1 \mapsto v_\mu$

$$\mathbf{V}_0 + \mathbf{C} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0) = \mathbf{V}_\mu + \begin{pmatrix} \mathbf{A} \\ \mathbf{s}\mathbf{A} + \mathbf{e} \end{pmatrix} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0)$$

- $\mathbf{A} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0)$ looks random (LHL) if $\mathbf{G}^{-1}$ has enough min-entropy

The genius idea

$$\text{MUX: } \mu, v_0, v_1 \mapsto v_\mu$$

$$\mathbf{V}_0 + \mathbf{C} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0) = \mathbf{V}_\mu + \begin{pmatrix} \mathbf{A} \\ \mathbf{sA} + \mathbf{e} \end{pmatrix} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0)$$

- $\mathbf{A} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0)$ looks random (LHL) if $\mathbf{G}^{-1}$ has enough min-entropy
- Need a way to make $\mathbf{e} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0)$ independent of $\mathbf{V}_1 - \mathbf{V}_0$

The technical part

$$\mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0) = \mathrm{BitDecomp}(\mathbf{V}_1 - \mathbf{V}_0)$$

$\mathbf{e} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0)$ entirely determined

The technical part

$\mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0)$ Gaussian such that $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0) = \mathbf{V}_1 - \mathbf{V}_0$

$\mathbf{e} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0)$ Gaussian in a coset of $\mathbf{e} \cdot \Lambda^{\perp}(\mathbf{G})$

The technical part

$\mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0)$ Gaussian such that $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0) = \mathbf{V}_1 - \mathbf{V}_0$

$\mathbf{e} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0)$ Gaussian in a coset of $\mathbf{e} \cdot \Lambda^{\perp}(\mathbf{G})$

Coset still depends on $\mathbf{V}_1 - \mathbf{V}_0$

The technical part

$\mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0)$ Gaussian, $\mathbf{z}$ Gaussian

$\mathbf{e} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0) + \mathbf{z}$ ensures the domain is $\mathbb{Z}$

The technical part

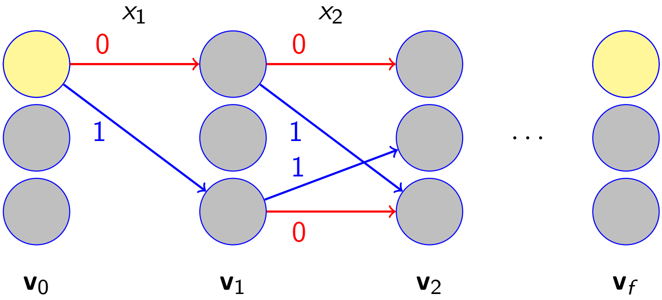
$\mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0)$ Gaussian, $\mathbf{z}$ Gaussian

$\mathbf{e} \cdot \mathbf{G}^{-1}(\mathbf{V}_1 - \mathbf{V}_0) + \mathbf{z}$ ensures the domain is $\mathbb{Z}$

Now only depends on $\|\mathbf{e}\|$

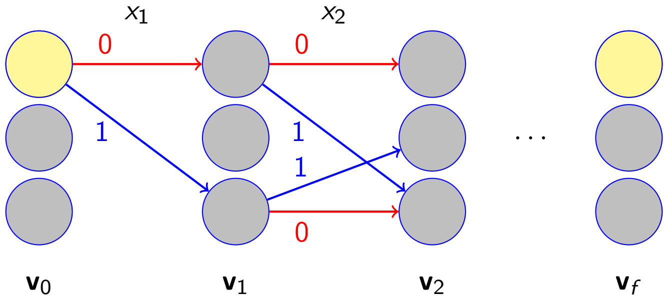
The final result

Branching Program



The final result

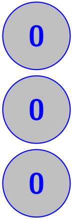
Branching Program



$$\mathbf{v}_{t-1}[i] = \mathrm{MUX}(x_t, \mathbf{v}_{t-1}[j], \mathbf{v}_{t-1}[k])$$

The final result

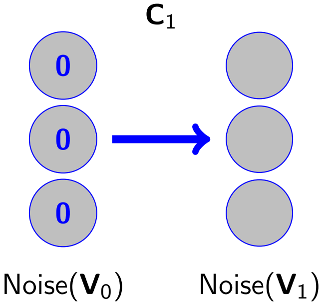
Branching Program



Noise($\mathbf{V}_0$)

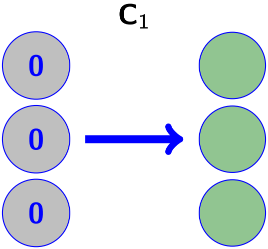
The final result

Branching Program



The final result

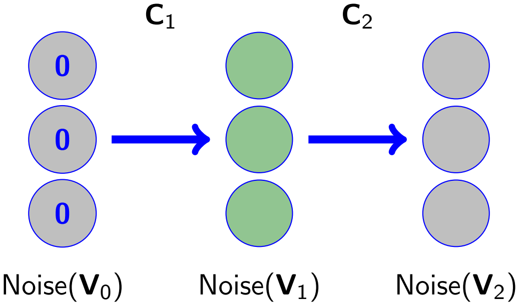
Branching Program



Noise($\mathbf{V}_0$) Noise($\mathbf{V}_1$)

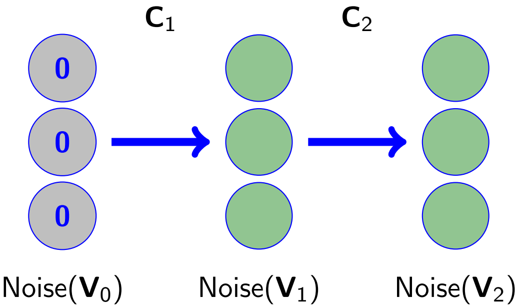
The final result

Branching Program



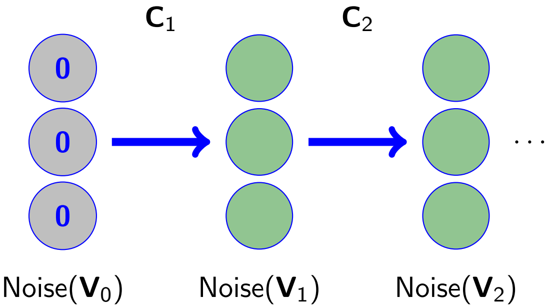
The final result

Branching Program



The final result

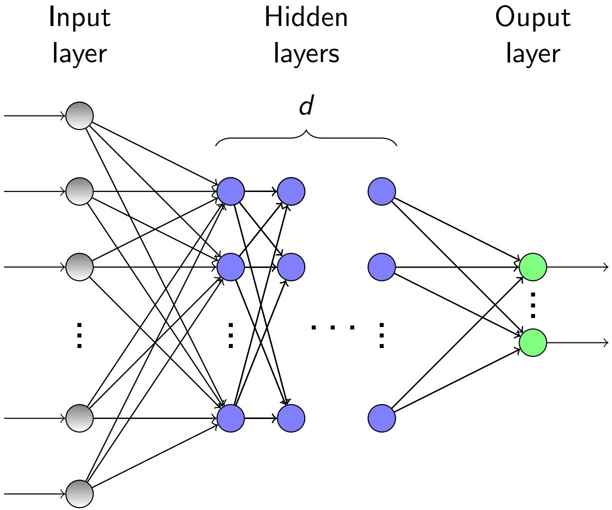
Branching Program



Bourse, Minelli, Minihold, Paillier

Fast Homomorphic Evaluation of Deep Discretized Neural Networks

The problem



The problem

Previous approaches:

The problem

Previous approaches:

- use Somewhat Homomorphic Encryption

The problem

Previous approaches:

- use Somewhat Homomorphic Encryption
- low degree activation function

The problem

Previous approaches:

- use Somewhat Homomorphic Encryption
- low degree activation function
- doesn't scale very well for deep networks :(

The problem

Previous approaches:

- use Somewhat Homomorphic Encryption
- low degree activation function
- doesn't scale very well for deep networks :(

Our approach:

The problem

Previous approaches:

- use Somewhat Homomorphic Encryption
- low degree activation function
- doesn't scale very well for deep networks :(

Our approach:

- use efficient bootstrapping techniques

The problem

Previous approaches:

- use Somewhat Homomorphic Encryption
- low degree activation function
- doesn't scale very well for deep networks :(

Our approach:

- use efficient bootstrapping techniques
- sign activation function

The problem

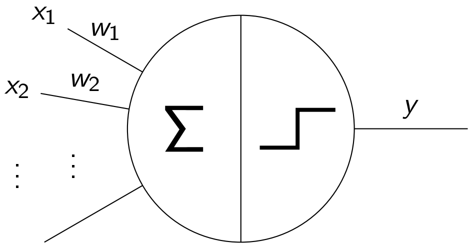
Previous approaches:

- use Somewhat Homomorphic Encryption
- low degree activation function
- doesn't scale very well for deep networks :(

Our approach:

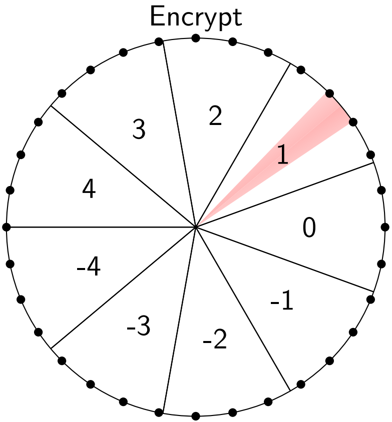
- use efficient bootstrapping techniques
- sign activation function
- evaluation linear in number of neuron :)

The problem



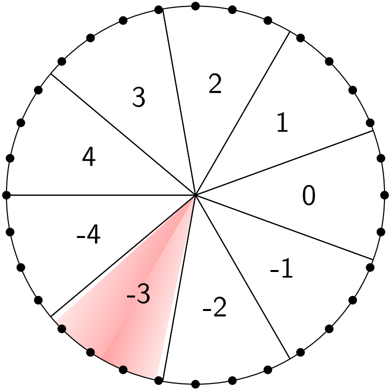
$$y = \text{sign}\left(\sum_i w_i x_i\right)$$

Per neuron evaluation

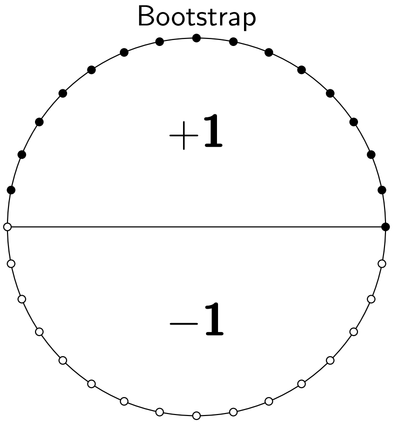


Per neuron evaluation

Homomorphic addition



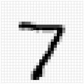
Per neuron evaluation



The big picture

Evaluation of a 784:30:10 neural network for handwritten digit recognition

User



$\xrightarrow{\mathrm{Enc}(\sum_i p_i X^i)}$ 1 RLWE $\xrightarrow{\cdot \sum_i w_i X^{-i}}$ 30 RLWE

Server

30 RLWE

$\downarrow$ Extract

30 N-LWE

$\downarrow$ Key Switching

30 n-LWE

$\downarrow$ Sign Bootstrapping

30 N-LWE

$\downarrow$ weighted sums

10 N-LWE

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

The results

MNIST dataset, classification of handwritten digits:

	Neurons	ct size	Accuracy	Time enc.	Time eval	Time dec.
Cryptonets	945	586 MB	98.95%	122 s	570 s	5 s
Cryptonets*	945	73.3 kB	98.95%	0.015 s	0.07 s	0.0006 s
FHE-DiNN30	30	≈8.2 kB	93.71%	0.168 ms	0.49 s	0.0106 ms
FHE-DiNN30	100	≈8.2 kB	96.35%	0.168 ms	1.65 s	0.0106 ms