# Oblivious Neural Network

**Stanisław Barański**
**stanislaw.baranski@pg.edu.pl**
**https://stan.bar**

**08.06.2022**

# Agenda

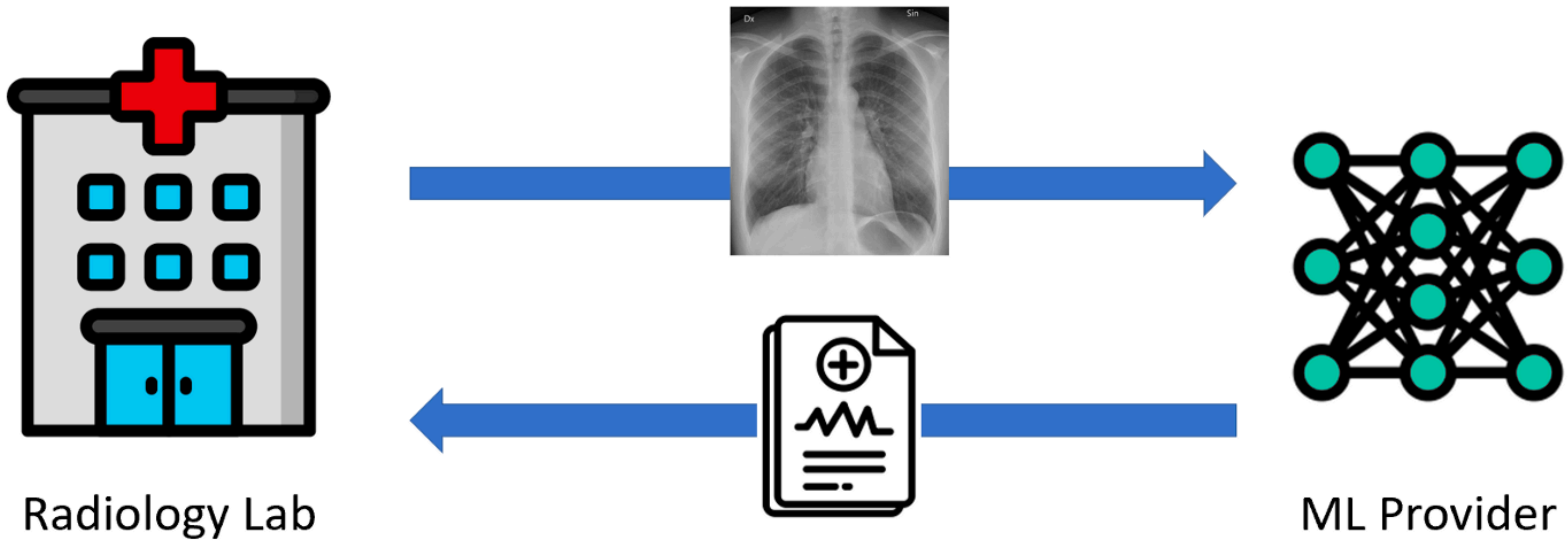
- Use case
- Multi-party computation (MPC)
- Neural Network (NN)
- Oblivious Neural Network (ONN) = NN + MPC
- State-of-the-art
- My work

# Use case

Sławomir Barański •••

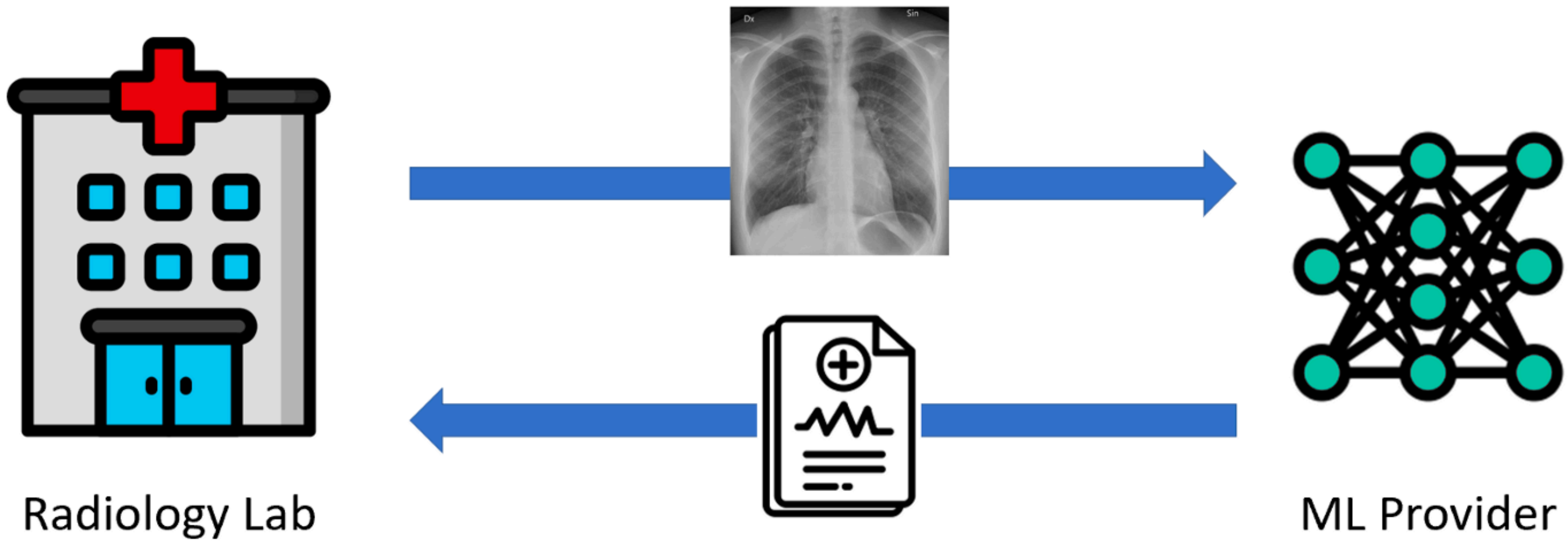
# Use case

## Medical prognosis using machine learning



# Use case

## Medical prognosis using machine learning

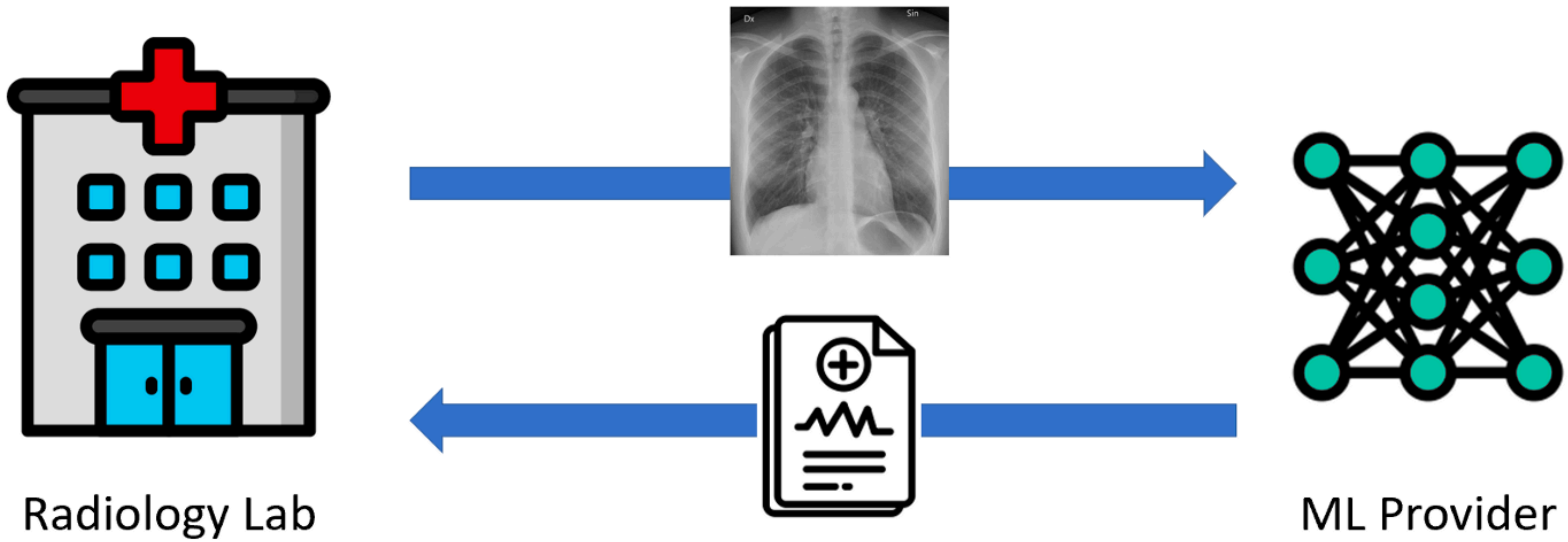



Problems:

- The ML Provider cannot share the model as it may be proprietary
- The laboratory can not send input data to the ML Provide because of legal prohibitions or complex agreements.
- Privacy risks

# Use case

## Medical prognosis using machine learning



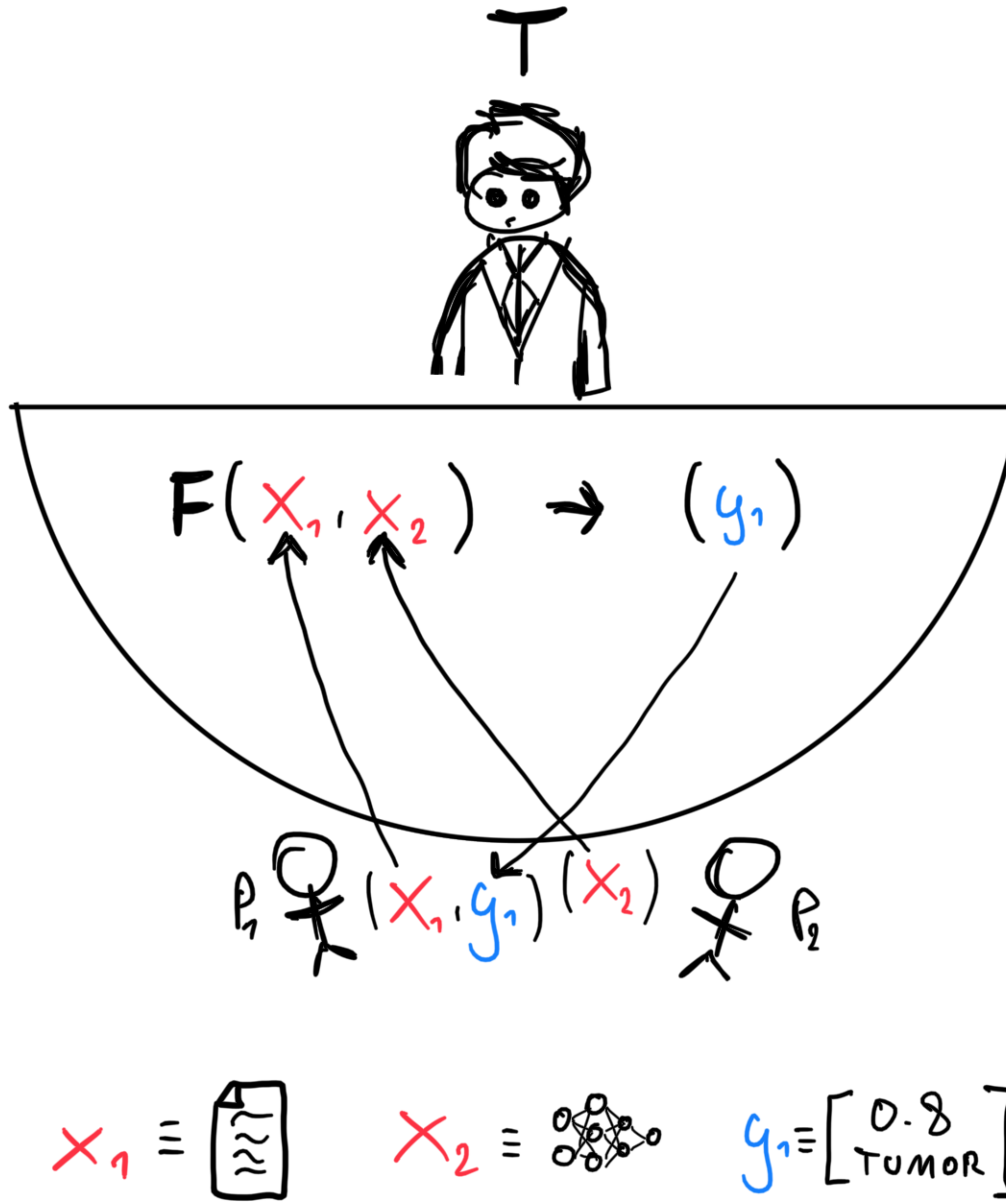

**Is it possible to do this computation without the radiology lab ever sharing the patient's sensitive data and the ML provider sharing its proprietary model?**

# Problem statement

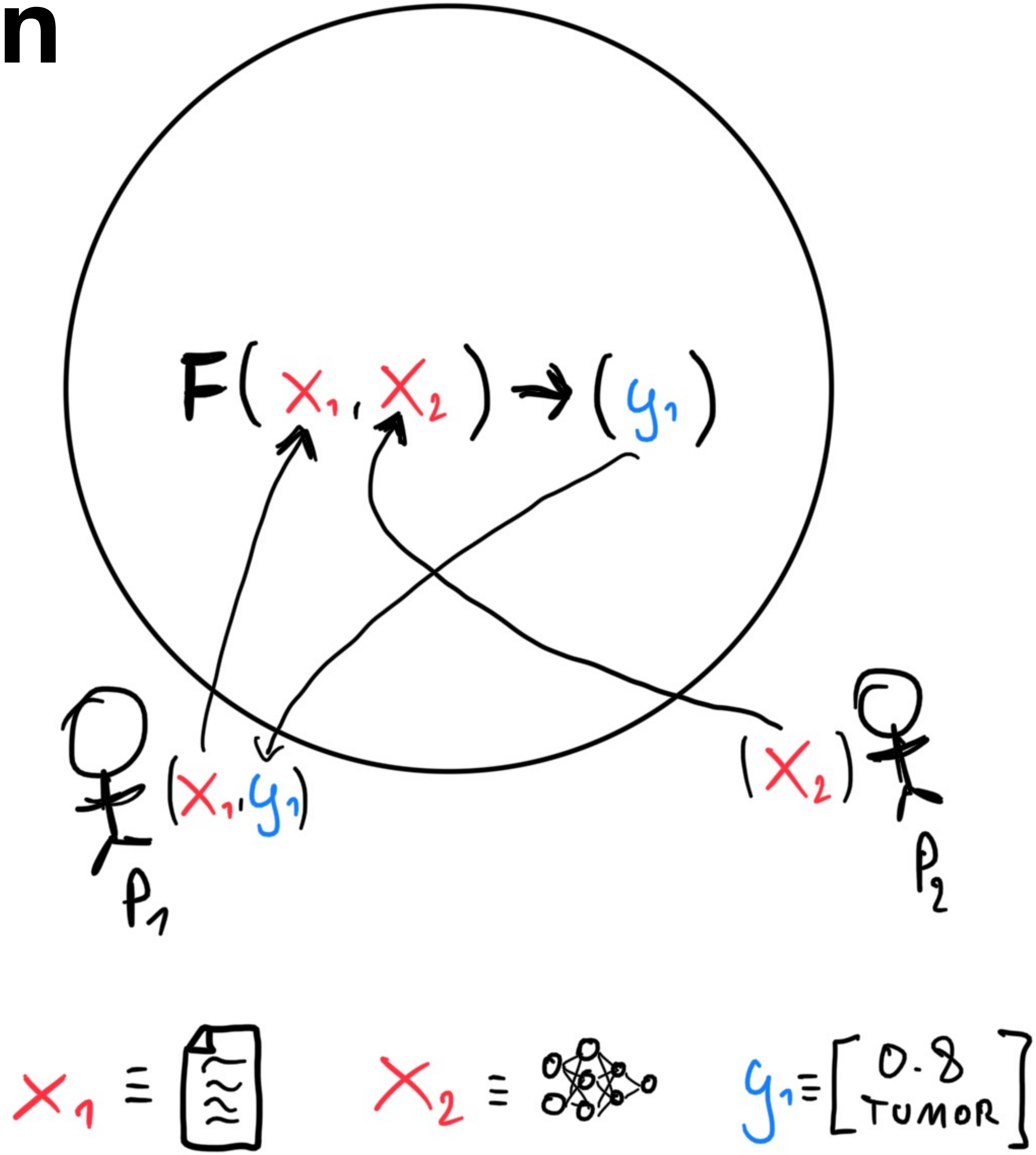
## Medical prognosis using machine learning

- **S**erver learns nothing about **C**lient's input;
- **C**lient learns nothing about the **S**erver’s model;
- Yet the predictions are correct.

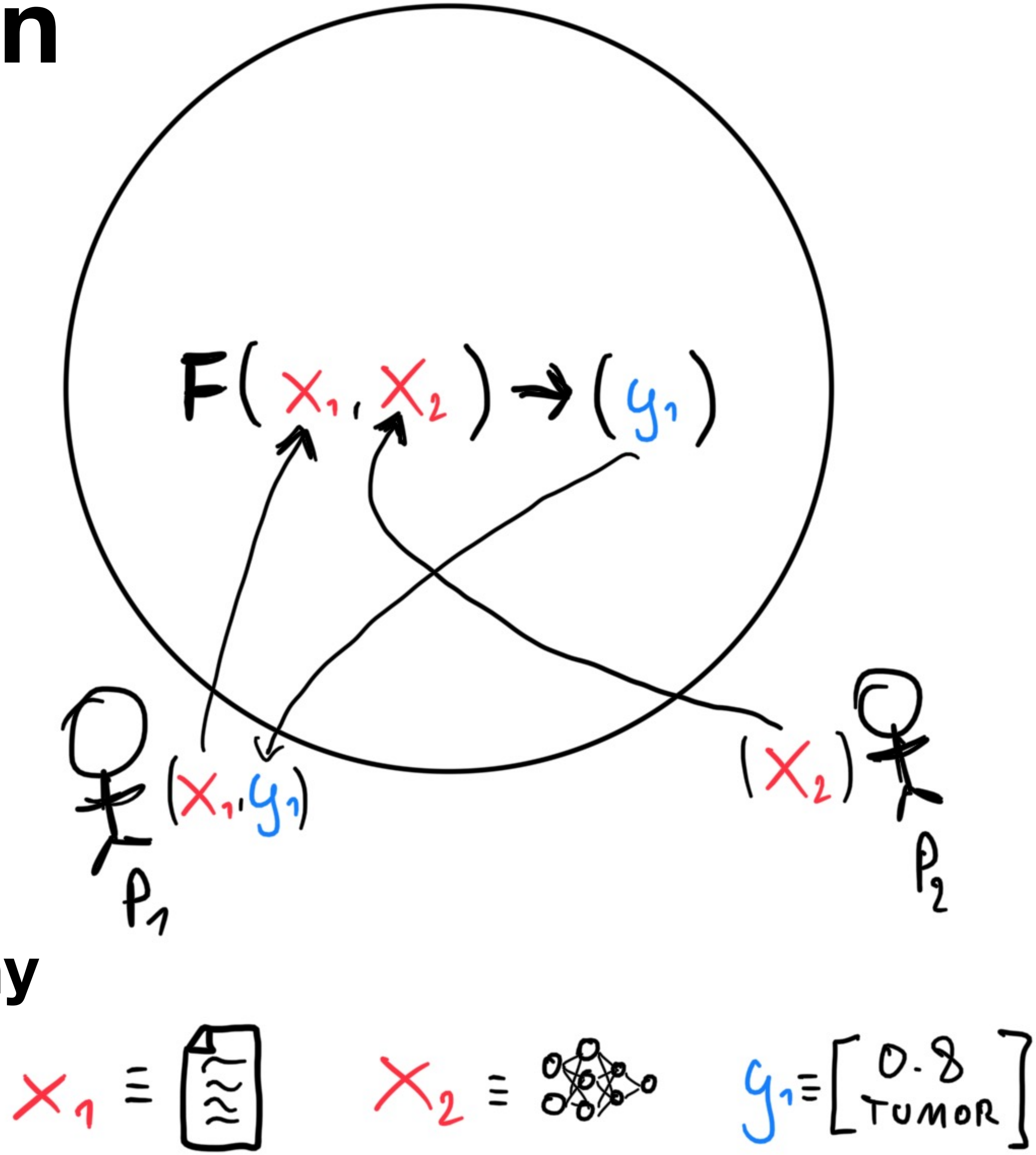
# Ideal world



# Ideal solution



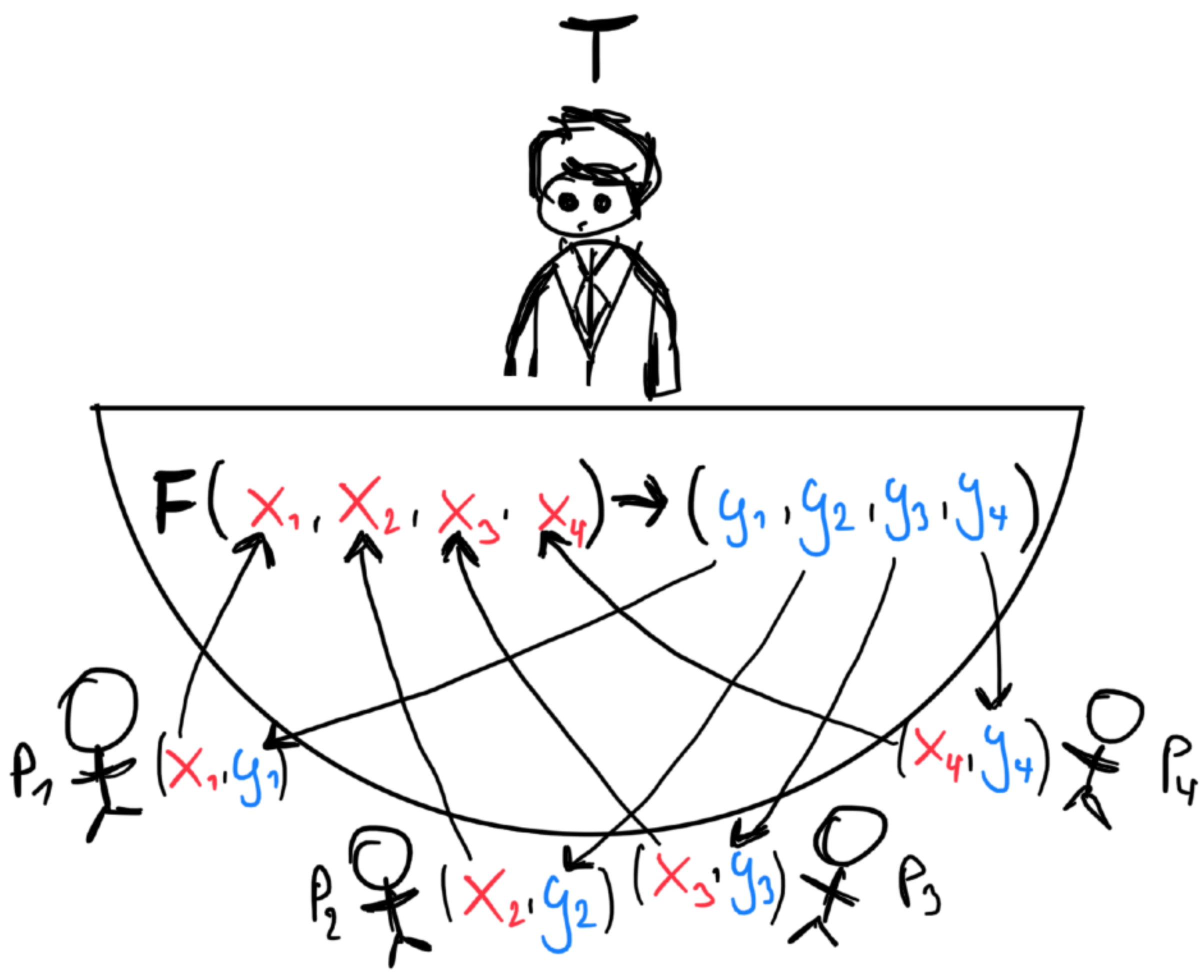
# Ideal solution

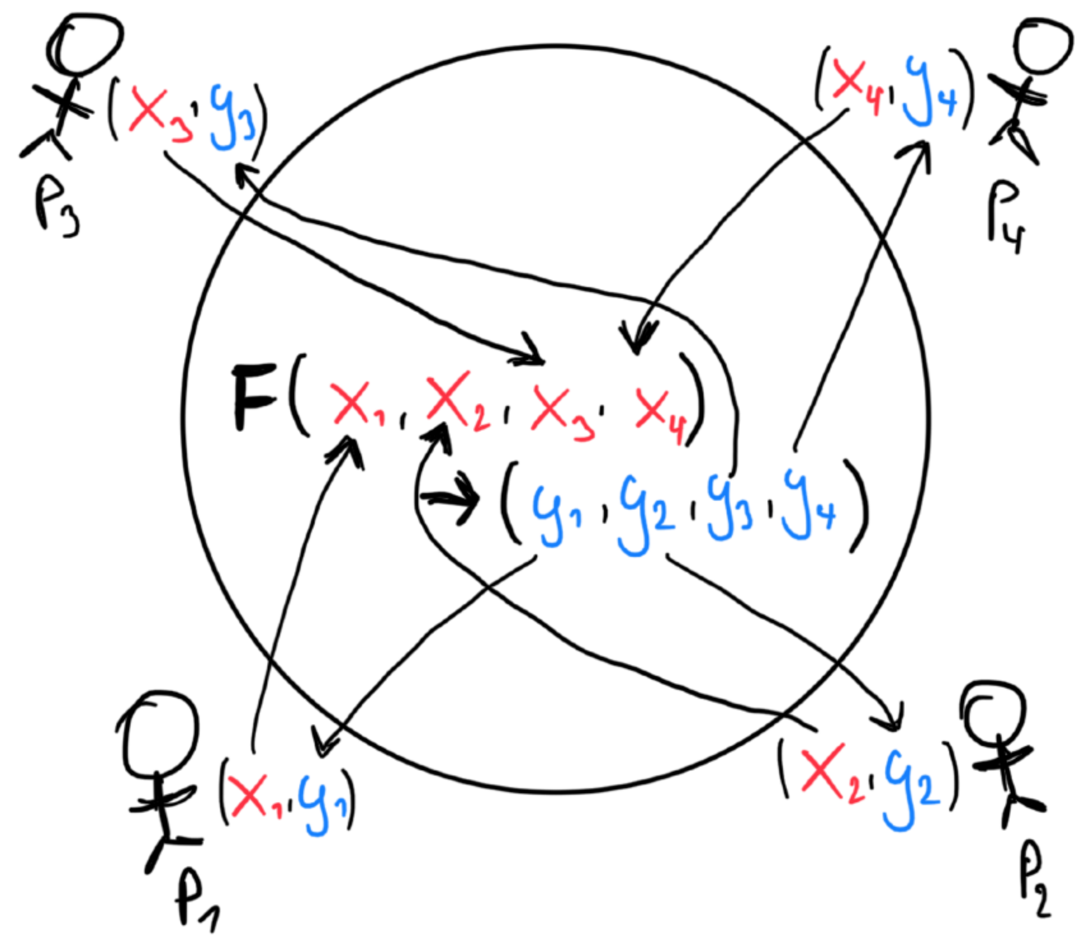



**Magic?**
**No, just cryptography**

# Multi-party computation (MPC)

- Multi-party computation (MPC) enables a group of independent parties who do not trust each other to jointly compute a function $f(x_1, x_2 \ldots x_n)$ where $x_i$ is the private input for *i*-th party.



# MPC Applications

## Yao's Millionaires Problem

"Two millionaires wish to know who is richer without revealing their actual wealth."

So the goal is to compute $x_1 \leq x_2$ where $x_1$ is the first party's *private* input and $x_2$ is the second party's *private* input.

$$f(x_1, x_2) = x_1 \leq x_2$$

# MPC Intuition

## Function as a lookup table

- P_x (server) represents a function $f(x_1, x_2 \ldots x_n)$ as a lookup table.

| x | y | f(x,y) = x > y |
|---|---|---|
| 0 | 0 | False |
| 0 | 1 | False |
| 0 | 2 | False |
| 0 | 3 | False |
| 1 | 0 | True |
| 1 | 1 | False |
| 1 | 2 | False |
| 1 | 3 | False |
| 2 | 0 | True |
| 2 | 1 | True |
| 2 | 2 | False |
| 2 | 3 | False |
| 3 | 0 | True |
| 3 | 1 | True |
| 3 | 2 | True |
| 3 | 3 | False |

# MPC Intuition

## Function as a lookup table

- P_x (server) represents a function $f(x_1, x_2 \ldots x_n)$ as a lookup table.
- P_x encrypts the table using randomly selected keys for each value of x and y.

| x | y | f(x,y) = x > y |
|---|---|---|
| 0 | 0 | $E_{k_x^0,k_y^0}(False)$ |
| 0 | 1 | $E_{k_x^0,k_y^1}(False)$ |
| 0 | 2 | $E_{k_x^0,k_y^2}(False)$ |
| 0 | 3 | $E_{k_x^0,k_y^3}(False)$ |
| 1 | 0 | $E_{k_x^1,k_y^0}(True)$ |
| 1 | 1 | $E_{k_x^1,k_y^1}(False)$ |
| 1 | 2 | $E_{k_x^1,k_y^2}(False)$ |
| 1 | 3 | $E_{k_x^1,k_y^3}(False)$ |
| 2 | 0 | $E_{k_x^2,k_y^0}(True)$ |
| 2 | 1 | $E_{k_x^2,k_y^1}(True)$ |
| 2 | 2 | $E_{k_x^2,k_y^2}(False)$ |
| 2 | 3 | $E_{k_x^2,k_y^3}(False)$ |
| 3 | 0 | $E_{k_x^3,k_y^0}(True)$ |
| 3 | 1 | $E_{k_x^3,k_y^1}(True)$ |
| 3 | 2 | $E_{k_x^3,k_y^2}(True)$ |
| 3 | 3 | $E_{k_x^3,k_y^3}(False)$ |

# MPC Intuition

## Function as a lookup table

- P_x (server) represents a function $f(x_1, x_2 \ldots x_n)$ as a lookup table.
- P_x encrypts the table using randomly selected keys for each value of x and y.
- P_x randomly permute the table and send it to the other party P_y (client).
- The goal is to let the other party encrypt only the f(x,y) corresponding to the selected values x and y.
  - P_x sends his k_x to P_y.
  - P_x offers P_y to pick **one** value out of |Y| using Oblivious Transfer (OT)
  - P_y, having both keys k_x and k_y can decrypt the value f(x,y), without learning anything about x.

| x | y | f(x,y) = x > y |
|---|---|---|
| 3 | 1 | $E_{k_x^3,k_y^1}(True)$ |
| 1 | 1 | $E_{k_x^1,k_y^1}(False)$ |
| 0 | 1 | $E_{k_x^0,k_y^1}(False)$ |
| 0 | 3 | $E_{k_x^0,k_y^3}(False)$ |
| 1 | 2 | $E_{k_x^1,k_y^2}(False)$ |
| 1 | 0 | $E_{k_x^1,k_y^0}(True)$ |
| 3 | 2 | $E_{k_x^3,k_y^2}(True)$ |
| 1 | 3 | $E_{k_x^1,k_y^3}(False)$ |
| 2 | 0 | $E_{k_x^2,k_y^0}(True)$ |
| 2 | 2 | $E_{k_x^2,k_y^2}(False)$ |
| 0 | 2 | $E_{k_x^0,k_y^2}(False)$ |
| 2 | 3 | $E_{k_x^2,k_y^3}(False)$ |
| 3 | 0 | $E_{k_x^3,k_y^0}(True)$ |
| 3 | 3 | $E_{k_x^3,k_y^3}(False)$ |
| 0 | 0 | $E_{k_x^0,k_y^0}(False)$ |
| 2 | 1 | $E_{k_x^2,k_y^1}(True)$ |

# Function as a lookup problem

- The size of the table is $|X| \times |Y|$
- For two uint32's the size of the table is $|\text{uint32}| \times |\text{uint32}| = 2^{32} \times 2^{32} = 2^{64}$
- Each value encoded on 4 bytes
- $2^{64} \times 4\text{bytes} = \text{too large}$

# Function as a lookup problem

- The size of the table is $|X| \times |Y|$
- For two uint32's the size of the table is $|\text{uint32}| \times |\text{uint32}| = 2^{32} \times 2^{32} = 2^{64}$
- Each value encoded on 4 bytes
- $2^{64} \times 4\text{bytes} = \text{too large}$

**But, for a small domains it works fine**

# MPC practical solution

## Yao’s Garbled Circuit

- Logic gates have a domain size of $4 = |\{0,1\}| \times |\{0,1\}|$.
- We can apply the lookup table technique to a sequence of logic gates, without blowing out the table size.

Thanks to Church Theorem we know that every algorithm can be represented as

- Turing machine
- RAM machine
- Logic gate network

As a result we can represent any algorithm via logic gate network.

Then by representing them as an encrypted lookup table compute MPC, and we get Yao’s Garbled Circuit (GC) technique.

# MPC practical solution

## Yao's Garbled Circuit

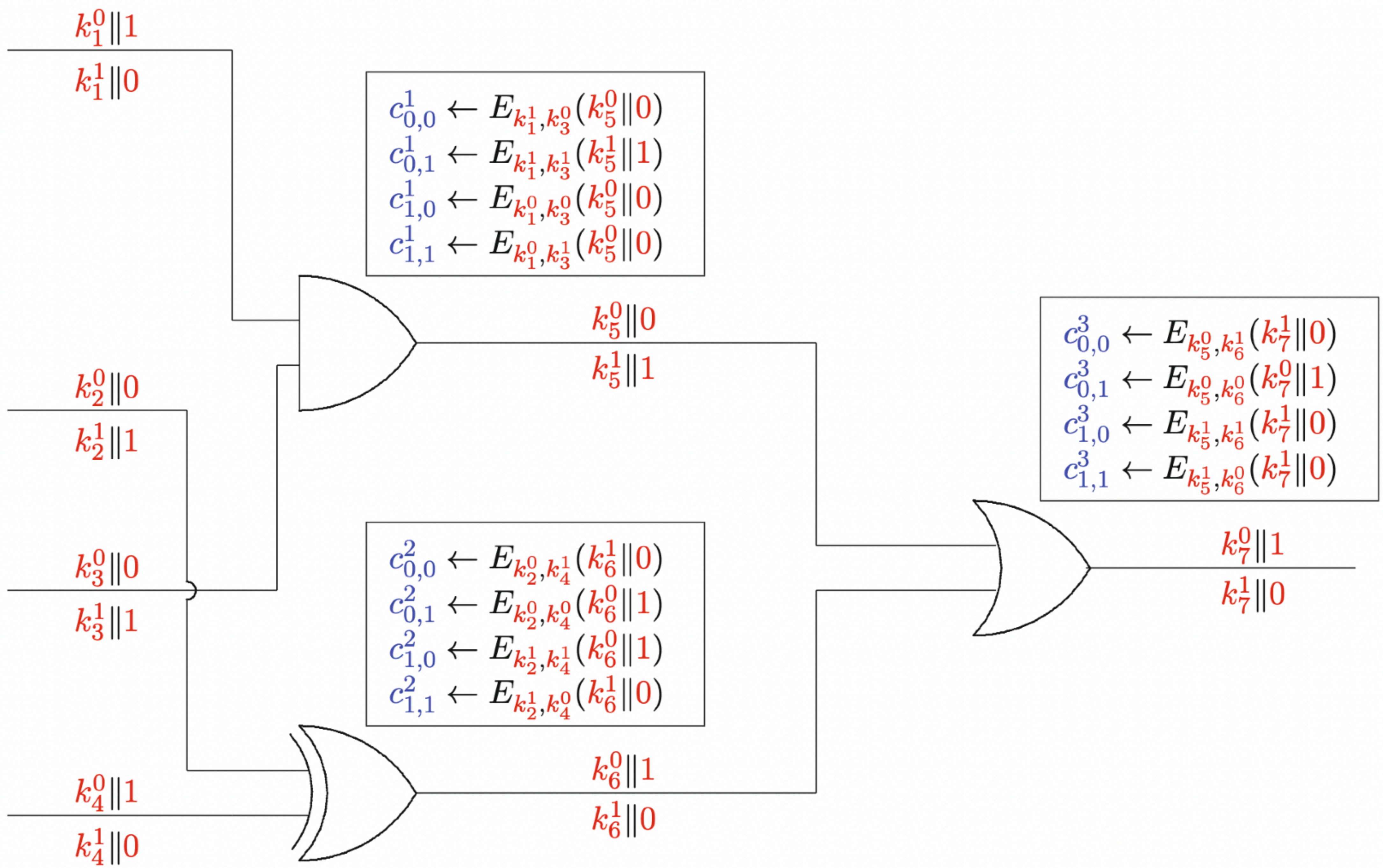
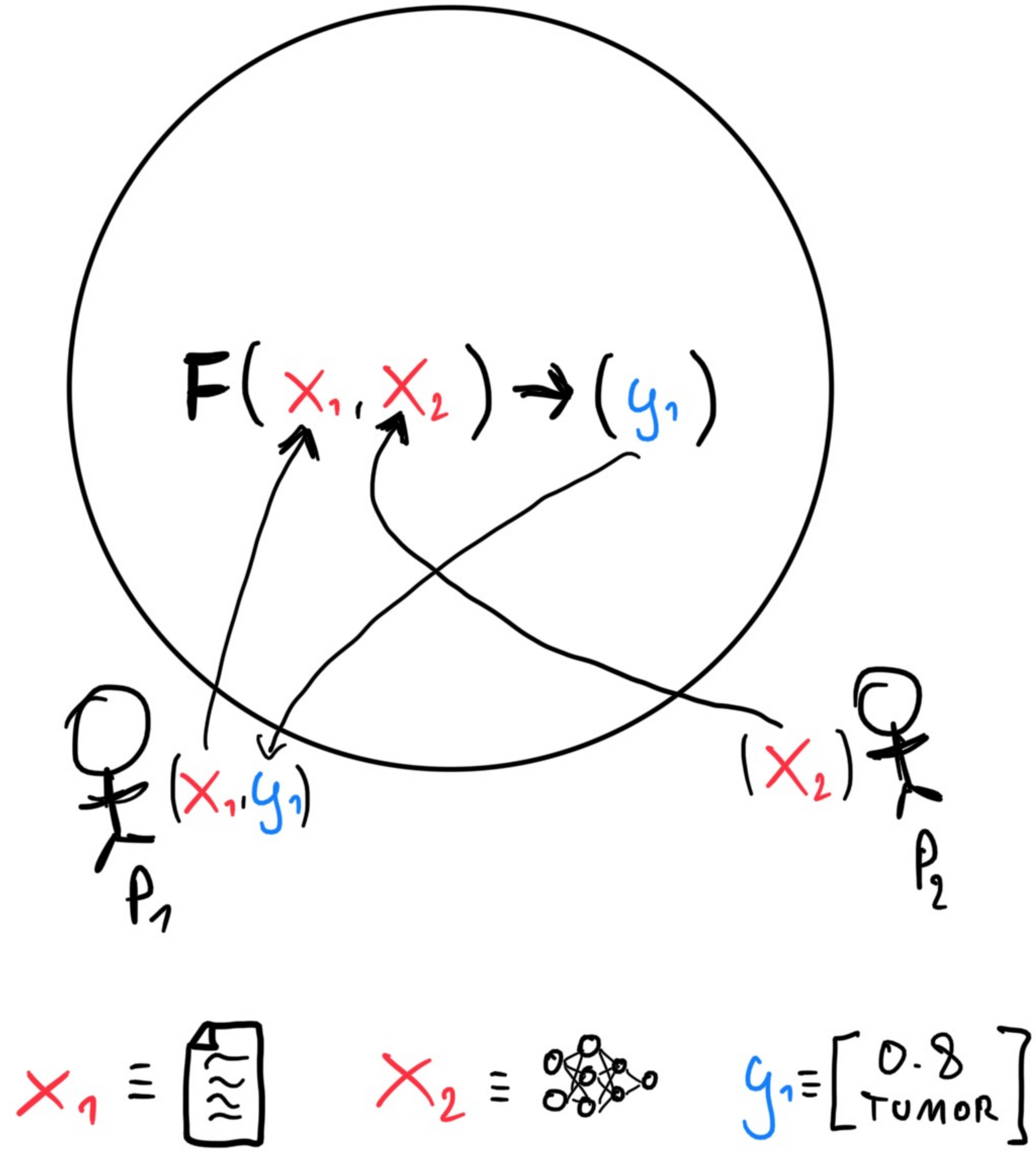



FIGURE 22.2. A garbled circuit

# Obvious Neural Network

## What is the function f(x,y) ?



# What is the function f(x,y) ?

## What is neural network

- A neural network consist of a pipeline of layers. Each layer receives and input vector, process it to produce an output that serves as input to the next layer. The first layer is an input, the last layer outputs the final prediction.

- A typical neural network process input data in groups of layers, by first applying linear transformations, followed by the application of non-linear activation function.

- input -> linear transformations -> non-linear activation function -> ... -> output

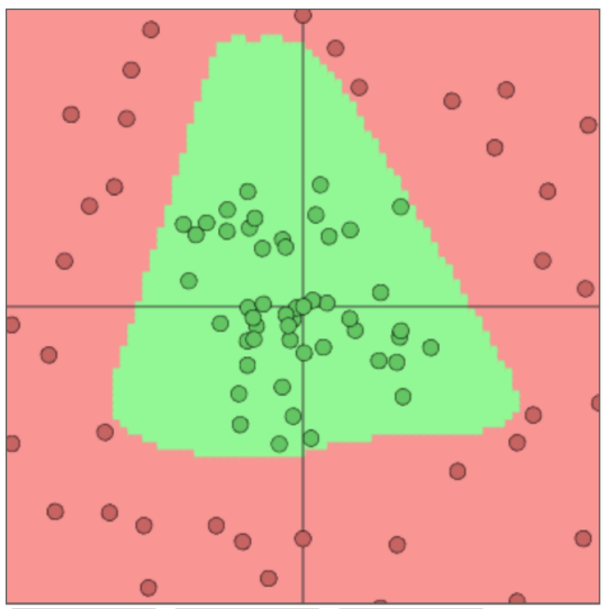
# What is neural network

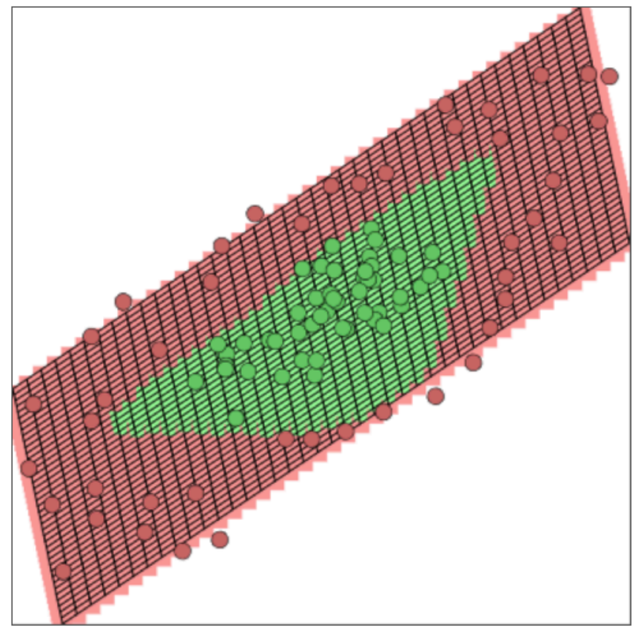
## Linear and non-linear transformations

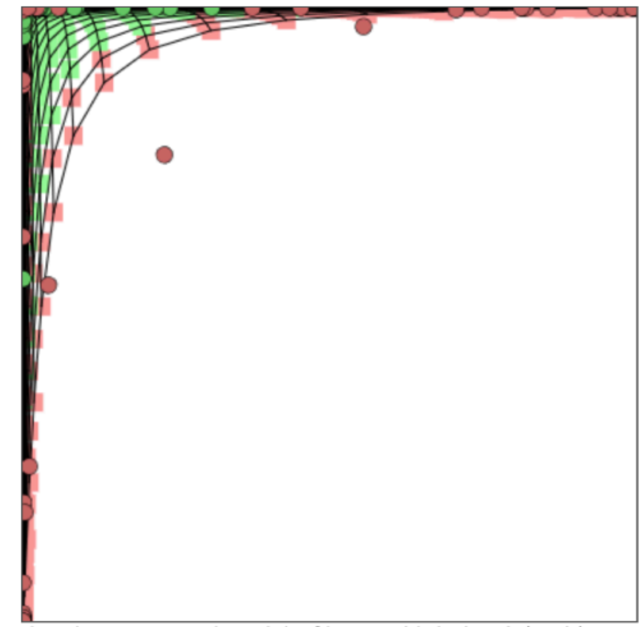
- The commonest linear transformation in NN is matrix multiplications and additions.
- $$y := W \cdot x + b$$
- The commonest non-linear transformation in NN is logistic function (sigmoid), Tanh, ReLU, softmax …

# What is neural network

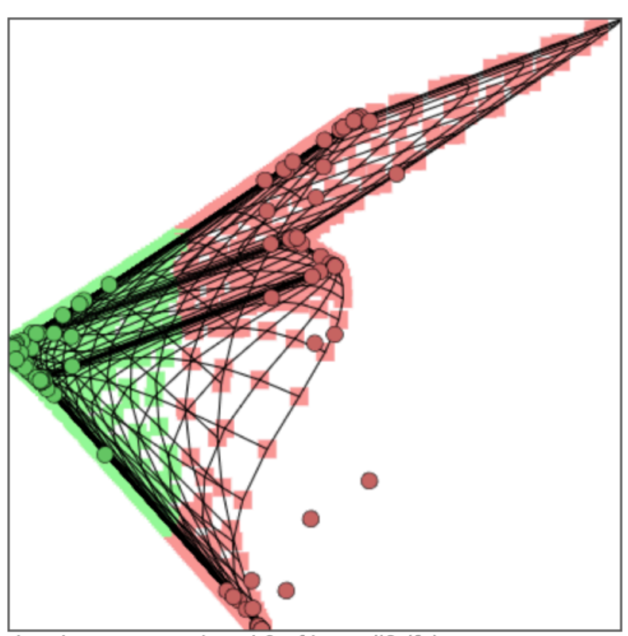
## Linear and non-linear transformations

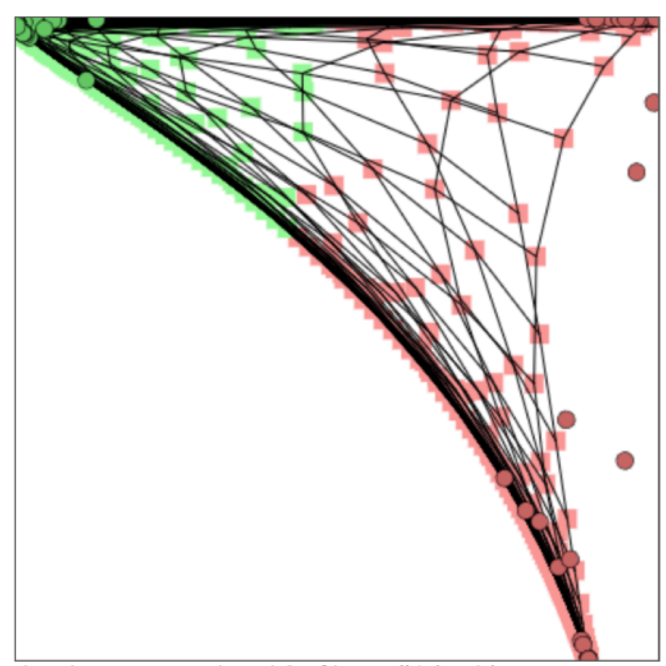


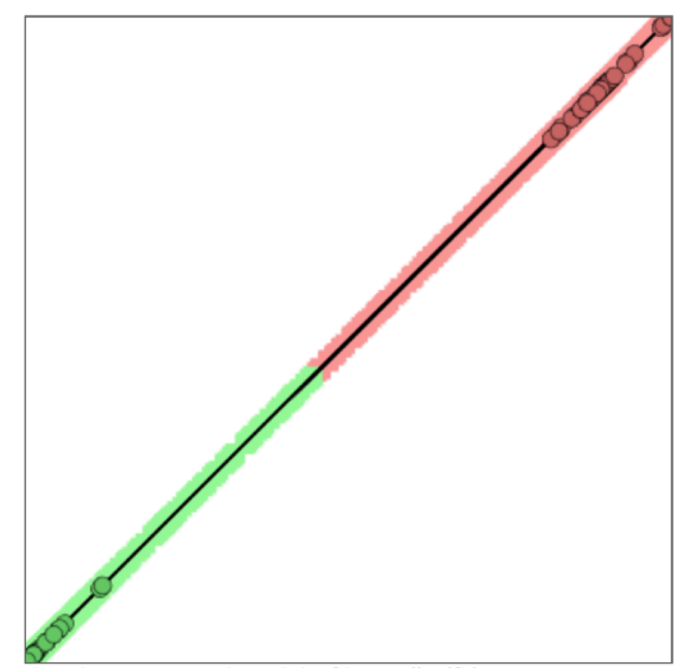




$y := W \cdot x + b$ $\quad$ $y := \tanh(x)$







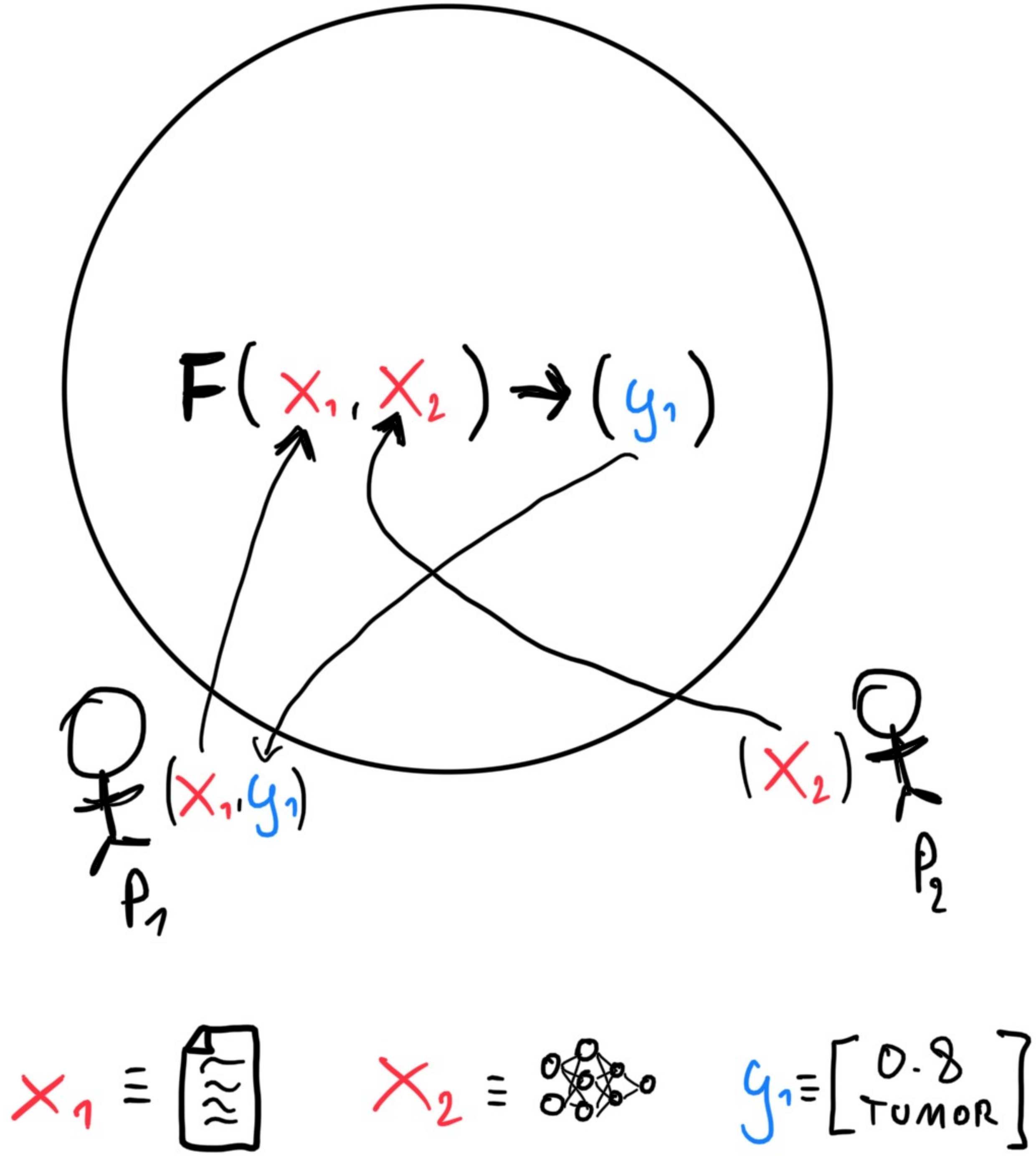
$y := W \cdot x + b$ $\quad$ $y := \tanh(x)$ $\quad$ $z := W \cdot x + b$

## Neural network prediction is all about matrix multiplications and additions

- As a result, any neural network can be represented as a model
- $$z := (W_L * f_{L-1}(\ldots f_1(W_1 * X + B_1)\ldots) + b_L)$$

# What is the function f(x,y) ?

- $f(X,(W_1,\ldots,W_L,B_1,\ldots,b_L)) := W_L \cdot f_{L-1}(\ldots f_1(W_1 \cdot X + B_1)\ldots) + b_L$



# Oblivious Neural Network

$$z = \boxed{W' \cdot \boxed{f(\boxed{W \cdot x + b})} + b'}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad W = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \quad W' = \begin{bmatrix} w'_{1,1} & w'_{1,2} \\ w'_{2,1} & w'_{2,2} \end{bmatrix}, \quad b' = \begin{bmatrix} b'_1 \\ b'_2 \end{bmatrix}$$

$$x \to \boxed{W \cdot \square + b} \xrightarrow{y} \boxed{f(\square)} \xrightarrow{x'} \boxed{W' \cdot \square + b'} \to z$$

All operations are in a finite field $\mathbb{Z}_N$

# Oblivious Linear transformation

$$\boxed{W \cdot \square + b}$$

$$\begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} =$$

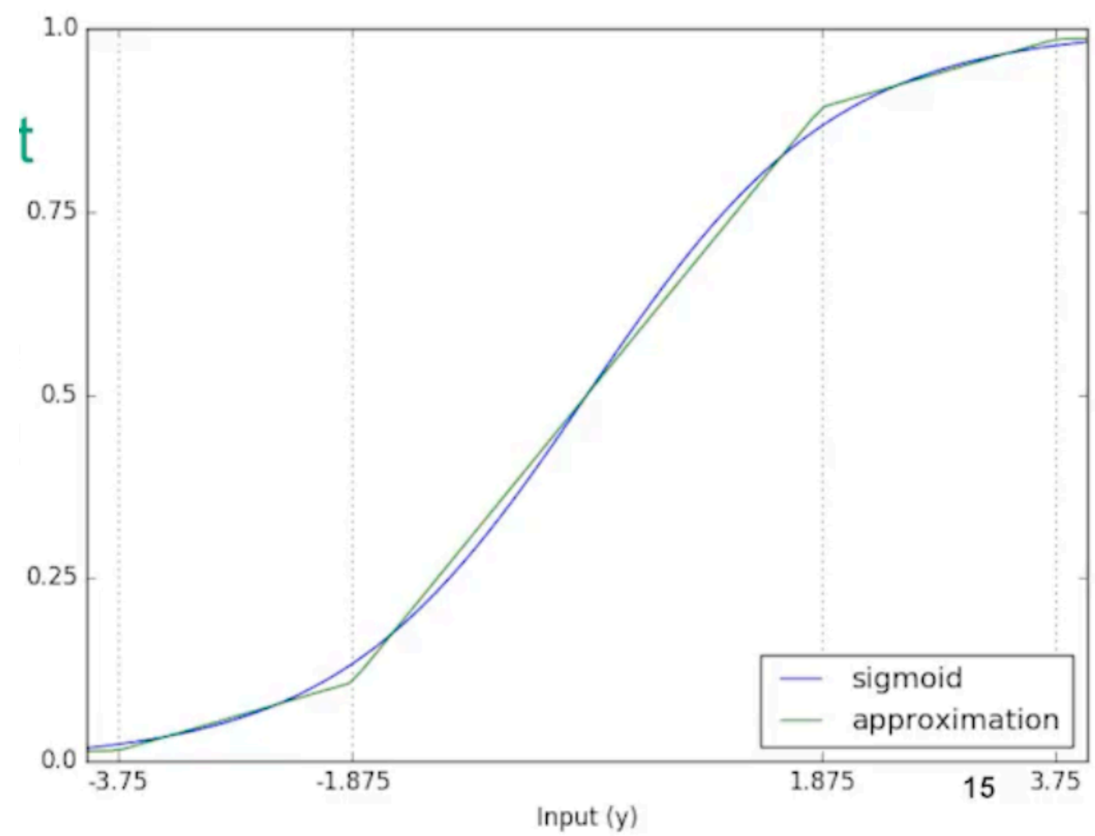
$$= \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix} \cdot \begin{bmatrix} x_1^{(s)} + x_1^{(c)} \\ x_2^{(s)} + x_2^{(c)} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} =$$

$$= \begin{bmatrix} w_{1,1}\left(x_1^{(s)} + x_1^{(c)}\right) + w_{1,2}\left(x_2^{(s)} + x_2^{(c)}\right) + b_1 \\ w_{2,1}\left(x_1^{(s)} + x_1^{(c)}\right) + w_{2,2}\left(x_2^{(s)} + x_2^{(c)}\right) + b_2 \end{bmatrix} =$$

$$\begin{bmatrix} w_{1,1} x_1^{(s)} + w_{1,2} x_2^{(s)} + b_1 + w_{1,1} x_1^{(c)} + w_{1,2} x_2^{(c)} \\ w_{2,1} x^{(s)} + w_{2,2} x_2^{(s)} + b_1 + w_{2,1} x_1^{(c)} + w_{2,2} x_2^{(c)} \end{bmatrix}$$

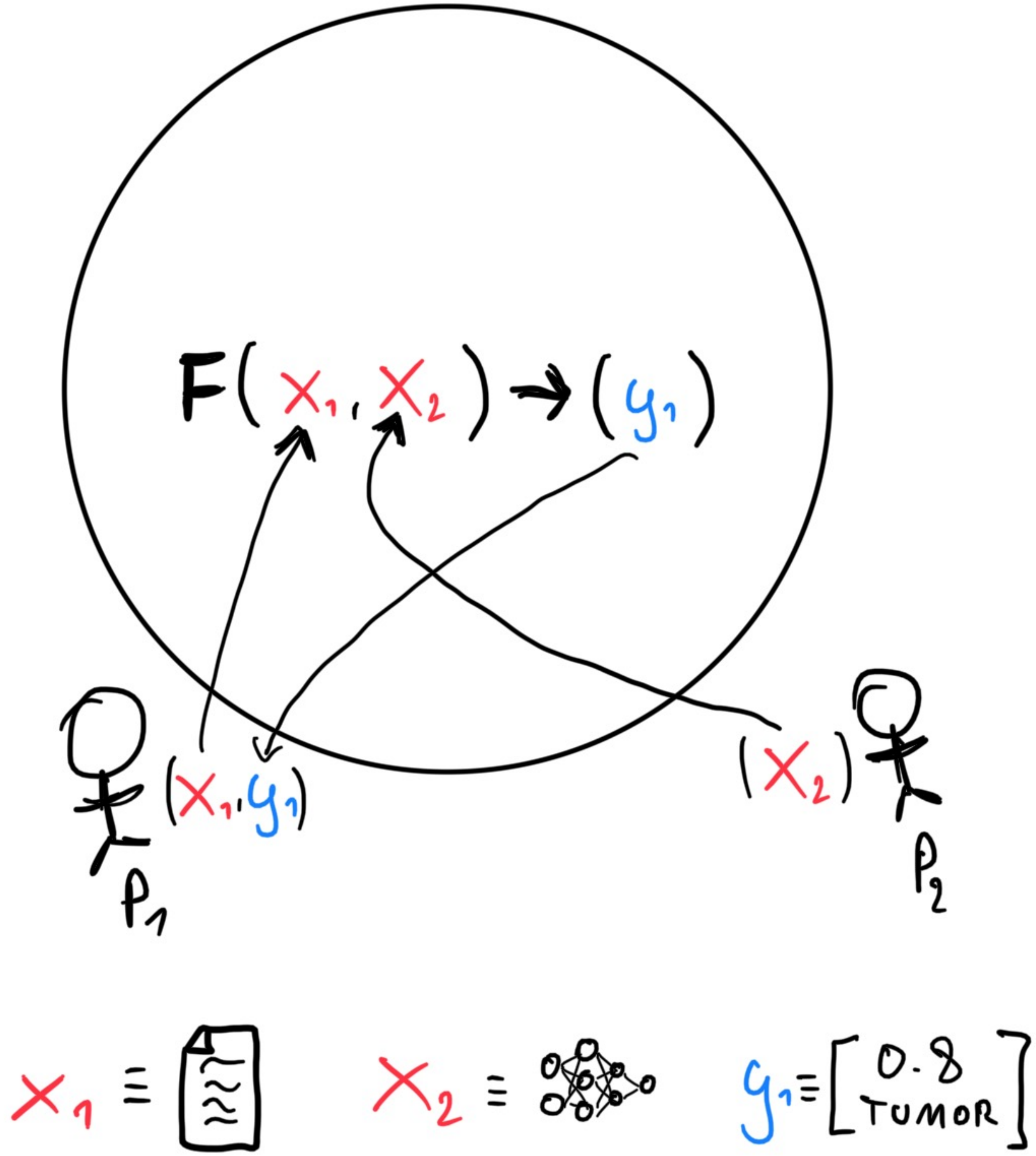
# Oblivious activation function

- Piecewise linear functions like LeRU = max(0,y) are easily implemented using GC.
- Smooth functions like sigmoid $f(y) = \dfrac{1}{1+e^{-y}}$ are hard to implement using MPC because both division and exponentiation are computationally expensive, therefore approximation is used.
- About 14 segments are enough.



# What is the function f(x,y) ?

- $f(X,(W_1,\ldots,W_L,B_1,\ldots,b_L)) := W_L \cdot f_{L-1}(\ldots f_1(W_1 \cdot X + B_1)\ldots) + b_L$



# Oblivious Neural Network

## State-of-the-art

- CryptoNets — CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy (Microsoft Research)(2016)
- SecureML — SecureML: A System for Scalable Privacy-Preserving Machine Learning (Visa Research, University of Maryland) (2017)
- MiniONN — Oblivious Neural Network Predictions via MiniONN Transformations (Aalto University) (2017)
- Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications (UC San Diego, TU Darmstadt) (2018)
- GAZELLE — GAZELLE: A Low Latency Framework for Secure Neural Network Inference (MIT) (2018)
- XONN — XONN: XNOR-based Oblivious Deep Neural Network Inference (Microsoft Research) (2019)
- EzPC — EzPC: Programmable, Efficient, and Scalable Secure Two-Party Computation for Machine Learning (Microsoft Research)(2019)
- Delphi — Delphi: A Cryptographic Inference Service for Neural Networks (UC Berkeley) (2020)
- CrypTFlow — CRYPTFLOW: Secure TensorFlow Inference (Microsoft) (2020)

# Oblivious Neural Network

## State-of-the-art

**Table 3: Comparison of secure deep learning frameworks, their characteristics, and performance results for classifying one image from the MNIST dataset in the LAN setting.**

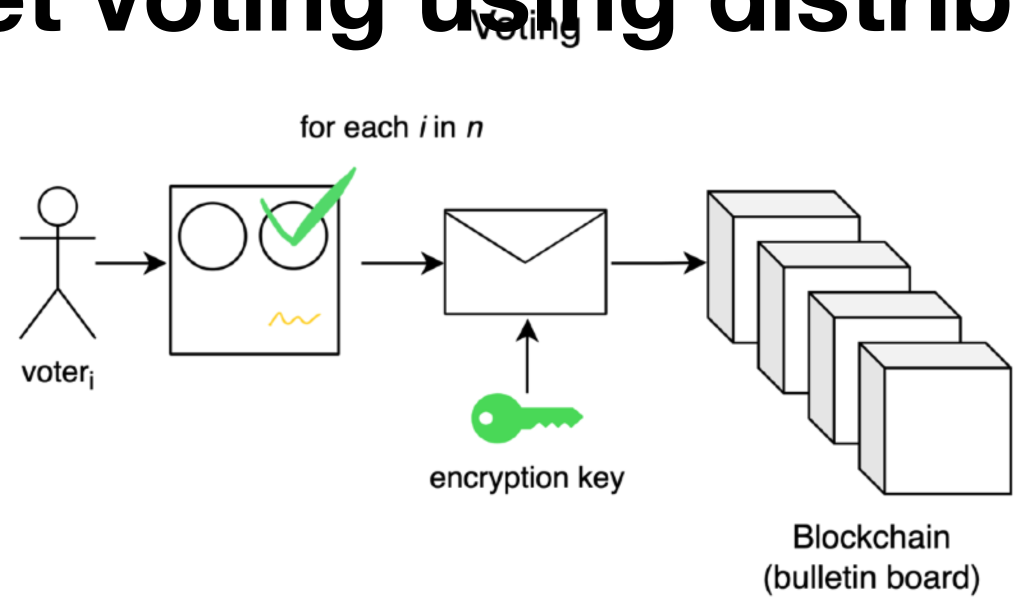
| Framework | Methodology | Non-linear Activation and Pooling Functions | Classification Timing (s) | | | Communication (MB) | | | Classification Accuracy |
|---|---|---|---|---|---|---|---|---|---|
| | | | Offline | Online | Total | Offline | Online | Total | |
| **Microsoft CryptoNets** [36] | Leveled HE | ✗ | - | - | 297.5 | - | - | 372.2 | 98.95 % |
| **DeepSecure** [77] | GC | ✓ | - | - | 9.67 | - | - | 791 | 99 % |
| **SecureML** [67] | Linearly HE, GC, SS | ✗ | 4.70 | 0.18 | 4.88 | - | - | - | 93.1 % |
| **MiniONN** (Sqr Act.) [62] | Additively HE, GC, SS | ✗ | 0.90 | 0.14 | 1.04 | 3.8 | 12 | 15.8 | 97.6 % |
| **MiniONN** (ReLu + Pooling) [62] | Additively HE, GC, SS | ✓ | 3.58 | 5.74 | 9.32 | 20.9 | 636.6 | 657.5 | 99 % |
| **EzPC** [29] | GC, Additive SS | ✓ | - | - | 5.1 | - | - | 501 | 99 % |
| **Chameleon (This Work)** | GC, GMW, Additive SS | ✓ | 1.25 | 0.99 | 2.24 | 5.4 | 5.1 | 10.5 | 99 % |

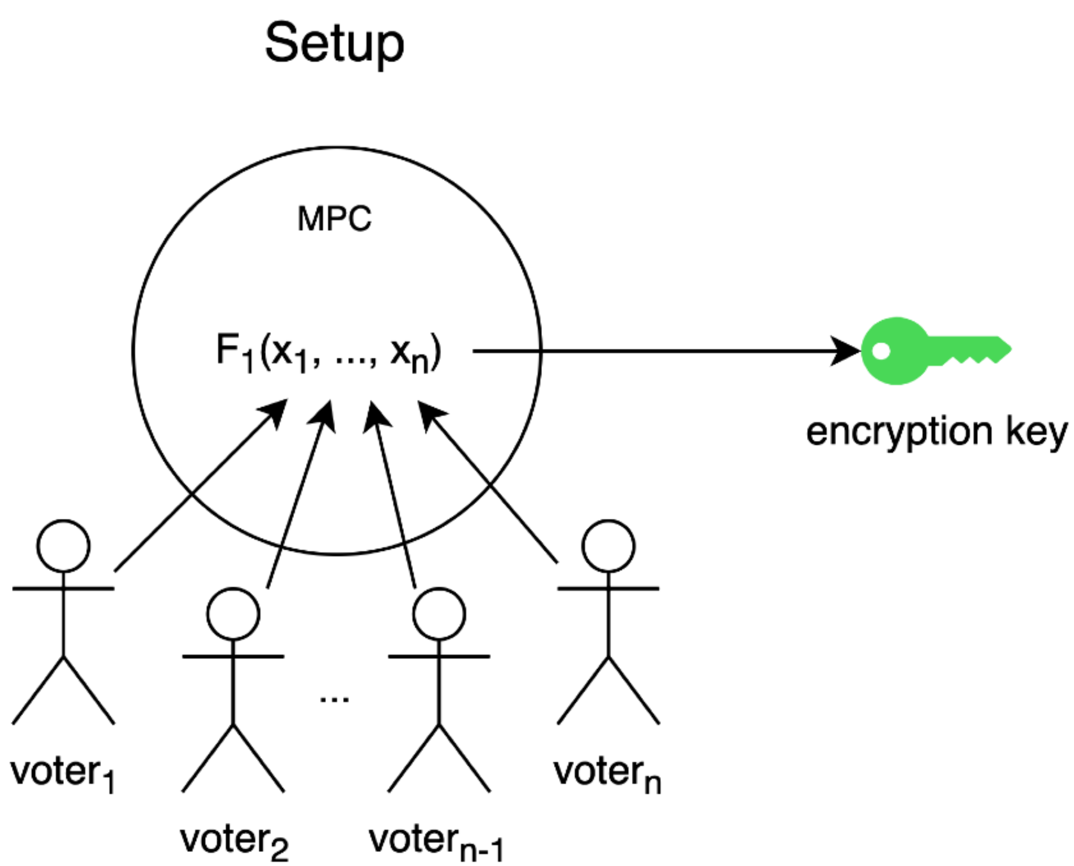
**Table 4: Classification time (in seconds) and communication costs (in megabytes) of Chameleon for different batch sizes of the MNIST dataset in the WAN setting (100 Mbit/s bandwidth, 100 ms round-trip time).**

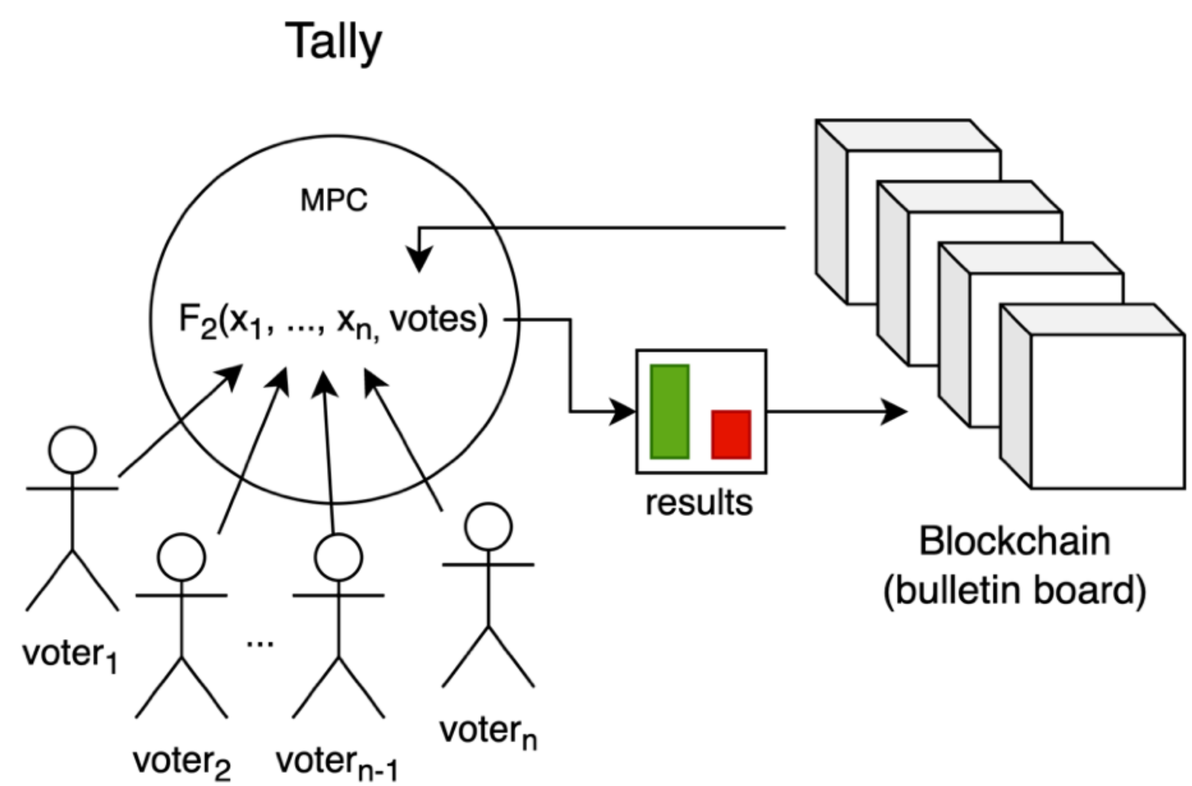
| | Classification Time (s) | | | Communication (MB) | | |
|---|---|---|---|---|---|---|
| Batch Size | Offline | Online | Total | Offline | Online | Total |
| 1 | 4.03 | 2.85 | 6.88 | 7.8 | 5.1 | 12.9 |
| 10 | 10.00 | 10.65 | 20.65 | 78.4 | 50.5 | 128.9 |
| 100 | 69.38 | 84.09 | 153.47 | 784.1 | 505.3 | 1289.4 |

# My work

## Secure internet voting using distributed networks




$$\mathtt{F}_1(x_1, \ldots, x_n) = \mathtt{DerivePubKey}(\mathtt{DerivePrivKey}(\mathtt{SS}(x_1, \ldots, x_n)))$$

$$, \ldots, x_n, votes) = \mathtt{Count}(\mathtt{Decrypt}(votes, \mathtt{DerivePrivKey}(\mathtt{SS}(x_1, \ldots, x_n))))$$

# References


- Smart, Nigel P., and Nigel P. Smart. *Cryptography made simple*. Springer, 2016.
- Evans, David, Vladimir Kolesnikov, and Mike Rosulek. "A pragmatic introduction to secure multi-party computation." _Foundations and Trends® in Privacy and Security_ 2.2-3 (2018): 70-246.
- Gilad-Bachrach, Ran, et al. "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy." *International conference on machine learning*. PMLR, 2016.
- Mohassel, Payman, and Yupeng Zhang. "Secureml: A system for scalable privacy-preserving machine learning." *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017.
- Liu, Jian, et al. "Oblivious neural network predictions via minionn transformations." *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 2017.
- Riazi, M. Sadegh, et al. "Chameleon: A hybrid secure computation framework for machine learning applications." *Proceedings of the 2018 on Asia conference on computer and communications security*. 2018.
- Juvekar, Chiraag, Vinod Vaikuntanathan, and Anantha Chandrakasan. "{GAZELLE}: A low latency framework for secure neural network inference." *27th USENIX Security Symposium (USENIX Security 18)*. 2018.
- Riazi, M. Sadegh, et al. "{XONN}:{XNOR-based} Oblivious Deep Neural Network Inference." *28th USENIX Security Symposium (USENIX Security 19)*. 2019.
- Chandran, Nishanth, et al. "EzPC: programmable, efficient, and scalable secure two-party computation for machine learning." *Cryptology ePrint Archive* (2017).
- Mishra, Pratyush, et al. "Delphi: A cryptographic inference service for neural networks." *29th USENIX Security Symposium (USENIX Security 20)*. 2020.
- Kumar, Nishant, et al. "Cryptflow: Secure tensorflow inference." *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020.

# Questions?

**Stanisław Barański**
**stanislaw.baranski@pg.edu.pl**
**https://stan.bar**
**Presentation: https://stan.bar/slides/2022_06_08_oblivious-neural-networks.pdf**