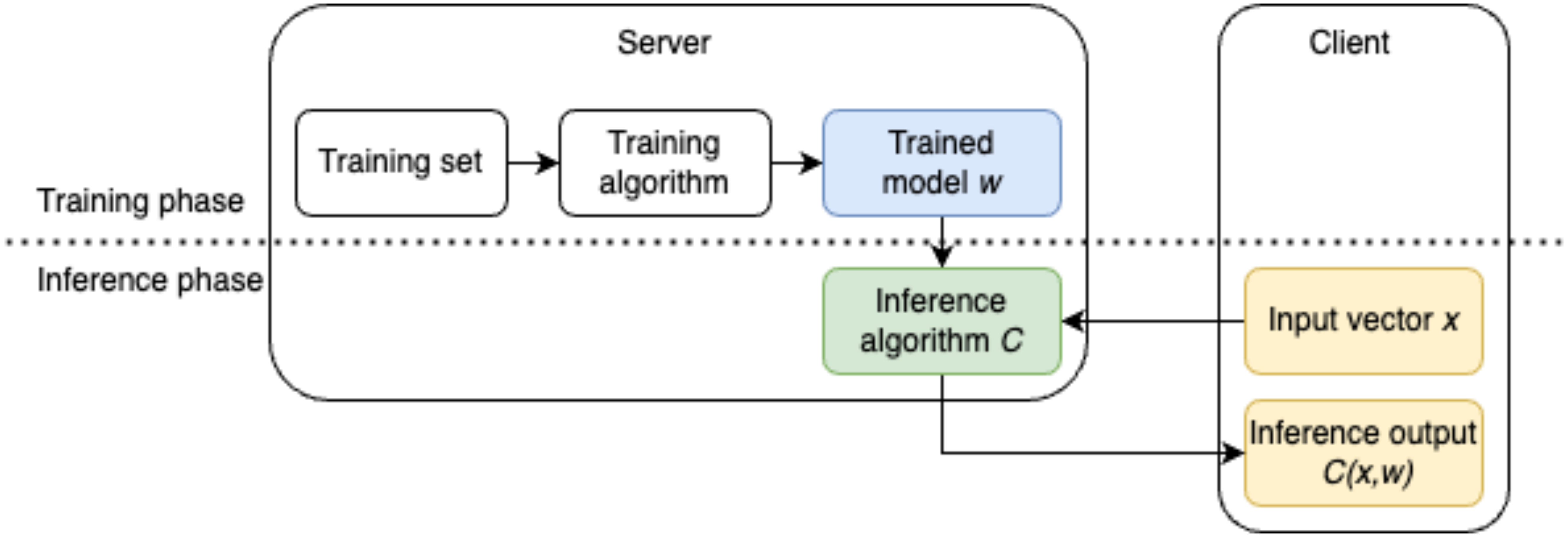
Privacy-Preserving Machine Learning

Stanislaw Baranski
stanislaw.baranski@pg.edu.pl
https://stan.bar

22.10.2024

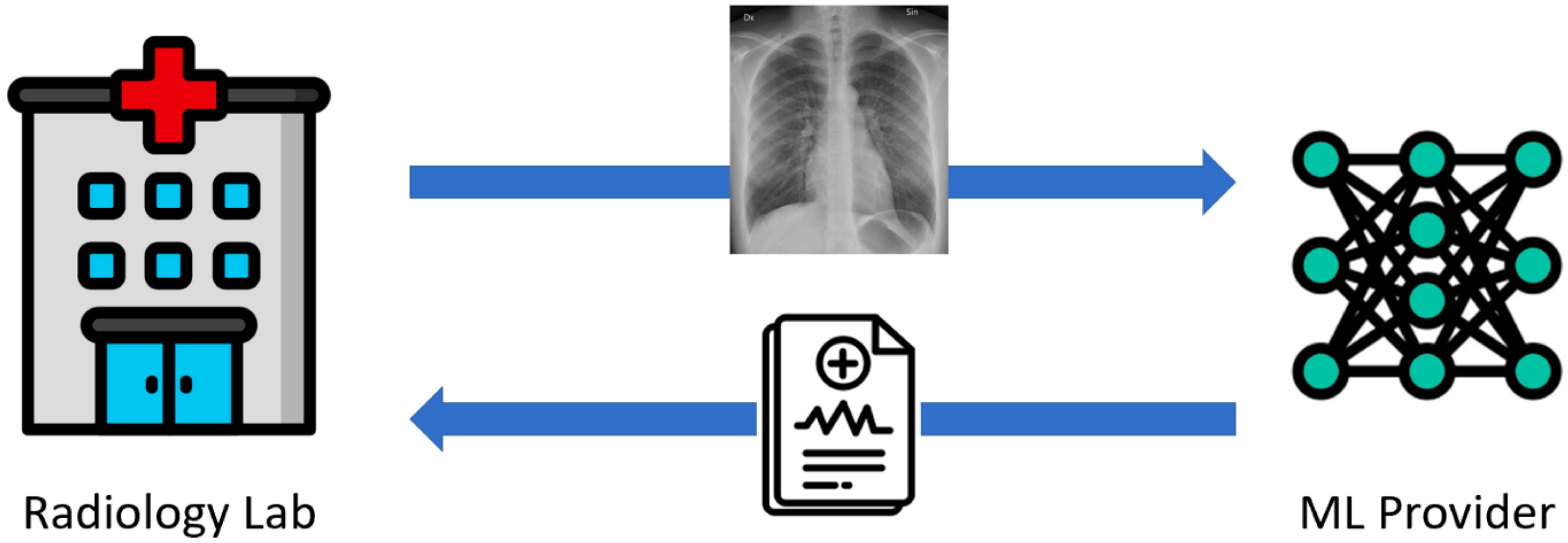
ML as a Service

- Client delegates the ML service to service provider (Server)
- Separation of specialization
- Cost reductions



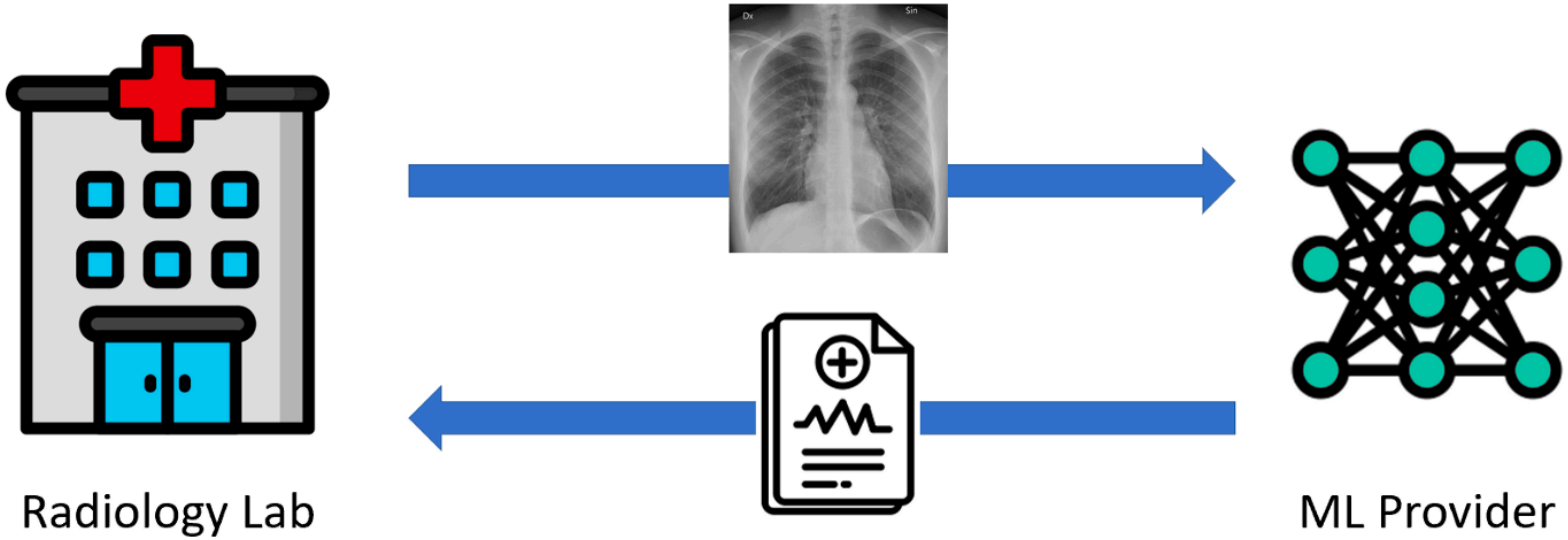
Use case

Medical prognosis using machine learning



Use case

Medical prognosis using machine learning



Radiology Lab

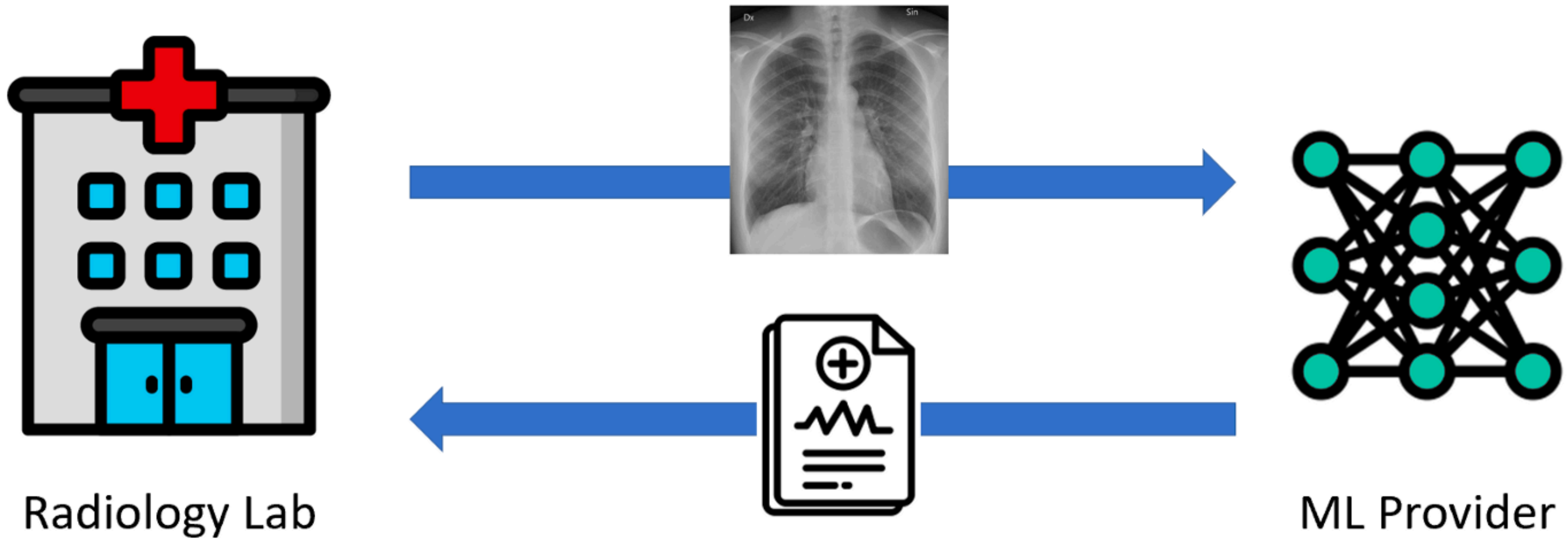
ML Provider

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

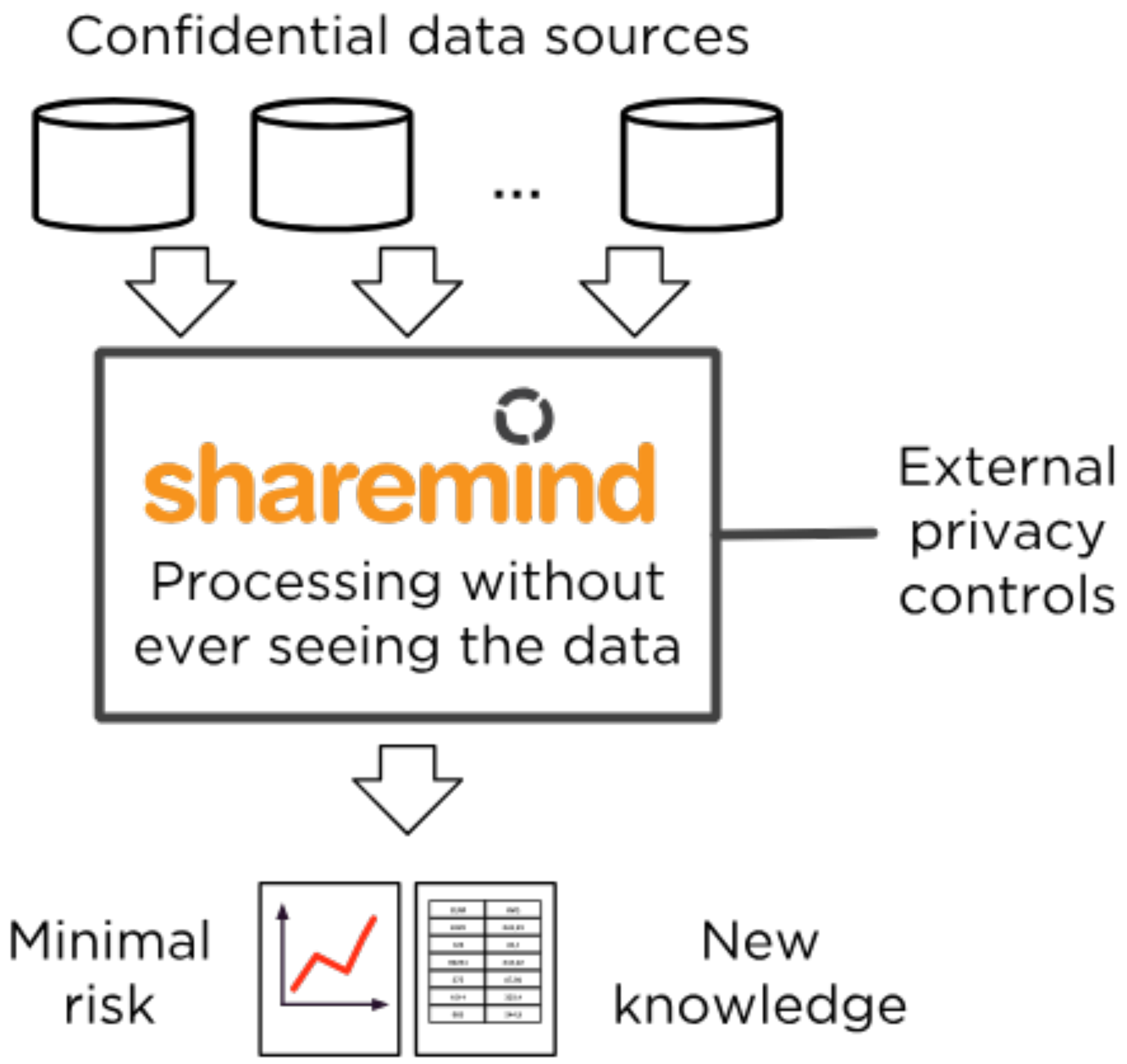
Medical prognosis using machine learning



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?

Use case

Estonian students study



Confidential data sources

...

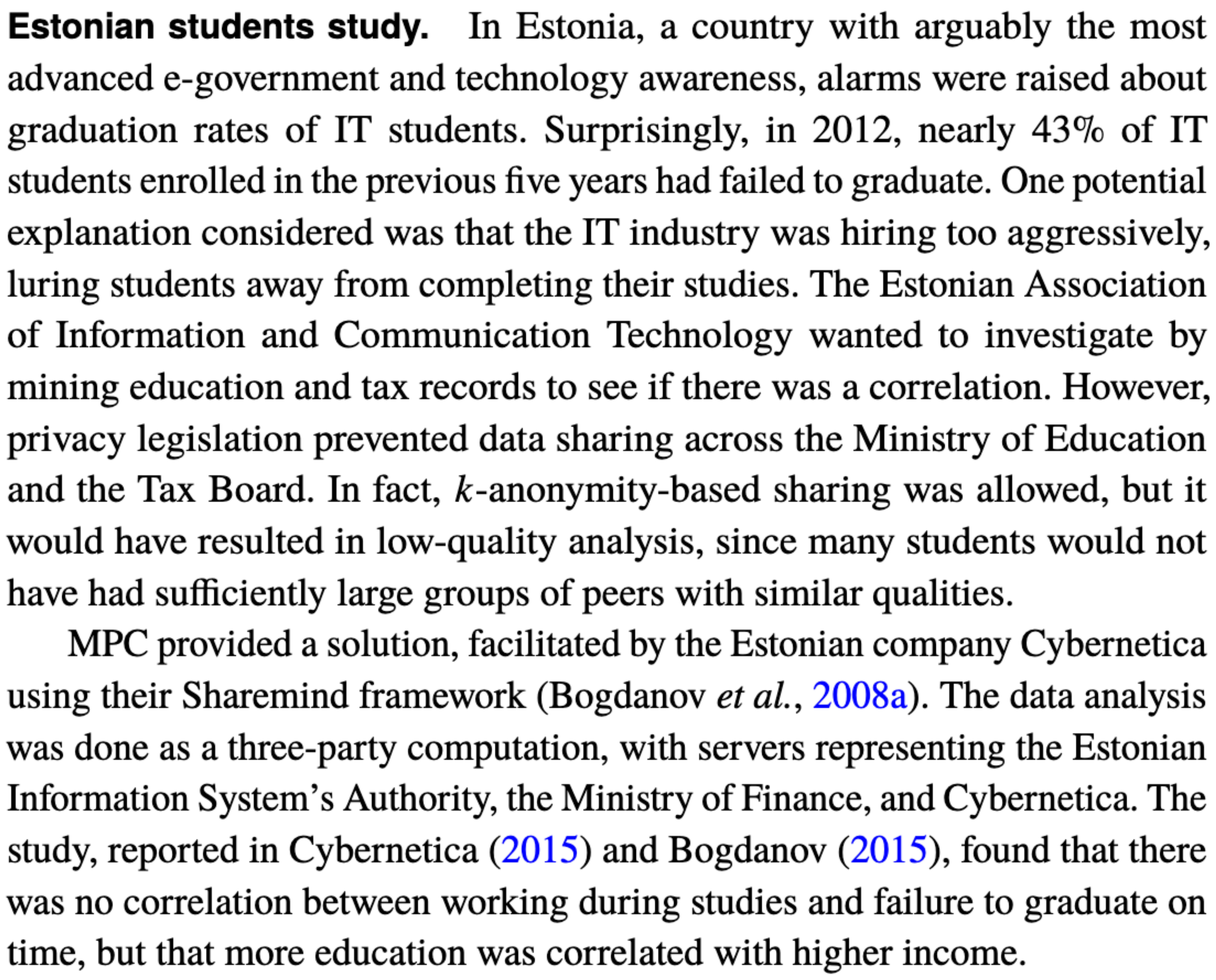
sharemind

Processing without ever seeing the data

External privacy controls

Minimal risk

New knowledge



Estonian students study. In Estonia, a country with arguably the most advanced e-government and technology awareness, alarms were raised about graduation rates of IT students. Surprisingly, in 2012, nearly 43% of IT students enrolled in the previous five years had failed to graduate. One potential explanation considered was that the IT industry was hiring too aggressively, luring students away from completing their studies. The Estonian Association of Information and Communication Technology wanted to investigate by mining education and tax records to see if there was a correlation. However, privacy legislation prevented data sharing across the Ministry of Education and the Tax Board. In fact, *k*-anonymity-based sharing was allowed, but it would have resulted in low-quality analysis, since many students would not have had sufficiently large groups of peers with similar qualities.

MPC provided a solution, facilitated by the Estonian company Cybernetica using their Sharemind framework (Bogdanov *et al.*, 2008a). The data analysis was done as a three-party computation, with servers representing the Estonian Information System's Authority, the Ministry of Finance, and Cybernetica. The study, reported in Cybernetica (2015) and Bogdanov (2015), found that there was no correlation between working during studies and failure to graduate on time, but that more education was correlated with higher income.

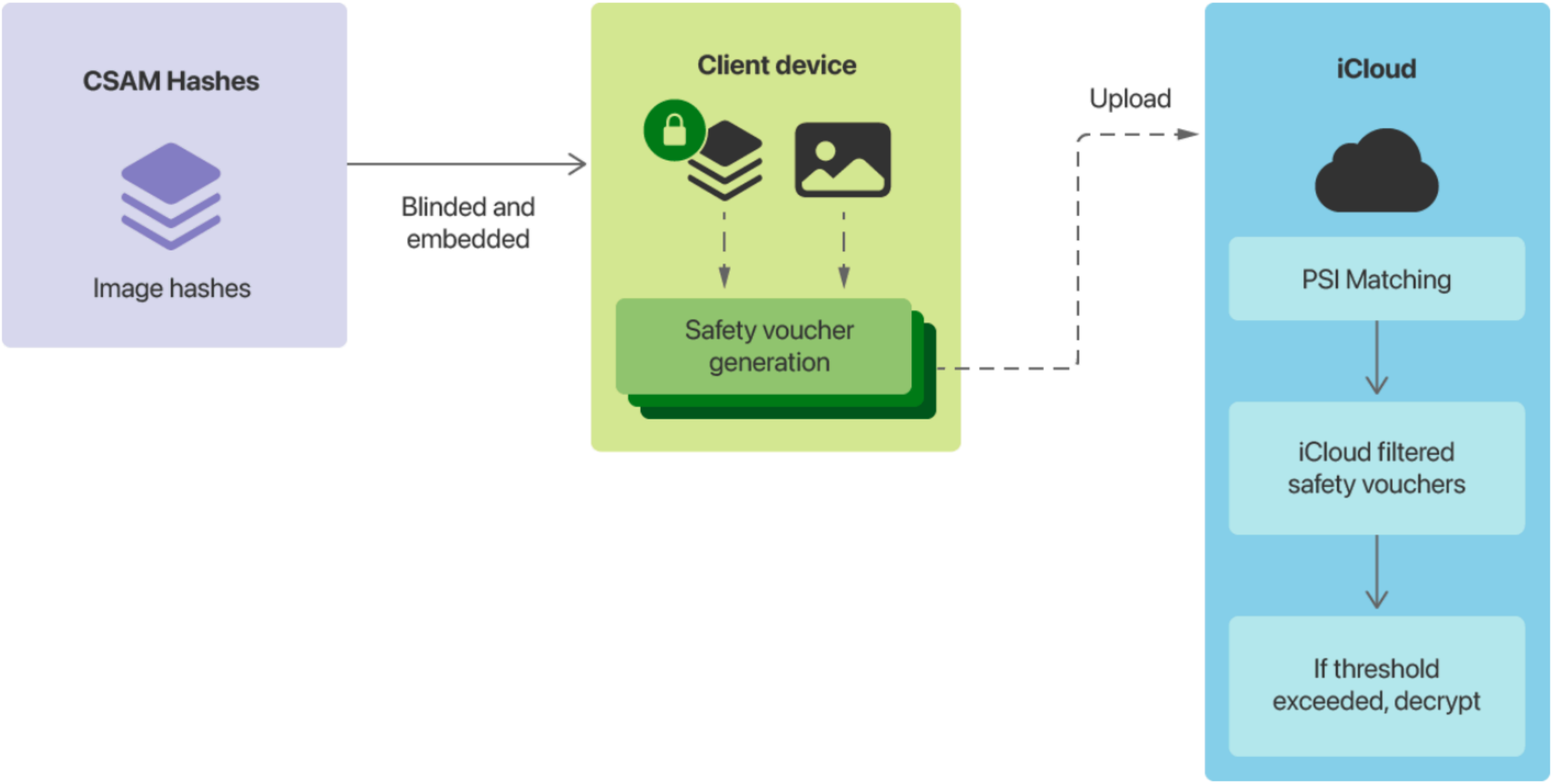
Source: Evans, David, Vladimir Kolesnikov, and Mike Rosulek. "A pragmatic introduction to secure multi-party computation." _Foundations and Trends® in Privacy and Security_ 2.2-3 (2018): 70-246.

Use case

Private set intersection

Apple CSAM Detection

CSAM Detection enables Apple to accurately identify and report iCloud users who store known Child Sexual Abuse Material (CSAM) in their iCloud Photos accounts. Apple servers flag accounts exceeding a threshold number of images that match a known database of CSAM image hashes so that Apple can provide relevant information to the National Center for Missing and Exploited Children (NCMEC). This process is secure, and is expressly designed to preserve user privacy.


CSAM Hashes
Image hashes
Blinded and embedded
Client device
Safety voucher generation
Upload
iCloud
PSI Matching
iCloud filtered safety vouchers
If threshold exceeded, decrypt

https://www.apple.com/child-safety/pdf/Apple_PSI_System_Security_Protocol_and_Analysis.pdf
https://www.apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.pdf

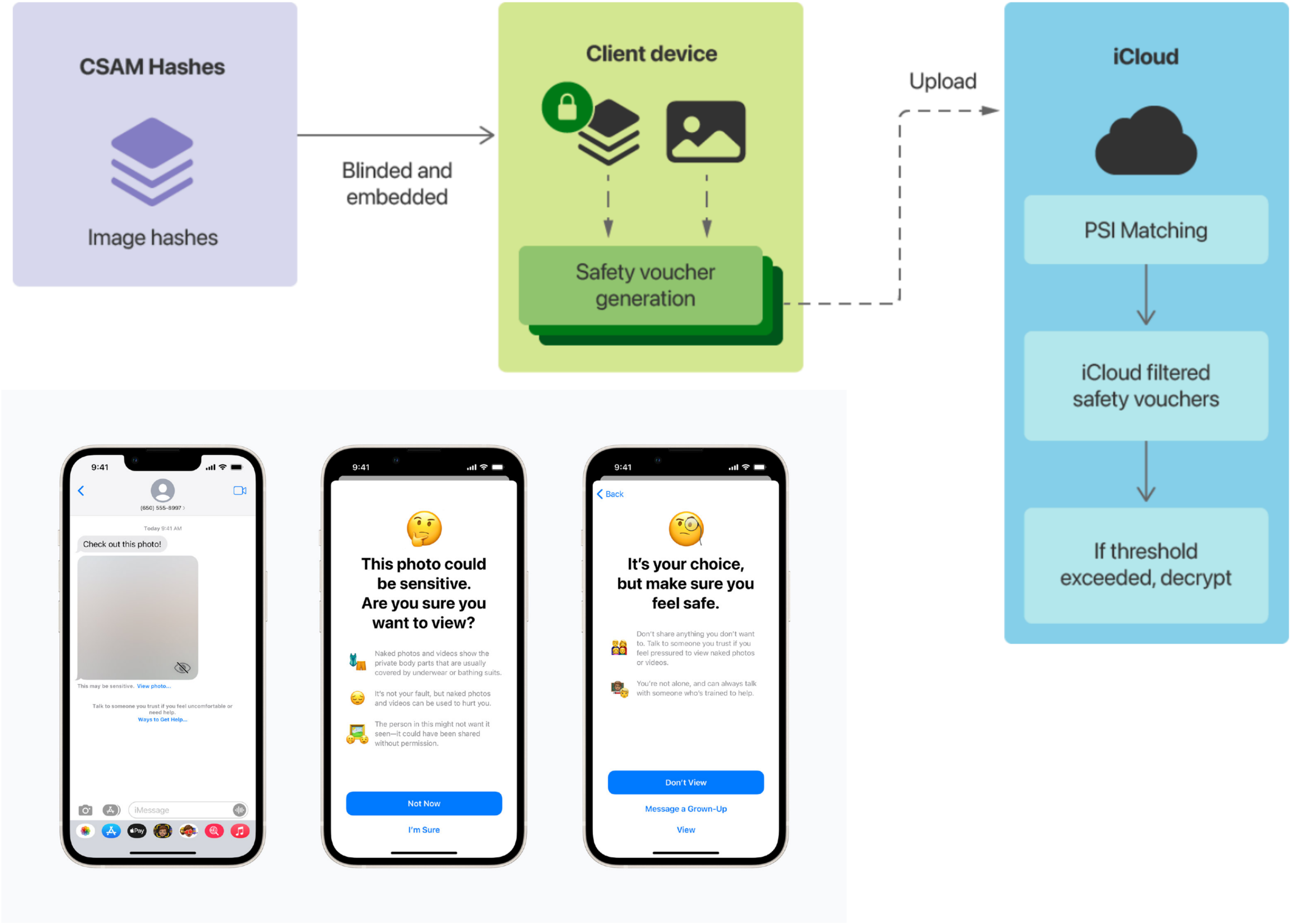
Use case

Private set intersection

Apple CSAM Detection

Password Monitoring

Communication Safety



Check out this photo!

This may be sensitive. View photo...

Talk to someone you trust if you feel uncomfortable or need help.
Ways to Get Help...

This photo could be sensitive. Are you sure you want to view?

Naked photos and videos show the private body parts that are usually covered by underwear or bathing suits.

It's not your fault, but naked photos and videos can be used to hurt you.

The person in this might not want it seen—it could have been shared without permission.

It's your choice, but make sure you feel safe.

Don't share anything you don't want to. Talk to someone you trust if you feel pressured to view naked photos or videos.

You're not alone, and can always talk with someone who's trained to help.

https://www.apple.com/child-safety/pdf/Apple_PSI_System_Security_Protocol_and_Analysis.pdf
https://www.apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.pdf

Other applications

- Secure machine learning
- privacy-preserving network security monitoring (Burkhart et al., 2010)
- privacy-preserving genomics (Wang et al., 2015a; Jagadeesh et al., 2017)
- Contact discovery (Li et al., 2013; De Cristofaro et al., 2013)
- Spam filtering on encrypted email (Gupta et al., 2017)
- Internet Voting, Credit score, everything which involves sensitive data…

Problem statement

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

Solution:
Oblivious Neural Networks

Solution:
~~Oblivious~~ Neural Networks
Privacy Preserving Machine Learning

Privacy Preserving Machine Learning

- The solution requires drawing knowledge from four fields of science
    - Machine learning
    - Computation theory,
    - Digital systems theory,
    - and Cryptography,

Neural Networks

- A neural network is a pipeline of layers. Each layer receives an input signal, processes it, and generates an output signal, which becomes the input for the next layer. The first layer receives the input data x, and the output signal of the last layer is the prediction result C(x, w).

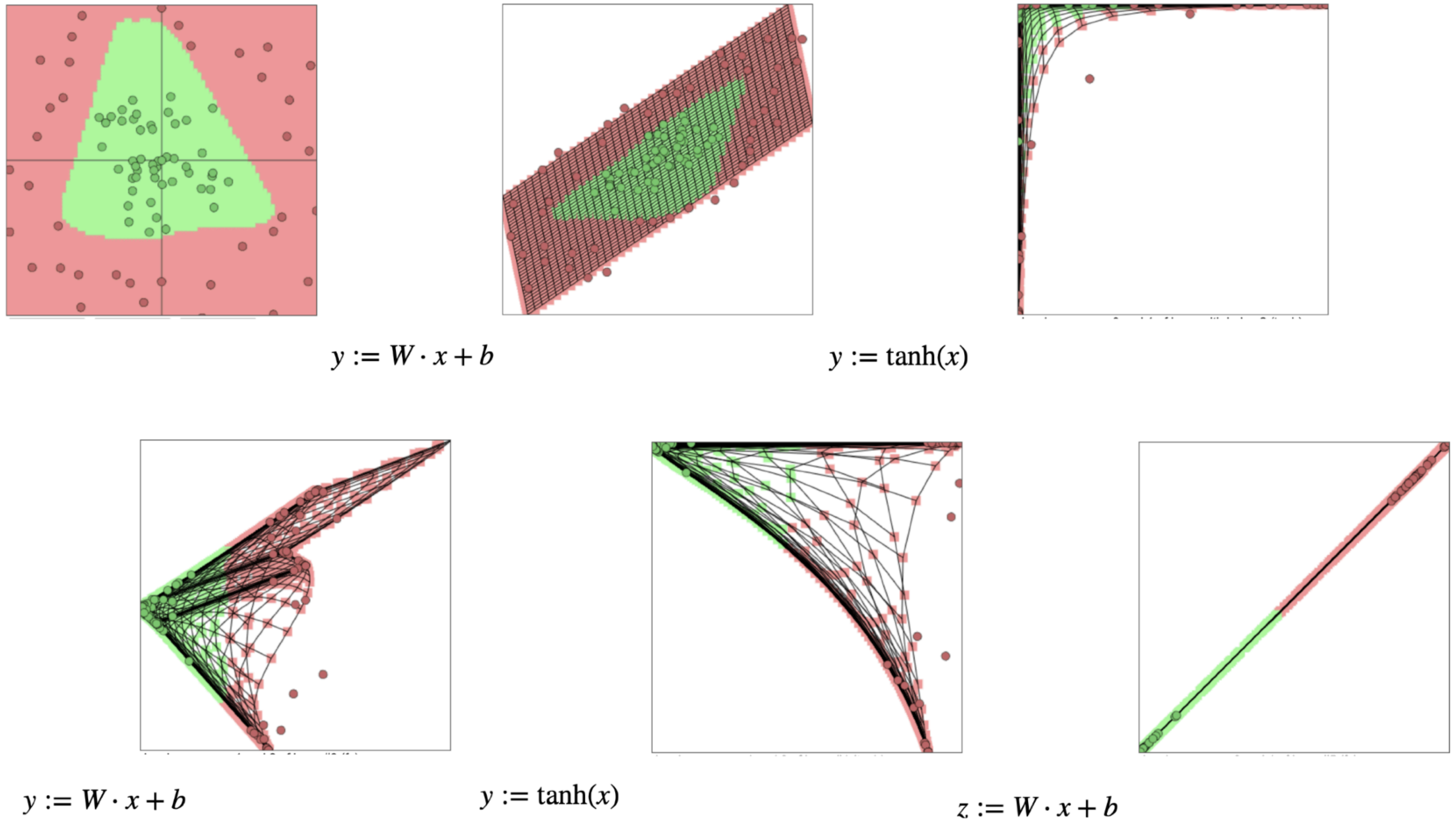
- A typical neural network layer performs a linear transformation (matrix multiplication and addition), followed by a nonlinear transformation (an activation function, and sometimes pooling to reduce data resolution). Predictions using neural networks can be represented as a pipeline of transformations:

$$x \to f_1 \to a_1 \to \ldots \to f_n \to a_n \to y$$

Neural Networks

- The goal of these transformations is to distort the input data space in such a way that it becomes linearly separable—that is, to create a line (if the input data is described in two dimensions), a plane (in three dimensions), or a hyperplane (in n + 1 dimensions) that can divide the input set into two subsets.

- Figure 7.2 illustrates the transformations of each layer of a sample neural network. Achieving linear separability of data (into green and red regions) is possible through a sequence of linear and nonlinear transformations. Linear transformations allow for rotating, tilting, and stretching the space, while nonlinear transformations enable deformation of the space.



Neural Networks

Linear transformations

- Matrix multiplication and addition are the most commonly used linear transformations in neural networks:
- $$y := W \cdot x + b$$
- where:
  - x is the input vector;
  - W is the weight matrix;
  - b is the bias vector;
  - y is the output vector.

Neural Networks

Linear transformations

- **Convolution** is a linear transformation that allows calculating the dot product between the weight tensor (filter, also called kernel) and an element of the matrix along with its neighboring elements. This process is repeated by sliding the filter over the input matrix. In practice, convolutions are transformed into a form of matrix multiplication and addition, which improves efficiency [12], similar to the previous equation, with the difference that both the input and the bias are matrices: $\mathbf{Y} := \mathbf{W} \cdot \mathbf{X} + \mathbf{B}$

Neural Networks

Non-linear transformations

- Neural networks use nonlinear transformations to model the nonlinear relationship between the input and output space.
- Activation functions can be classified into three categories.
  - **Piecewise Linear Activation Function**. This type of function can be represented as a set of n linear functions $f_i(x) = a_i x + b_i$, where x is constrained by a lower and upper limit for each interval. Examples of such functions are:
    - **Rectified Linear Units (ReLU)**: $f(x) = max(0,x)$
    - **Leaky ReLU**: $f(x) = max(0,x) + amin(0,x)$
    - **Maxout**: $f(x) = max(y_1, \ldots, y_n)$
  - **Smooth (Regular) Activation Function**
    - **Sigmoid (logistic)**: $f(x) = \frac{1}{1 + e^{-x}}$
    - **Hyperbolic tangent (tanh)**: $f(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$
    - **Softplus**: $f(x) = \log(e^x + 1)$
  - **Softmax** is most commonly used as the final layer of a neural network to determine the probability distribution for classification. The function is defined as
    - $softmax_i(x) = \frac{e^{x_i}}{\sum_k e^{x_k}}$

Neural Networks

Non-linear transformations

- **Pooling** is an operation that reduces the resolution of a matrix. Also known as a folding operation, it involves organizing the input data into subgroups and aggregating each subgroup, thereby reducing the dimensionality of the input data. Most commonly, the elements are aggregated using an averaging function (mean pooling) or maximization (max pooling).

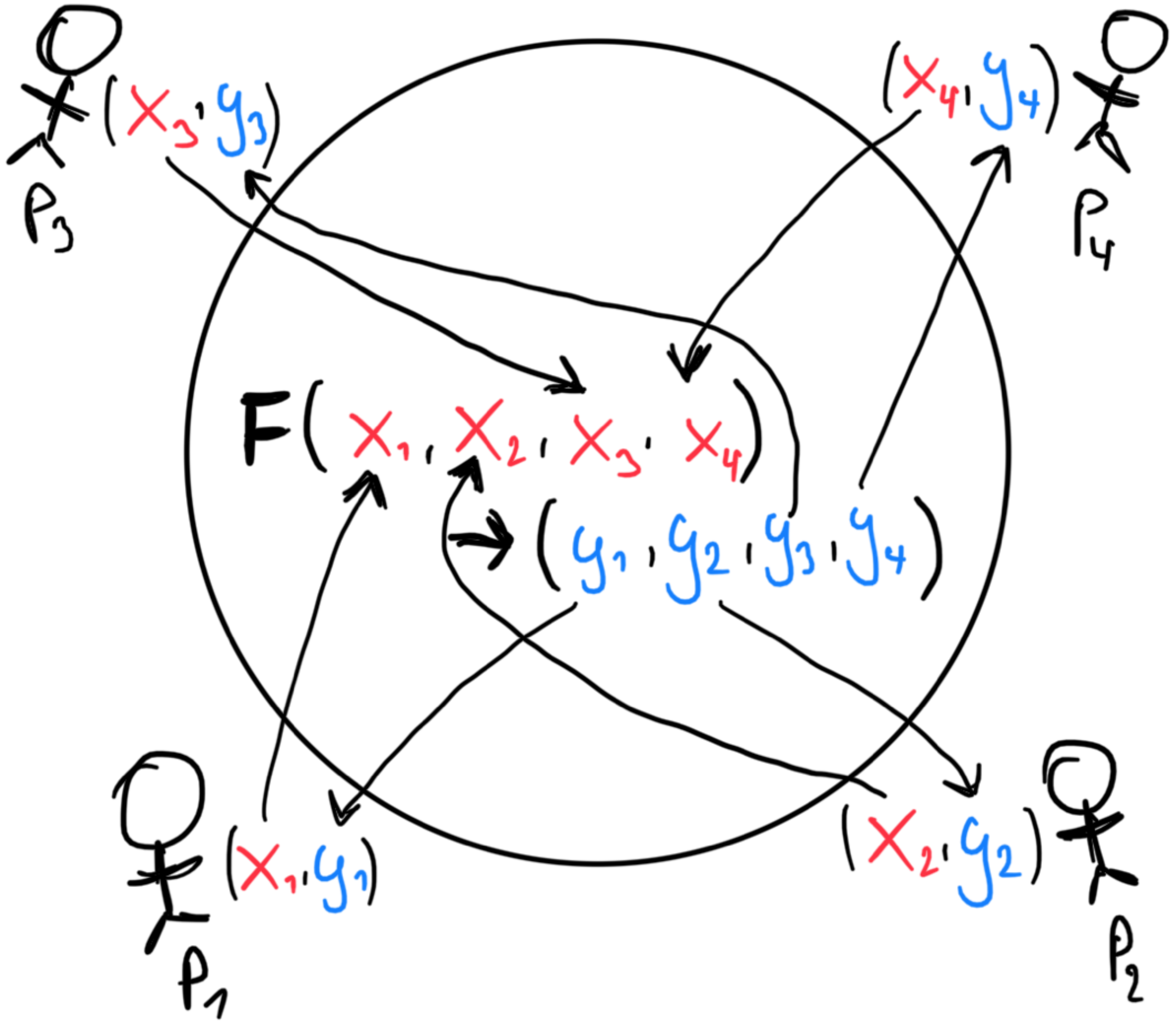
Privacy preserving neural networks

- The prediction phase using neural networks boils down to a sequence of linear and nonlinear transformations.
    - Linear transformations consist of matrix multiplication and addition.
    - Nonlinear transformations involve applying activation functions or pooling operations.
- Therefore, to make the entire inference process privacy-preserving, **it is sufficient to make the operations of matrix multiplication and addition, as well as activation functions and pooling operations, privacy-preserving**.

Secure Multi-party computation (MPC)

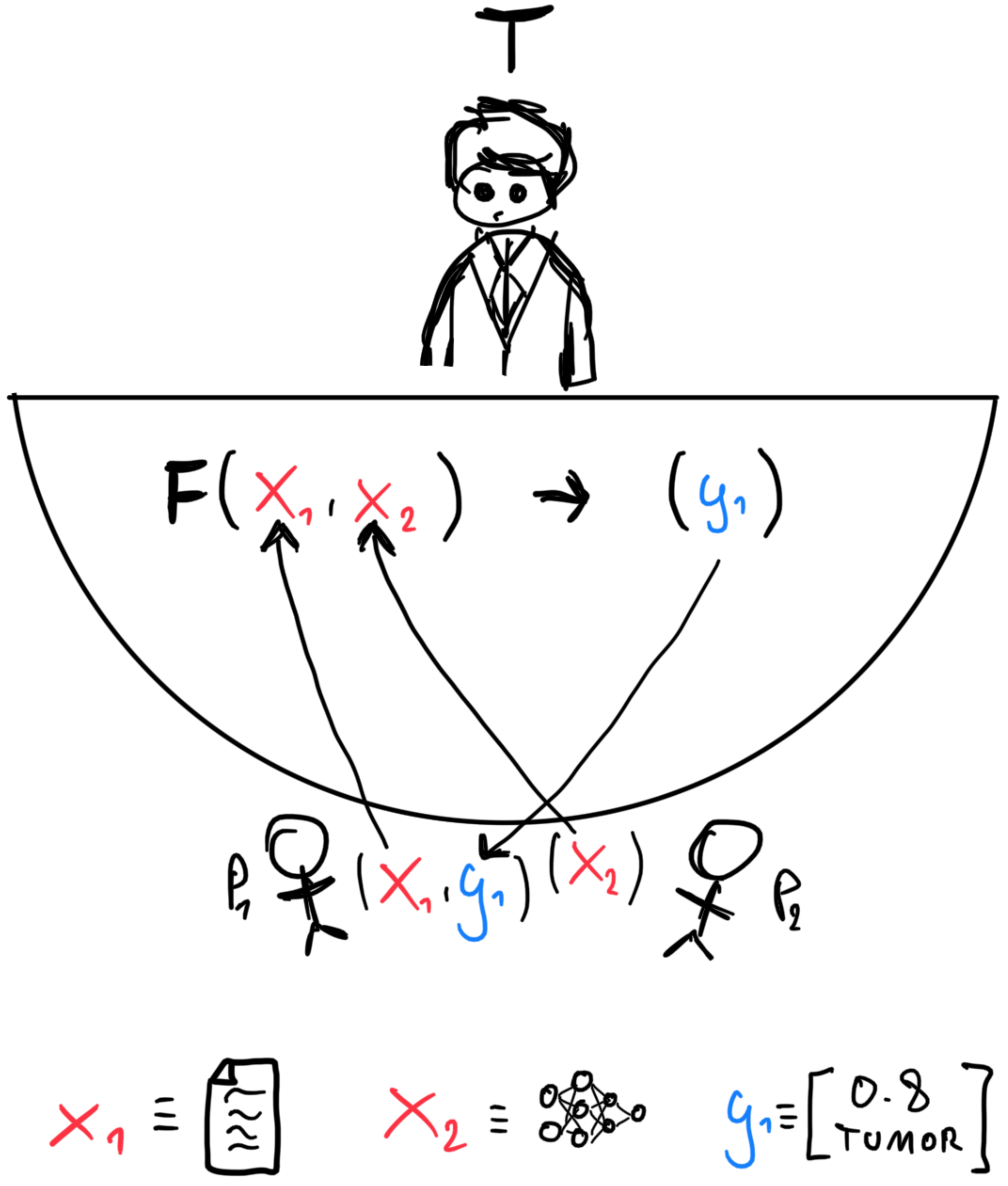
Secure multi-party computation (MPC)

- Secure multi-party computation (MPC) is a technique that allows n independent participants to execute an arbitrary algorithm $F(x_1, \ldots, x_n)$ in such a way that none of them learns anything about the input data of the other participants.
- For our purposes, we will focus on the case where only two parties (a client and a server) participate in executing the algorithm . This specific case is called **secure two-party computation (2PC).**



Secure two-party computation (2PC)

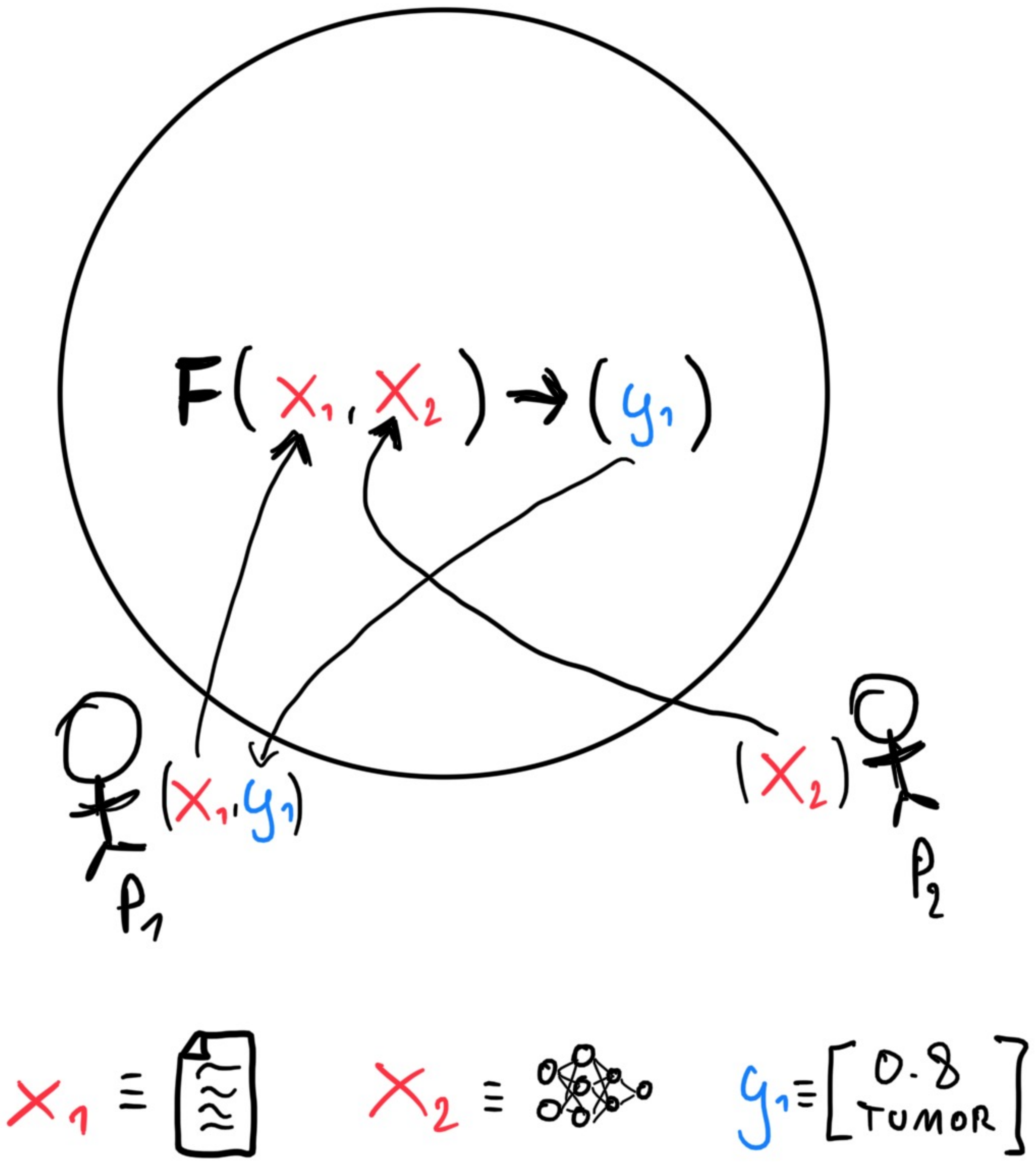
- The simplest way to execute the algorithm F so that neither party learns anything about the other party's input data is to introduce a trusted third party (TTP), which receives the input data from both participants, executes the functionality F, and returns the computation result.



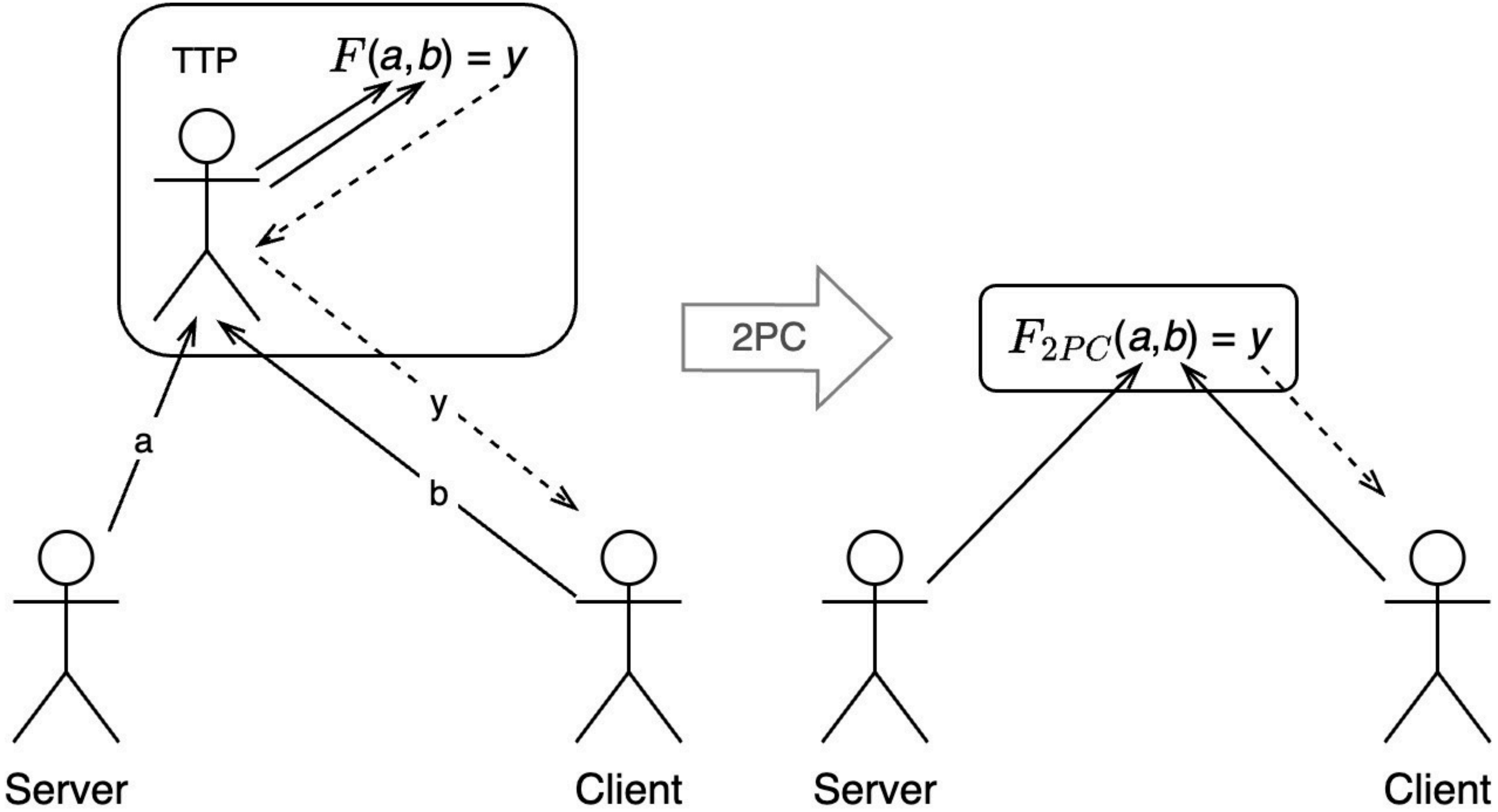
Secure two-party computation (2PC)

- Introducing a trusted third party is problematic for several reasons, the most important being the often unacceptable assumption regarding the actual honesty of the TTP. Therefore, this is an assumption we strive to avoid at all costs when designing trustworthy information systems.

- 2PC addresses this problem by enabling the same functionality without the help of a TTP.



Secure two-party computation (2PC)



Secure two-party computation (2PC)

Garbled Circuit (GC)

For many years, 2PC remained largely a theoretical pursuit among cryptographers. The first practical solution is Yao’s Garbled Circuit (GC) (Yao 1982).

Secure two-party computation (2PC)

Garbled Circuit (GC)

For many years, 2PC remained largely a theoretical pursuit among cryptographers. The first practical solution is Yao’s Garbled Circuit (GC) (Yao 1982).

He proposed a Yao’s Millionaires Problem which states: "Two millionaires wish to know who is richer without revealing their actual wealth.”

So the goal is to compute $a \geq b$ where a is the first party’s *private* input and b is the second party’s *private* input.

$$f(a, b) = a \geq b$$

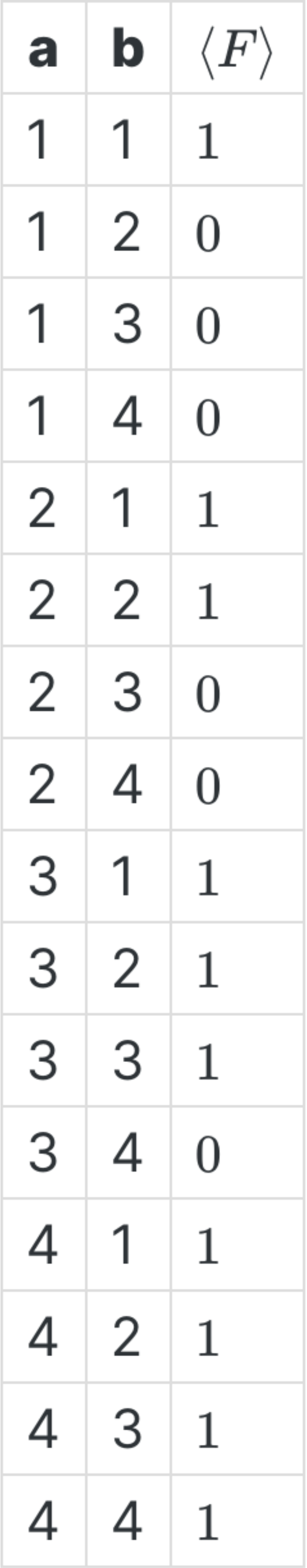
Secure two-party computation (2PC)

Yao's Millionaires Problem

To simplify the example, let's assume that the numbers a and b represent the number of digits in each student's wealth, and that the maximum number of digits is 4. Let's call a as **Server**, b as **Client**. The domain of the function F is $X = 1,2,3,4$, and the codomain is $Y = \{1,0\}$, where 1 represents true and 0 represents false.

In the first step, one of the parties (let's assume it's the server) creates a lookup table of all possible evaluations of the function F, i.e., $\langle F \rangle = \langle F(1,1) = 1, F(1,2) = 0, \ldots, F(4,4) = 1 \rangle$.

Evaluating the function $f(a, b) = a \geq b$ represented in the form of a table $\langle F \rangle$ involves retrieving the corresponding row from the column $\langle F \rangle$, for the values a and b.



a	b	$\langle F \rangle$
1	1	1
1	2	0
1	3	0
1	4	0
2	1	1
2	2	1
2	3	0
2	4	0
3	1	1
3	2	1
3	3	1
3	4	0
4	1	1
4	2	1
4	3	1
4	4	1

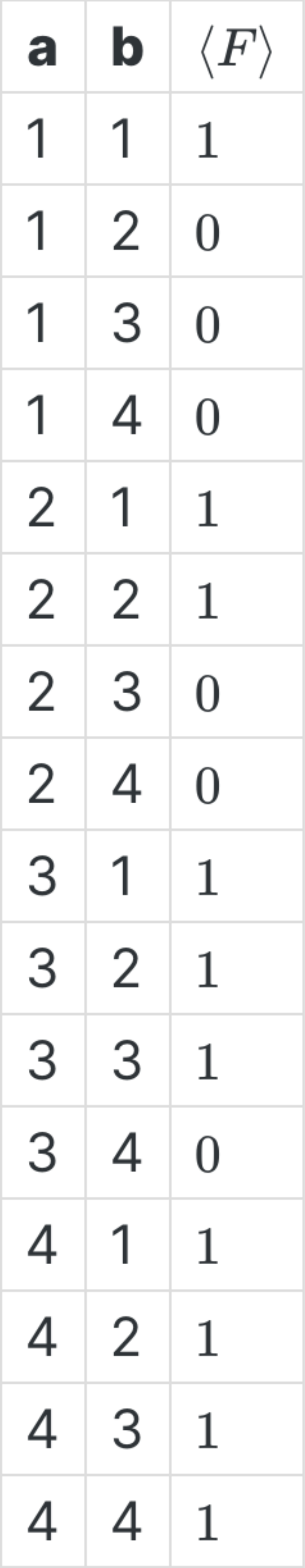
Secure two-party computation (2PC)

Yao’s Millionaires Problem

To create an unconscious version of the function F represented in the form of the table $\langle F \rangle$, the server encrypts the table using randomly generated keys for each value of a and b. Specifically, for each value $a \in X$ and $b \in X$, the server assigns a strong pseudorandom key $k_{\text{a}} \in 0{,}1^{\kappa}$ and $k_{\text{b}} \in 0{,}1^{\kappa}$, where κ is the security parameter indicating the bit length of the encryption key.

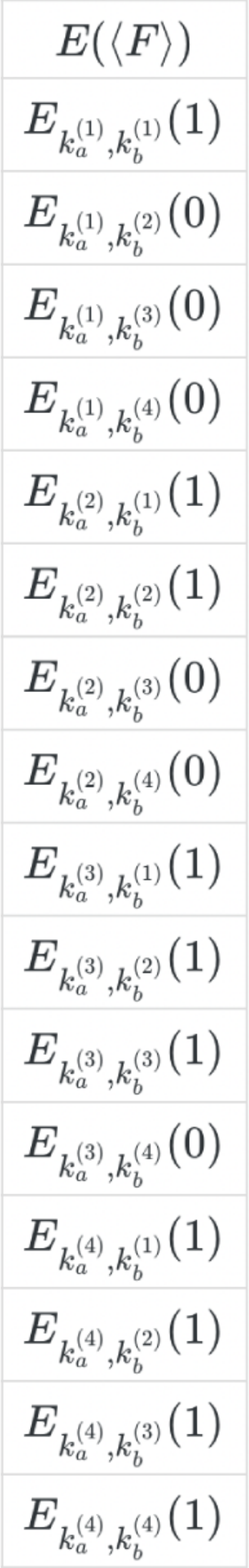
This creates a total of $|X| + |X| = 8$ keys $(k_a^{(1)}, k_a^{(2)}, k_a^{(3)}, k_a^{(4)}, k_b^{(1)}, k_b^{(2)}, k_b^{(3)}, k_b^{(4)})$ used to encrypt the function F result for each combination of arguments a and b. Let $E_{k_a,k_b}(\,\cdot\,)$ be a symmetric encryption function that is parameterized by two encryption keys: the server key k_a for value a, and the client key k_b for value b. Encryption with two keys can be implemented in several ways; one approach is double encryption—first with the server’s key, and then with the client’s key. Formally, this is represented as $E_{k_a,k_b}(\,\cdot\,) = E_{k_b}(E_{k_a}(\,\cdot\,))$.

We denote the encrypted table $\langle F \rangle$ as $E(\langle F \rangle)$



a	b	$\langle F \rangle$
1	1	1
1	2	0
1	3	0
1	4	0
2	1	1
2	2	1
2	3	0
2	4	0
3	1	1
3	2	1
3	3	1
3	4	0
4	1	1
4	2	1
4	3	1
4	4	1

a)



$E(\langle F \rangle)$
$E_{k_a^{(1)},k_b^{(1)}}(1)$
$E_{k_a^{(1)},k_b^{(2)}}(0)$
$E_{k_a^{(1)},k_b^{(3)}}(0)$
$E_{k_a^{(1)},k_b^{(4)}}(0)$
$E_{k_a^{(2)},k_b^{(1)}}(1)$
$E_{k_a^{(2)},k_b^{(2)}}(1)$
$E_{k_a^{(2)},k_b^{(3)}}(0)$
$E_{k_a^{(2)},k_b^{(4)}}(0)$
$E_{k_a^{(3)},k_b^{(1)}}(1)$
$E_{k_a^{(3)},k_b^{(2)}}(1)$
$E_{k_a^{(3)},k_b^{(3)}}(1)$
$E_{k_a^{(3)},k_b^{(4)}}(0)$
$E_{k_a^{(4)},k_b^{(1)}}(1)$
$E_{k_a^{(4)},k_b^{(2)}}(1)$
$E_{k_a^{(4)},k_b^{(3)}}(1)$
$E_{k_a^{(4)},k_b^{(4)}}(1)$

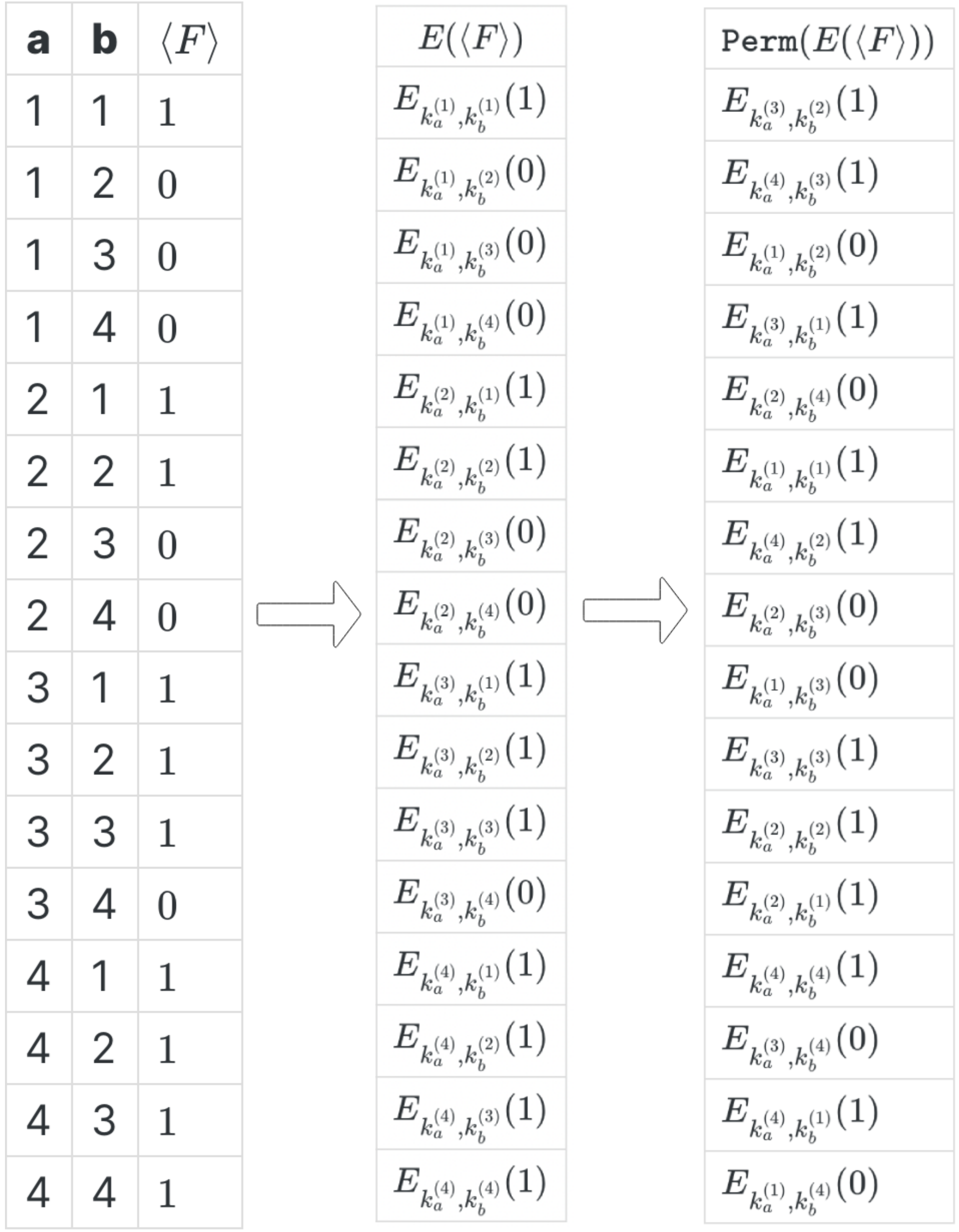
b)

Secure two-party computation (2PC)

Yao’s Millionaires Problem

The table prepared by the server is almost ready to be sent to the client. Almost, because if the client received the table in this form, they could deduce both a and b based on the specific row of the table. Therefore, the final modification is to apply a random permutation so that the client cannot deduce the value of a. This results in the table Perm$(E(\langle F \rangle))$

At this point, the table is ready. The server sends the table to the client along with the key $k_a^{(i)}$, where i is the value chosen by the server. Let’s assume that this value is $i = 4$, so the server sends the key $k_a^{(4)}$ to the client.



a	b	$\langle F \rangle$
1	1	1
1	2	0
1	3	0
1	4	0
2	1	1
2	2	1
2	3	0
2	4	0
3	1	1
3	2	1
3	3	1
3	4	0
4	1	1
4	2	1
4	3	1
4	4	1

a)

$E(\langle F \rangle)$
$E_{k_a^{(1)},k_b^{(1)}}(1)$
$E_{k_a^{(1)},k_b^{(2)}}(0)$
$E_{k_a^{(1)},k_b^{(3)}}(0)$
$E_{k_a^{(1)},k_b^{(4)}}(0)$
$E_{k_a^{(2)},k_b^{(1)}}(1)$
$E_{k_a^{(2)},k_b^{(2)}}(1)$
$E_{k_a^{(2)},k_b^{(3)}}(0)$
$E_{k_a^{(2)},k_b^{(4)}}(0)$
$E_{k_a^{(3)},k_b^{(1)}}(1)$
$E_{k_a^{(3)},k_b^{(2)}}(1)$
$E_{k_a^{(3)},k_b^{(3)}}(1)$
$E_{k_a^{(3)},k_b^{(4)}}(0)$
$E_{k_a^{(4)},k_b^{(1)}}(1)$
$E_{k_a^{(4)},k_b^{(2)}}(1)$
$E_{k_a^{(4)},k_b^{(3)}}(1)$
$E_{k_a^{(4)},k_b^{(4)}}(1)$

b)

$\texttt{Perm}(E(\langle F \rangle))$
$E_{k_a^{(3)},k_b^{(2)}}(1)$
$E_{k_a^{(4)},k_b^{(3)}}(1)$
$E_{k_a^{(1)},k_b^{(2)}}(0)$
$E_{k_a^{(3)},k_b^{(1)}}(1)$
$E_{k_a^{(2)},k_b^{(4)}}(0)$
$E_{k_a^{(1)},k_b^{(1)}}(1)$
$E_{k_a^{(4)},k_b^{(2)}}(1)$
$E_{k_a^{(2)},k_b^{(3)}}(0)$
$E_{k_a^{(1)},k_b^{(3)}}(0)$
$E_{k_a^{(3)},k_b^{(3)}}(1)$
$E_{k_a^{(2)},k_b^{(2)}}(1)$
$E_{k_a^{(2)},k_b^{(1)}}(1)$
$E_{k_a^{(4)},k_b^{(4)}}(1)$
$E_{k_a^{(3)},k_b^{(4)}}(0)$
$E_{k_a^{(4)},k_b^{(1)}}(1)$
$E_{k_a^{(1)},k_b^{(4)}}(0)$

c)

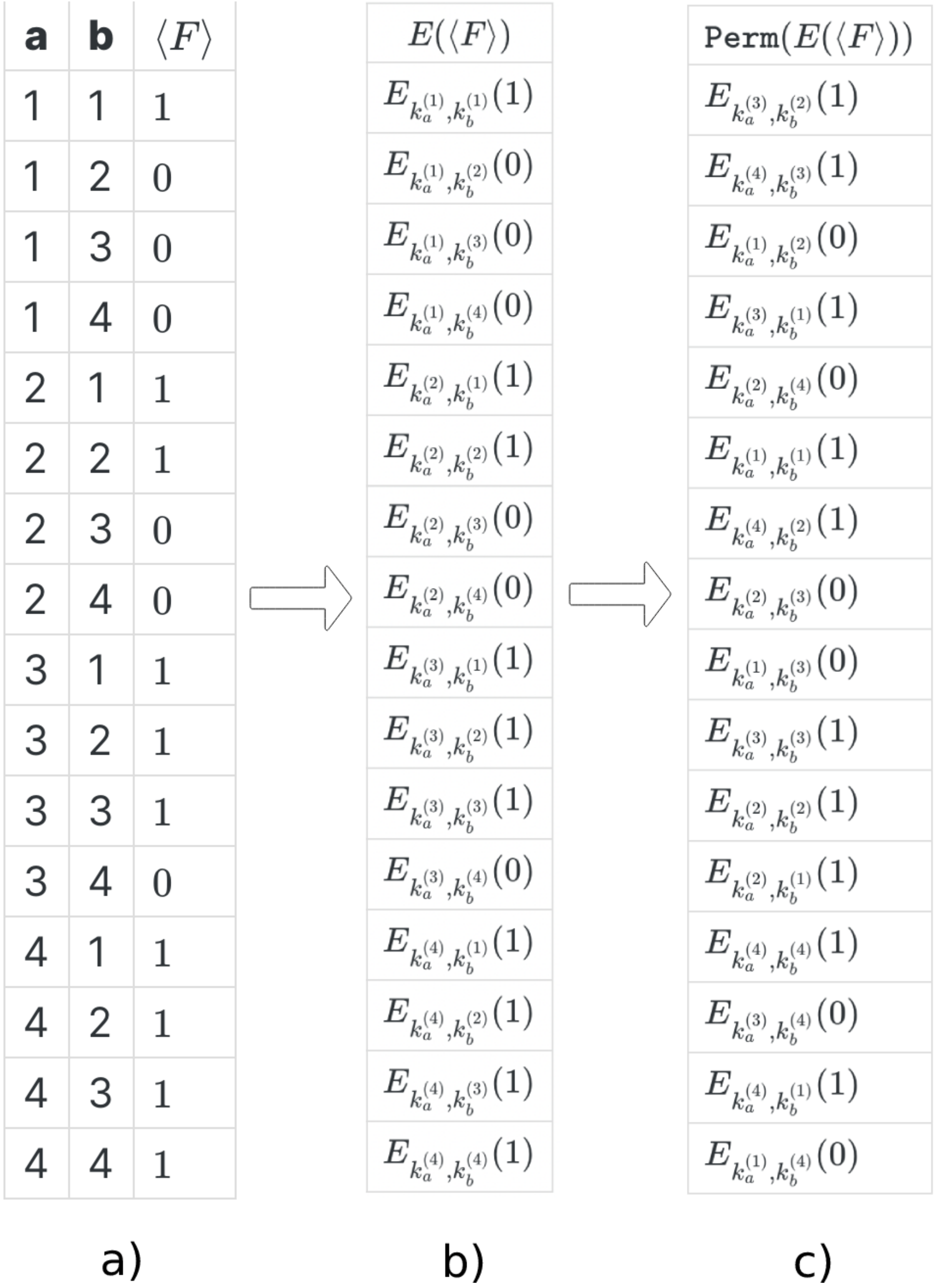
Secure two-party computation (2PC)

Yao’s Millionaires Problem

Note that the key $k_a^{(4)}$ itself is a random bit string that carries no information. It does not allow decryption of any row in the table and does not provide any clue about the server’s input. The obliviousness of the computation is achieved only in the next step, which proceeds as follows.

The server conducts an Oblivious Transfer (OT) procedure with the client, allowing the client to choose one key from the set of offered keys $k_b^{(1)}, k_b^{(2)}, k_b^{(3)}, k_b^{(4)}$ in such a way that the server does not learn which key was chosen, and the client learns nothing about the other keys besides the one they selected. In our example, let’s assume the client’s wealth has a digit count of $j = 3$, so the client selects the key $k_b^{(3)}$.

At this point, the client has all the necessary elements to reconstruct the value of the function $F(a, b)$. Combined with the previously received key $k_a^{(4)}$, the client can decrypt only the row encrypted with both keys $k_a^{(4)}$ and $k_b^{(3)}$, which is the second row with the value $E_{k_a^{(4)},k_b^{(3)}}(1)$. The client decrypts the row as $E_{k_a^{(4)},k_b^{(3)}}(E_{k_a^{(4)},k_b^{(3)}}(1)) = 1$, obtaining a positive value that indicates that the server’s wealth is greater, i.e., $F(4,3) = 4 \geq 3$.



a	b	$\langle F \rangle$
1	1	1
1	2	0
1	3	0
1	4	0
2	1	1
2	2	1
2	3	0
2	4	0
3	1	1
3	2	1
3	3	1
3	4	0
4	1	1
4	2	1
4	3	1
4	4	1

a)

$E(\langle F \rangle)$
$E_{k_a^{(1)},k_b^{(1)}}(1)$
$E_{k_a^{(1)},k_b^{(2)}}(0)$
$E_{k_a^{(1)},k_b^{(3)}}(0)$
$E_{k_a^{(1)},k_b^{(4)}}(0)$
$E_{k_a^{(2)},k_b^{(1)}}(1)$
$E_{k_a^{(2)},k_b^{(2)}}(1)$
$E_{k_a^{(2)},k_b^{(3)}}(0)$
$E_{k_a^{(2)},k_b^{(4)}}(0)$
$E_{k_a^{(3)},k_b^{(1)}}(1)$
$E_{k_a^{(3)},k_b^{(2)}}(1)$
$E_{k_a^{(3)},k_b^{(3)}}(1)$
$E_{k_a^{(3)},k_b^{(4)}}(0)$
$E_{k_a^{(4)},k_b^{(1)}}(1)$
$E_{k_a^{(4)},k_b^{(2)}}(1)$
$E_{k_a^{(4)},k_b^{(3)}}(1)$
$E_{k_a^{(4)},k_b^{(4)}}(1)$

b)

$\texttt{Perm}(E(\langle F \rangle))$
$E_{k_a^{(3)},k_b^{(2)}}(1)$
$E_{k_a^{(4)},k_b^{(3)}}(1)$
$E_{k_a^{(1)},k_b^{(2)}}(0)$
$E_{k_a^{(3)},k_b^{(1)}}(1)$
$E_{k_a^{(2)},k_b^{(4)}}(0)$
$E_{k_a^{(1)},k_b^{(1)}}(1)$
$E_{k_a^{(4)},k_b^{(2)}}(1)$
$E_{k_a^{(2)},k_b^{(3)}}(0)$
$E_{k_a^{(1)},k_b^{(3)}}(0)$
$E_{k_a^{(3)},k_b^{(3)}}(1)$
$E_{k_a^{(2)},k_b^{(2)}}(1)$
$E_{k_a^{(2)},k_b^{(1)}}(1)$
$E_{k_a^{(4)},k_b^{(4)}}(1)$
$E_{k_a^{(3)},k_b^{(4)}}(0)$
$E_{k_a^{(4)},k_b^{(1)}}(1)$
$E_{k_a^{(1)},k_b^{(4)}}(0)$

c)

Secure two-party computation (2PC)

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
- $2^{64} \times 4\text{bytes} = \text{too large}$

Secure two-party computation (2PC)

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

Secure two-party computation (2PC)

Function as a lookup problem

Logic gates have a domain size of 4 = |{0,1}| x |{0,1}|.

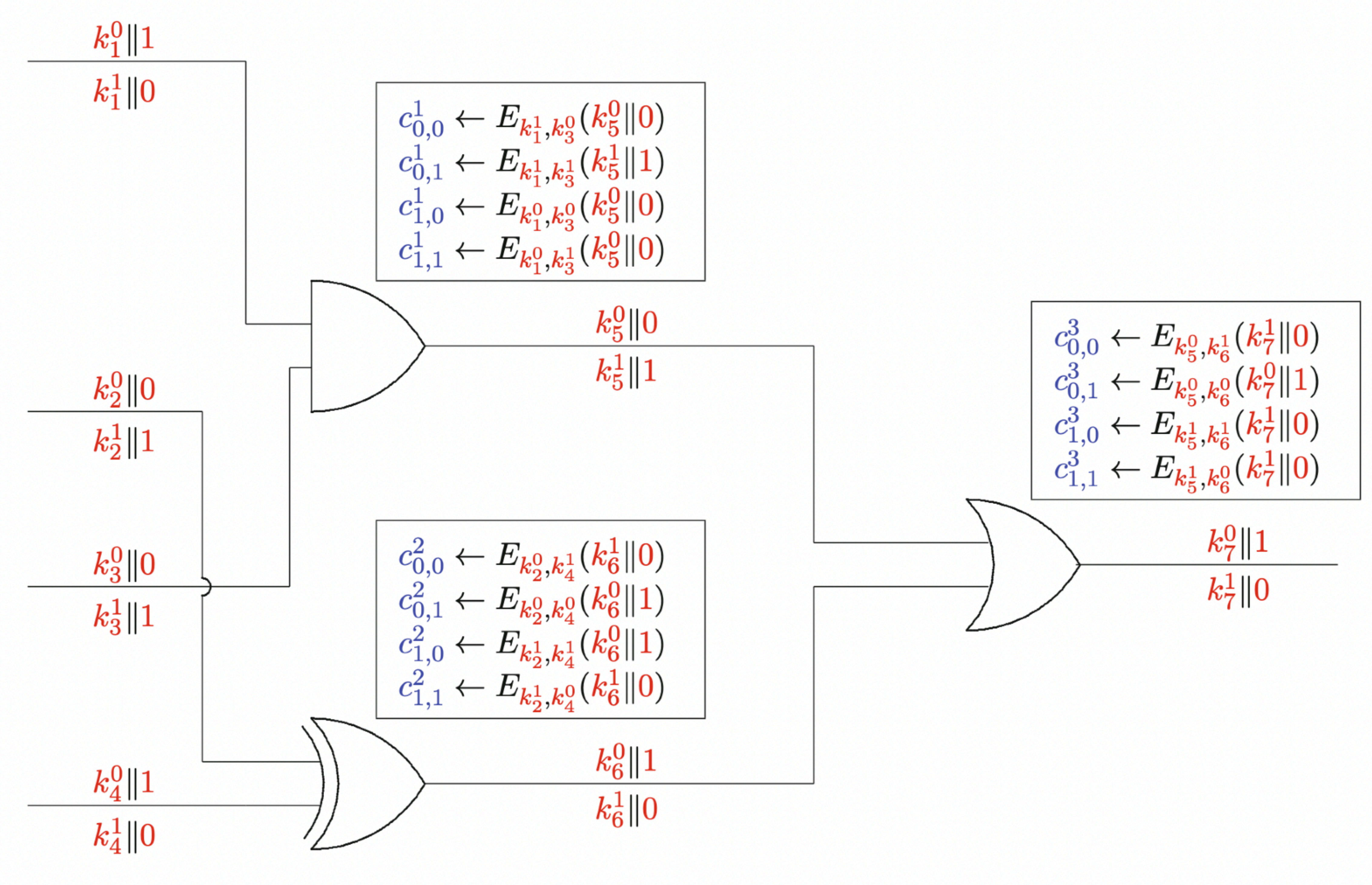
We can apply the lookup table technique to a sequence of logic gates, without blowing out the table size.

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.



$k_1^0\|1$
$k_1^1\|0$
$c_{0,0}^1 \leftarrow E_{k_1^1,k_3^0}(k_5^0\|0)$
$c_{0,1}^1 \leftarrow E_{k_1^1,k_3^1}(k_5^1\|1)$
$c_{1,0}^1 \leftarrow E_{k_1^0,k_3^0}(k_5^0\|0)$
$c_{1,1}^1 \leftarrow E_{k_1^0,k_3^1}(k_5^0\|0)$
$k_5^0\|0$
$k_5^1\|1$
$c_{0,0}^3 \leftarrow E_{k_5^0,k_6^1}(k_7^1\|0)$
$c_{0,1}^3 \leftarrow E_{k_5^0,k_6^0}(k_7^0\|1)$
$c_{1,0}^3 \leftarrow E_{k_5^1,k_6^1}(k_7^1\|0)$
$c_{1,1}^3 \leftarrow E_{k_5^1,k_6^0}(k_7^1\|0)$
$k_2^0\|0$
$k_2^1\|1$
$k_3^0\|0$
$k_3^1\|1$
$k_7^0\|1$
$k_7^1\|0$
$c_{0,0}^2 \leftarrow E_{k_2^0,k_4^1}(k_6^1\|0)$
$c_{0,1}^2 \leftarrow E_{k_2^0,k_4^0}(k_6^0\|1)$
$c_{1,0}^2 \leftarrow E_{k_2^1,k_4^1}(k_6^0\|1)$
$c_{1,1}^2 \leftarrow E_{k_2^1,k_4^0}(k_6^1\|0)$
$k_4^0\|1$
$k_4^1\|0$
$k_6^0\|1$
$k_6^1\|0$

FIGURE 22.2. A garbled circuit

Garbled Circuit (GC) technique

- In the GC protocol, the algorithm F is represented in the form of a logical circuit C, which consists of a set of logic gates $G = g_1, g_2, \ldots, g_m$, connected by wires $W = w_1, w_2, \ldots, w_n$. Each gate g_i is a function that takes values from two input wires and returns a value on the output wire. For example, let g_{OR} be an OR gate, with input wires w_1 and w_2, and an output wire w_3. The gate g_{OR} represents the function $w_3 \leftarrow g_{\mathrm{OR}}(w_1, w_2)$, such that

- $0 \leftarrow g_{\mathrm{OR}}(0,0)$, $1 \leftarrow g_{\mathrm{OR}}(0,1)$, $1 \leftarrow g_{\mathrm{OR}}(1,0)$, and $1 \leftarrow g_{\mathrm{OR}}(1,1)$.

Garbled Circuit (GC) technique

- The functionality F that we want to execute obliviously must be converted into a logical circuit C_F, or more specifically, a netlist, which is then garbled using the GC protocol.
- The transformation of functionality F into a netlist is a common method in digital circuit design. It can be done natively using languages such as Verilog or VHDL, or by using HLS (high-level synthesis) tools that allow synthesis from high-level languages such as C (e.g., SPARK, CBMC-GC) or Python (e.g., PandA). This process results in a hardware description of the logic circuit in an HDL (Hardware Description Language) format, which allows for the creation of a simplified netlist describing the components of the circuit and the connections between them.
- Afterwards, the netlist created in this way can be executed obliviously using the GC protocol.
- There are tools that facilitate these transformations. Some of them take an HDL hardware description of a logic circuit as input, such as: TinyGarble, ABY
- Others provide a full set of tools, from a programming language to a runtime environment, such as: MP-SPDZ, MPyC, EzPC, EMP-Toolkit
- More https://github.com/MPC-SoK/frameworks

Neural network inference is all about matrix multiplications and activation functions

- We know how to convert any algorithm into an unconscious version using the GC protocol. We also know how inference is performed using neural networks. What remains is to combine the knowledge to create an unconscious neural network prediction.

- Prediction using a neural network can be modeled as a recursive sequence of matrix multiplication and addition, along with activation/pooling functions:

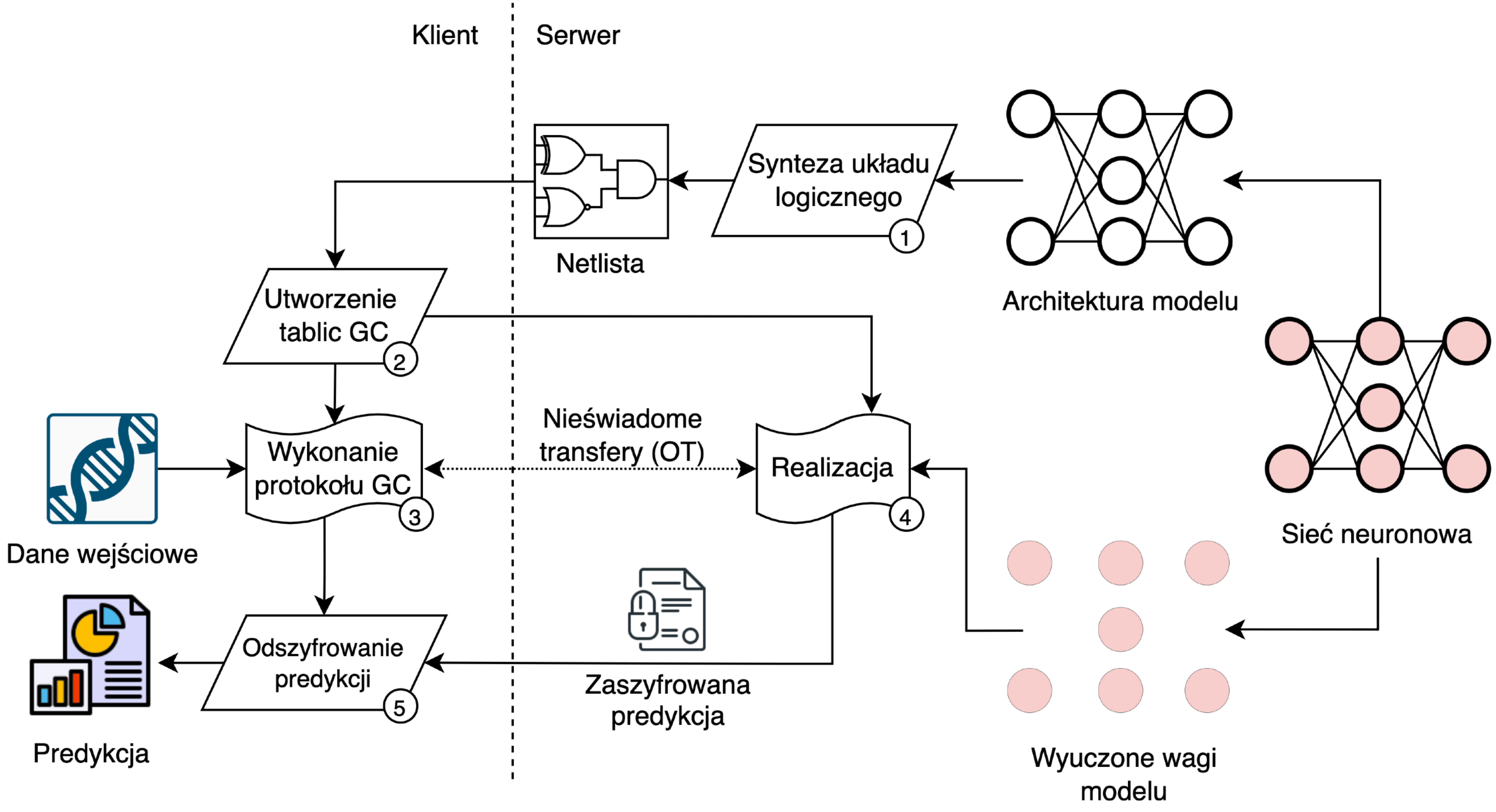
- $$z := W_L * f_{L-1}(\ldots f_1(W_1 * X + B_1)\ldots) + B_L$$

- The goal of unconscious neural networks is to compute the value z in such a way that the client learns nothing about $(W_1, W_2, \ldots, W_L)$ and $(B_1, B_2, \ldots, B_L)$, while the server learns nothing about X and z.

- To achieve such isolation, we substitute the inputs of each party into the functional model $F_{2PC}(a, b)$, where a is the private input of one party and b is the private input of the other party. In our case, $a = X$ and $b = ((W_1, W_2, \ldots, W_L), (B_1, B_2, \ldots, B_L))$. As a result, we have:

- $F_{2PC}(X, ((W_1, W_2, \ldots, W_L), (B_1, B_2, \ldots, B_L))) = W_L * f_{L-1}(\ldots f_1(W_1 * X + B_1)\ldots) + B_L$

Oblivious Neural Network pipeline



source: https://stan.bar/papers/ONN-KASKBook2022.pdf

Performance

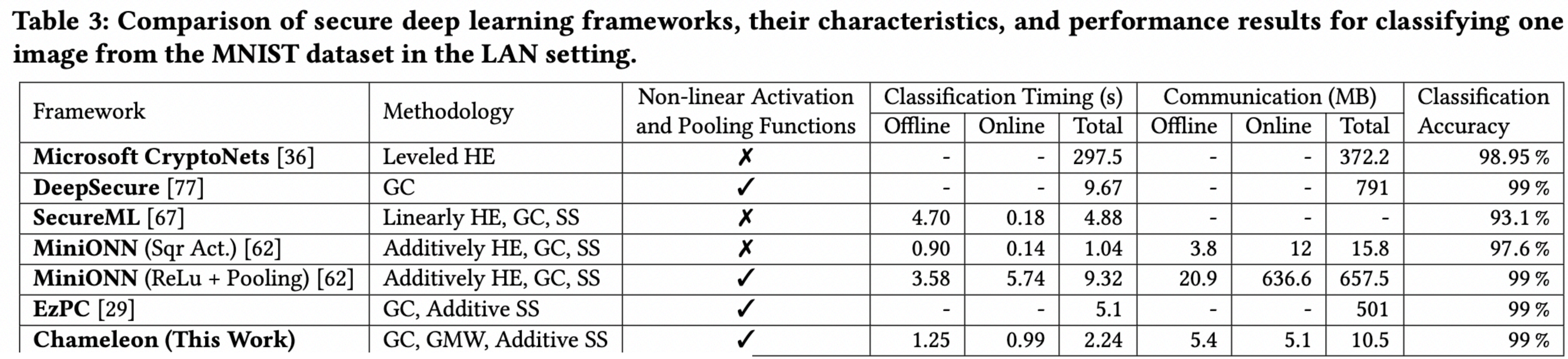


Table 3: Comparison of secure deep learning frameworks, their characteristics, and performance results for classifying one image from the MNIST dataset in the LAN setting.

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 %

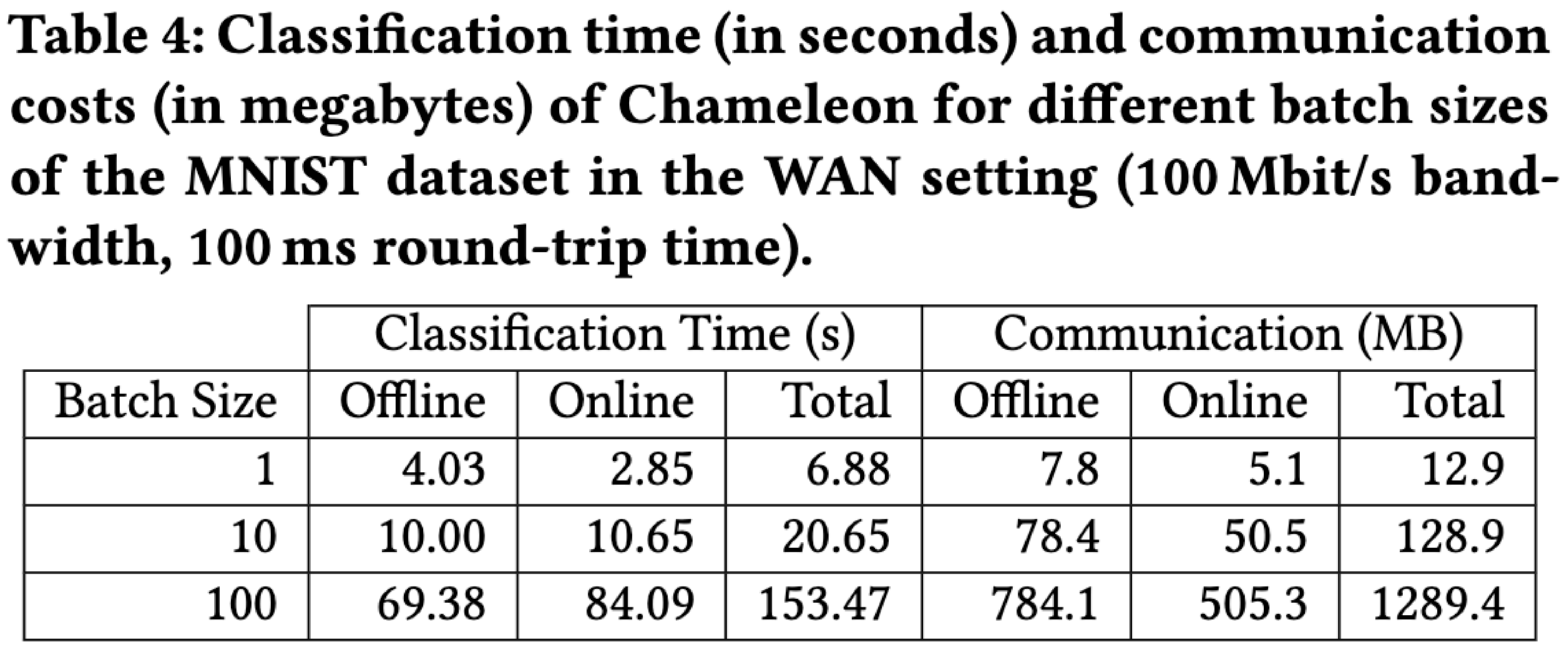


Table 4: Classification time (in seconds) and communication costs (in megabytes) of Chameleon for different batch sizes of the MNIST dataset in the WAN setting (100 Mbit/s bandwidth, 100 ms round-trip time).

	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

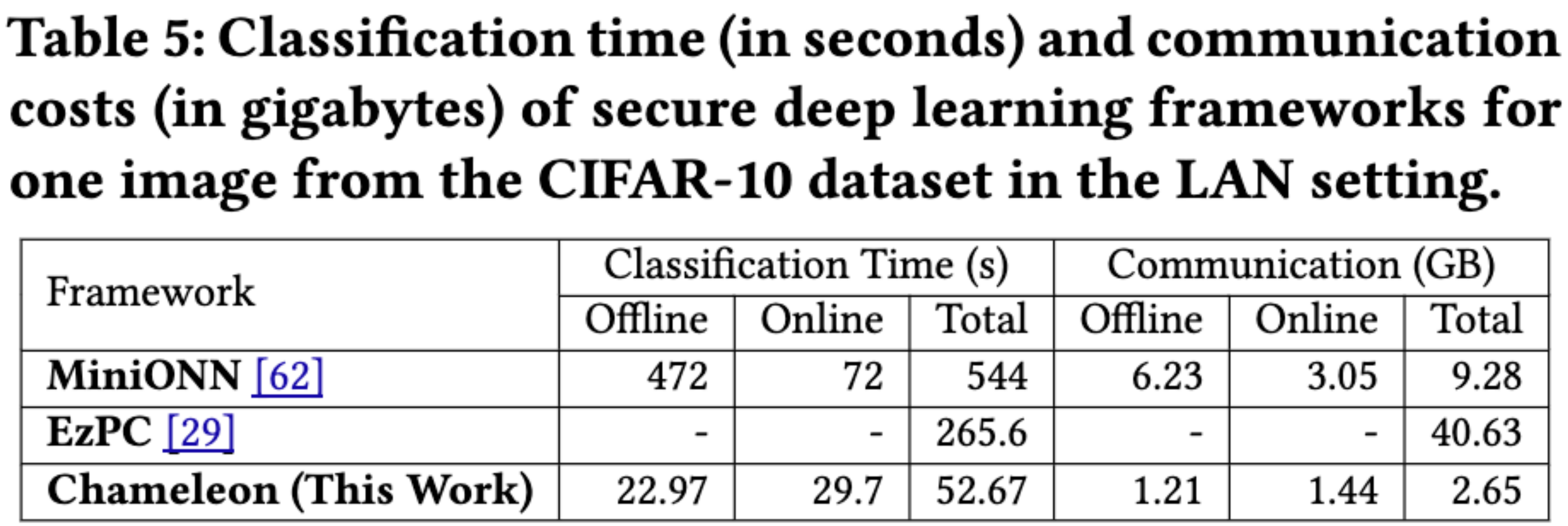


Table 5: Classification time (in seconds) and communication costs (in gigabytes) of secure deep learning frameworks for one image from the CIFAR-10 dataset in the LAN setting.

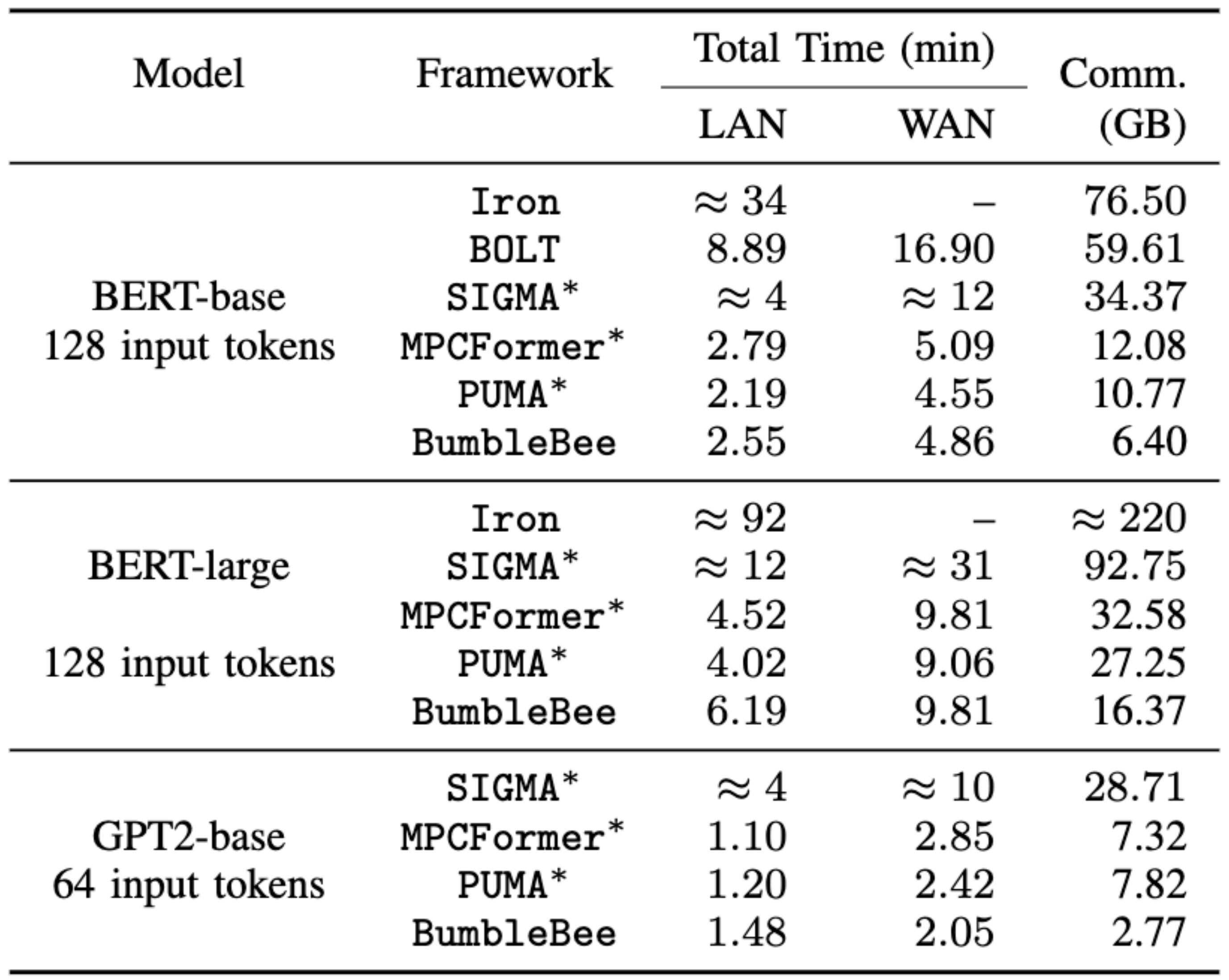
Framework	Classification Time (s)			Communication (GB)		
	Offline	Online	Total	Offline	Online	Total
MiniONN [62]	472	72	544	6.23	3.05	9.28
EzPC [29]	-	-	265.6	-	-	40.63
Chameleon (This Work)	22.97	29.7	52.67	1.21	1.44	2.65

Source: https://dl.acm.org/doi/pdf/10.1145/3196494.3196522#page=10

LLM

- Privacy-Preserving Large Language Models (PPLLMs)
- PermLLM: Private Inference of Large Language Models within 3 Seconds under WAN
- Faster Lookup Table Evaluation with Application to Secure LLM Inference

TABLE V: End-to-end comparisons with the existing private inference frameworks. The numbers of `Iron` and `CipherGPT` are taken from their papers. The timings of `SIGMA` include both the key-transmission and online inference, which are estimated based on our bandwidth. GPT2 models generated 1 token. Frameworks marked with “*” **are not** 2PC framework.



Model	Framework	Total Time (min)		Comm.
		LAN	WAN	(GB)
BERT-base 128 input tokens	Iron	≈ 34	–	76.50
	BOLT	8.89	16.90	59.61
	SIGMA*	≈ 4	≈ 12	34.37
	MPCFormer*	2.79	5.09	12.08
	PUMA*	2.19	4.55	10.77
	BumbleBee	2.55	4.86	6.40
BERT-large 128 input tokens	Iron	≈ 92	–	≈ 220
	SIGMA*	≈ 12	≈ 31	92.75
	MPCFormer*	4.52	9.81	32.58
	PUMA*	4.02	9.06	27.25
	BumbleBee	6.19	9.81	16.37
GPT2-base 64 input tokens	SIGMA*	≈ 4	≈ 10	28.71
	MPCFormer*	1.10	2.85	7.32
	PUMA*	1.20	2.42	7.82
	BumbleBee	1.48	2.05	2.77

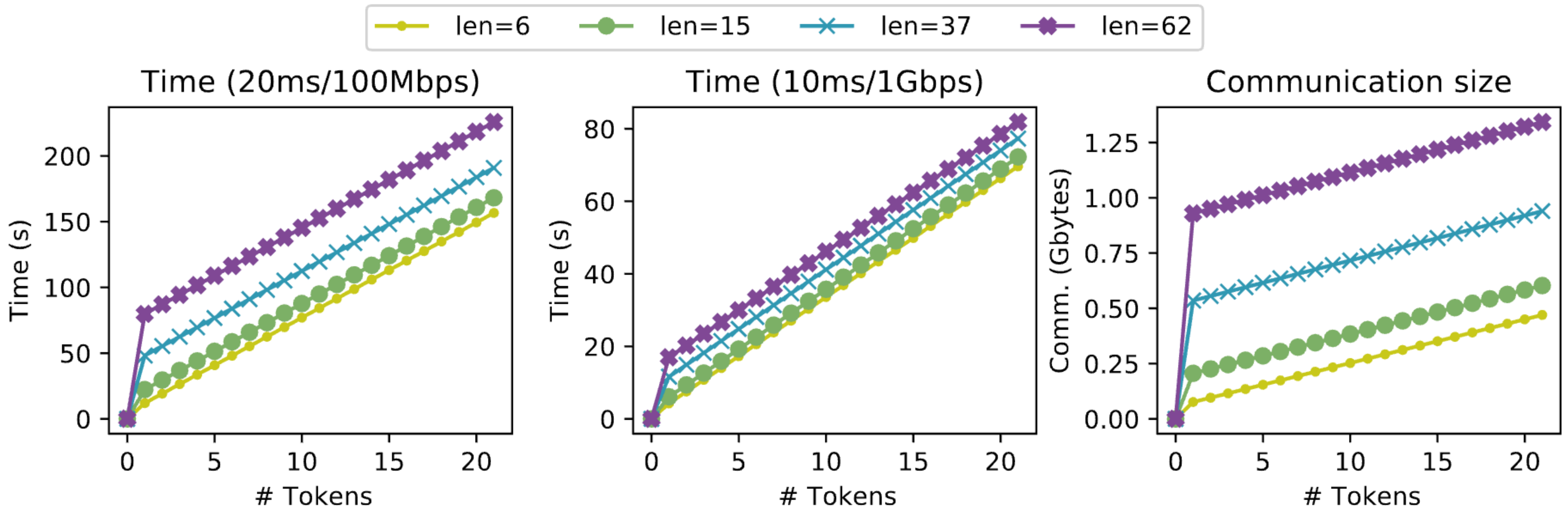
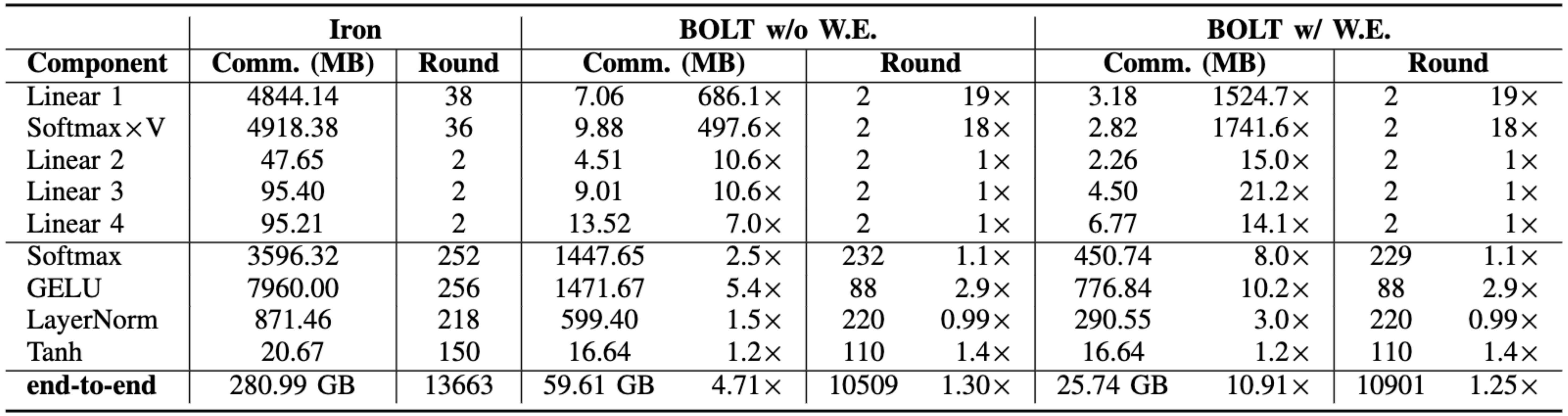


Figure 2: Time consumption and communication size of the ChatGLM-6B private inference.

State of the art

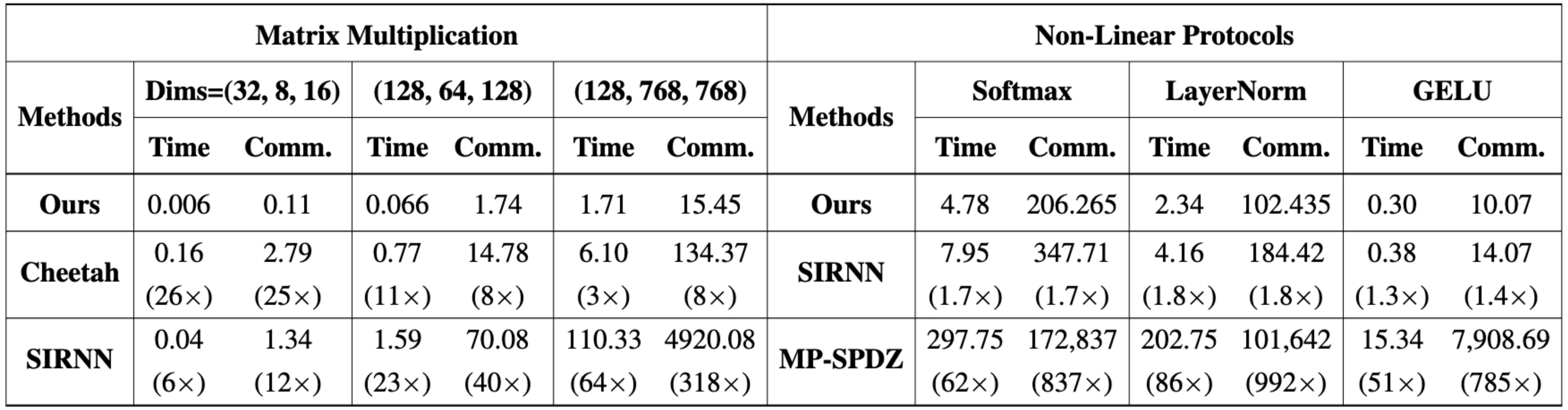
- SIRNN (Microsoft Research)
- Cheetah (Alibaba Group)
- CipherGPT (Zhejiang University, Ant Group)
- BumbleBee (Ant Group, Alibaba Group, Zhejiang University)
- CraterLake (Massachusetts Institute of Technology)
- Iron (University of Electronic Science and Technology of China, Nanyang Technological University)
- BOLT (Carnegie Mellon University UC Berkeley, Technical University of Darmstadt)



	Iron		BOLT w/o W.E.				BOLT w/ W.E.			
Component	**Comm. (MB)**	**Round**	**Comm. (MB)**		**Round**		**Comm. (MB)**		**Round**	
Linear 1	4844.14	38	7.06	686.1×	2	19×	3.18	1524.7×	2	19×
Softmax×V	4918.38	36	9.88	497.6×	2	18×	2.82	1741.6×	2	18×
Linear 2	47.65	2	4.51	10.6×	2	1×	2.26	15.0×	2	1×
Linear 3	95.40	2	9.01	10.6×	2	1×	4.50	21.2×	2	1×
Linear 4	95.21	2	13.52	7.0×	2	1×	6.77	14.1×	2	1×
Softmax	3596.32	252	1447.65	2.5×	232	1.1×	450.74	8.0×	229	1.1×
GELU	7960.00	256	1471.67	5.4×	88	2.9×	776.84	10.2×	88	2.9×
LayerNorm	871.46	218	599.40	1.5×	220	0.99×	290.55	3.0×	220	0.99×
Tanh	20.67	150	16.64	1.2×	110	1.4×	16.64	1.2×	110	1.4×
end-to-end	280.99 GB	13663	59.61 GB	4.71×	10509	1.30×	25.74 GB	10.91×	10901	1.25×

TABLE 3: Communication cost and rounds comparing Iron with BOLT. The costs of the components are for one layer, and there are 12 layers in BERT.

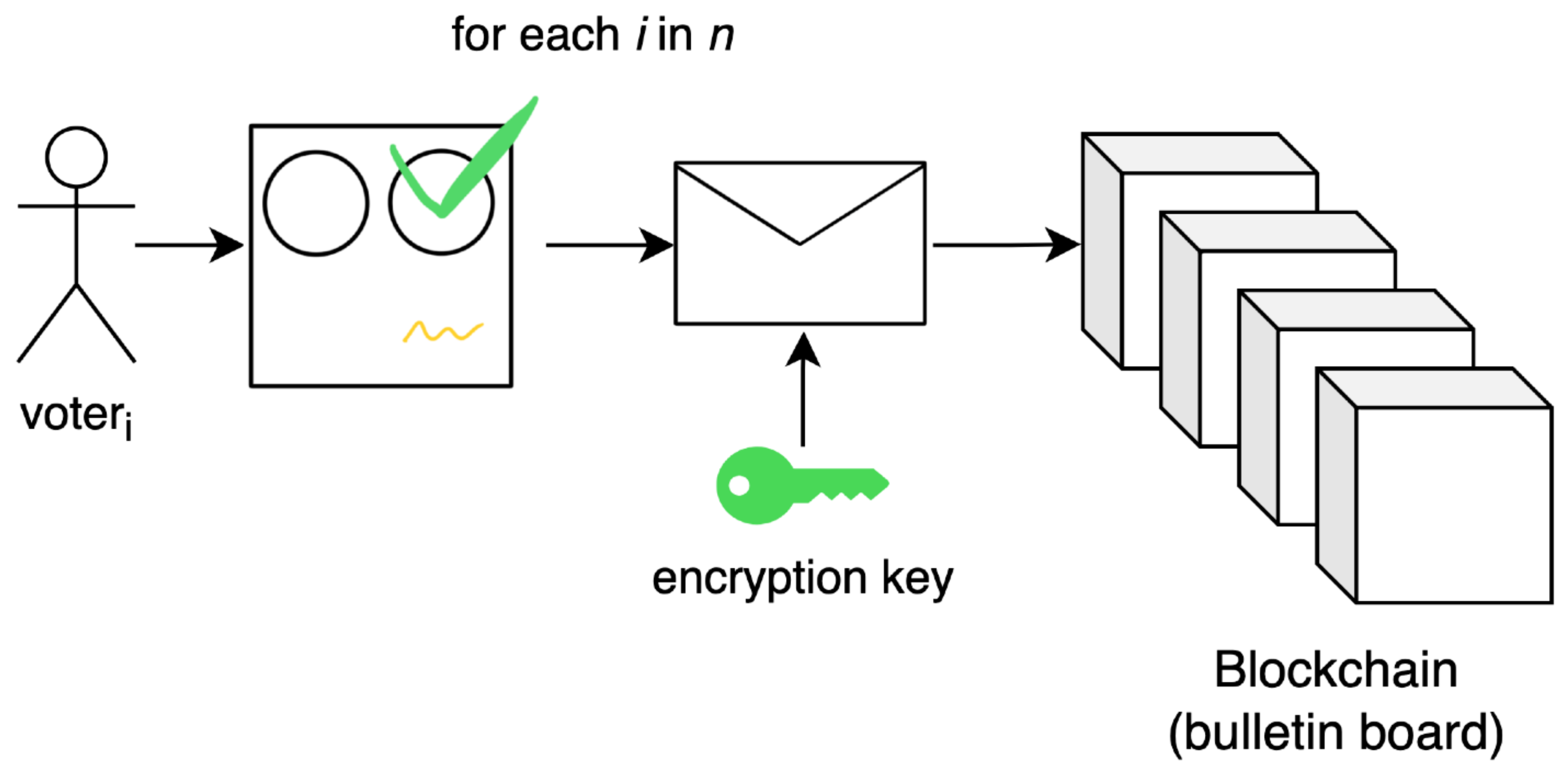
Table 1: Comparing the runtime (sec) and communication (MB) costs of our matrix multiplication and non-linear protocols with SOTA



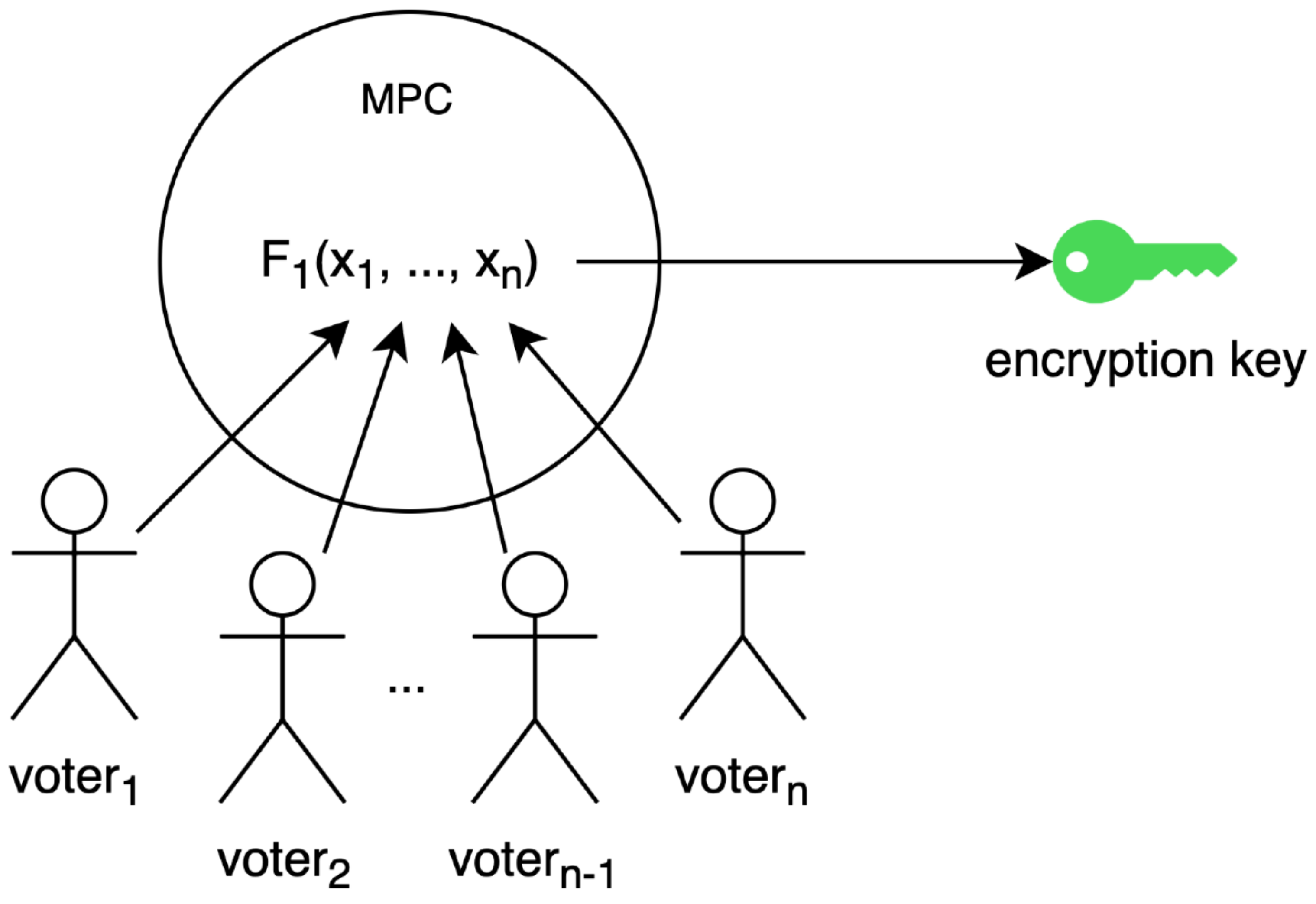
Matrix Multiplication							Non-Linear Protocols						
Methods	**Dims=(32, 8, 16)**		**(128, 64, 128)**		**(128, 768, 768)**		**Methods**	**Softmax**		**LayerNorm**		**GELU**	
	Time	**Comm.**	**Time**	**Comm.**	**Time**	**Comm.**		**Time**	**Comm.**	**Time**	**Comm.**	**Time**	**Comm.**
Ours	0.006	0.11	0.066	1.74	1.71	15.45	**Ours**	4.78	206.265	2.34	102.435	0.30	10.07
Cheetah	0.16 (26×)	2.79 (25×)	0.77 (11×)	14.78 (8×)	6.10 (3×)	134.37 (8×)	**SIRNN**	7.95 (1.7×)	347.71 (1.7×)	4.16 (1.8×)	184.42 (1.8×)	0.38 (1.3×)	14.07 (1.4×)
SIRNN	0.04 (6×)	1.34 (12×)	1.59 (23×)	70.08 (40×)	110.33 (64×)	4920.08 (318×)	**MP-SPDZ**	297.75 (62×)	172,837 (837×)	202.75 (86×)	101,642 (992×)	15.34 (51×)	7,908.69 (785×)

My Work

Voting

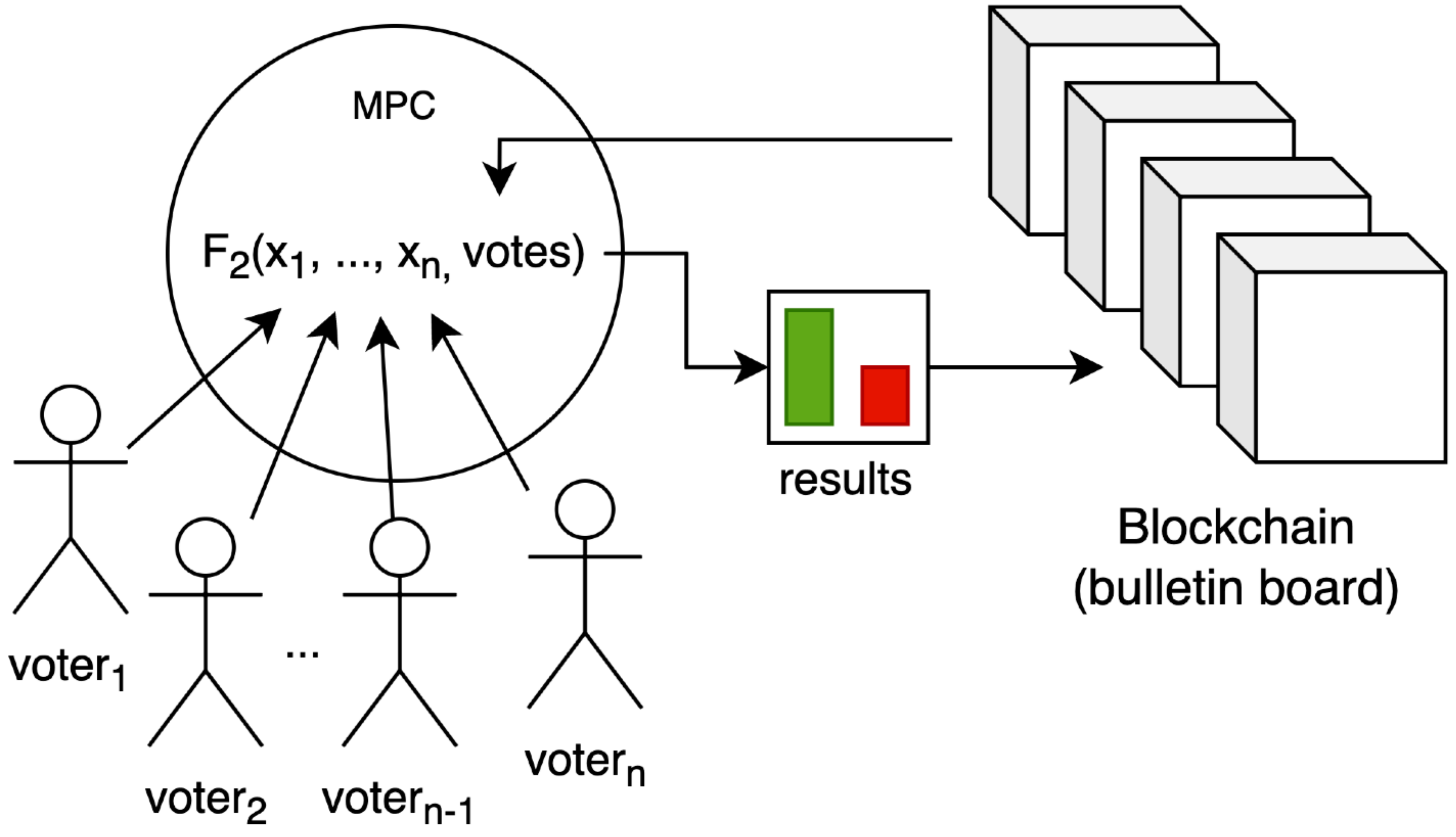


Setup



$F_1(x_1, \ldots, x_n) = \texttt{DerivePubKey}(\texttt{DerivePrivKey}(\texttt{SS}(x_1, \ldots, x_n)))$

Tally



$F_2(x_1, \ldots, x_n, votes) = \texttt{Count}(\texttt{Decrypt}(votes, \texttt{DerivePrivKey}(\texttt{SS}(x_1, \ldots, x_n))))$

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.
- Mann, Zoltán Ádám, et al. "Towards practical secure neural network inference: the journey so far and the road ahead." ACM Computing Surveys 56.5 (2023): 1-37.
- 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.
- Lu, Wen-jie, et al. "Bumblebee: Secure two-party inference framework for large transformers." Cryptology ePrint Archive (2023).
- Hou, Xiaoyang, et al. "Ciphergpt: Secure two-party gpt inference." Cryptology ePrint Archive (2023).