Aalto University

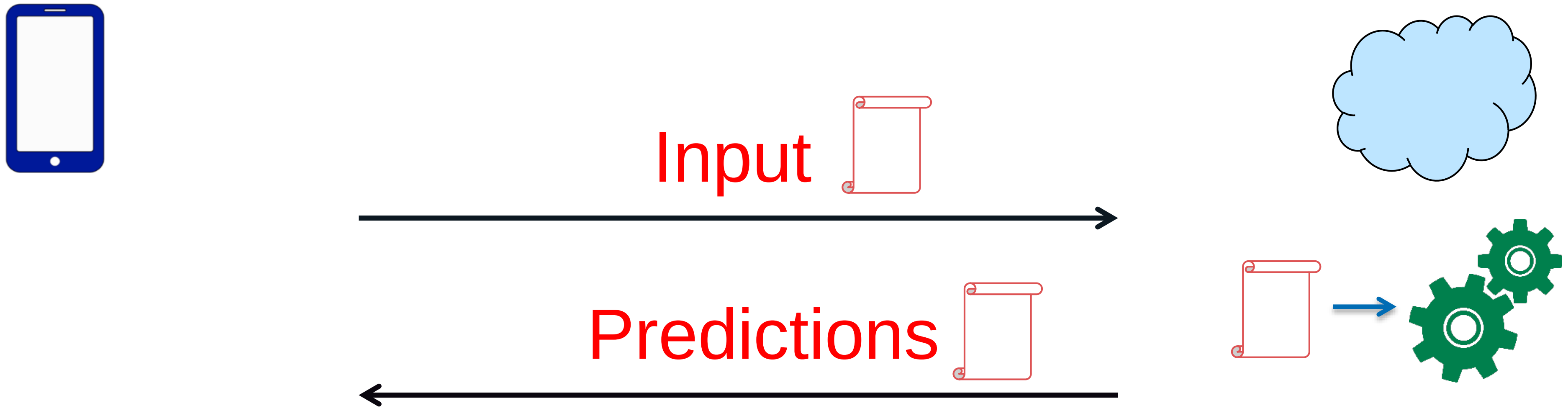
Oblivious Neural Network Predictions via MiniONN Transformations

N. Asokan, https://asokan.org/asokan/, @nasokan

(Joint work with Jian Liu, Mika Juuti, Yao Lu)

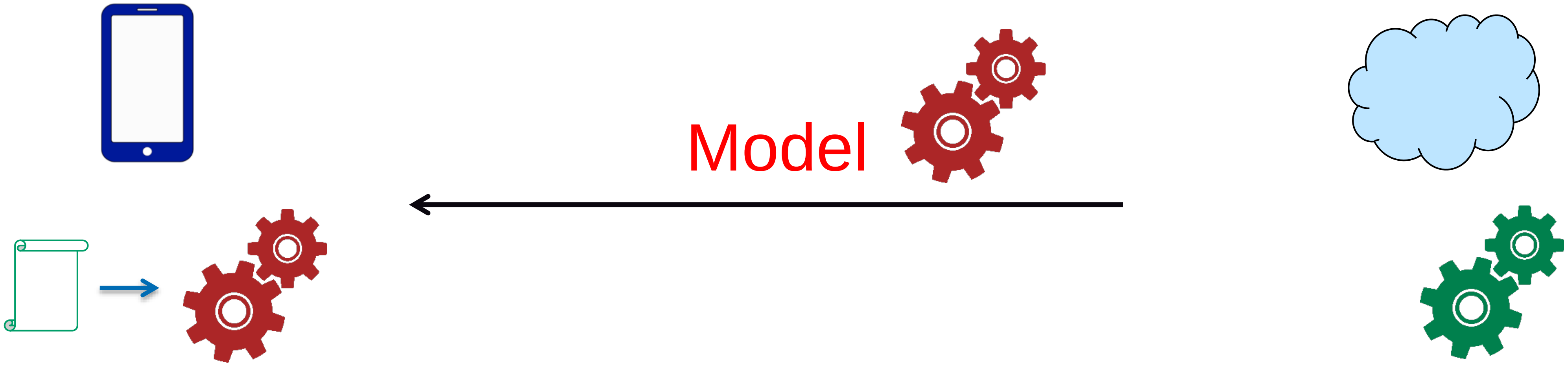
By Source, Fair use, https://en.wikipedia.org/w/index.php?curid=54119040

Machine learning as a service (MLaaS)



violation of clients’ privacy

Running predictions on client-side



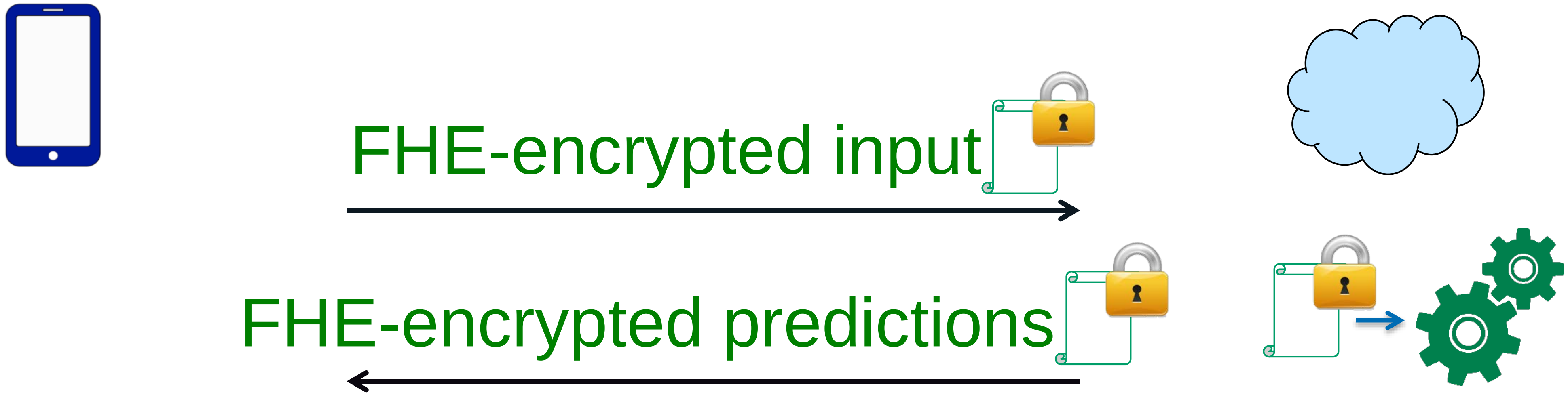
model theft
evasion
model inversion

Oblivious Neural Networks (ONN)

Given a neural network, is it possible to make it oblivious?

- server learns nothing about clients' input;
- clients learn nothing about the model.

Example: CryptoNets

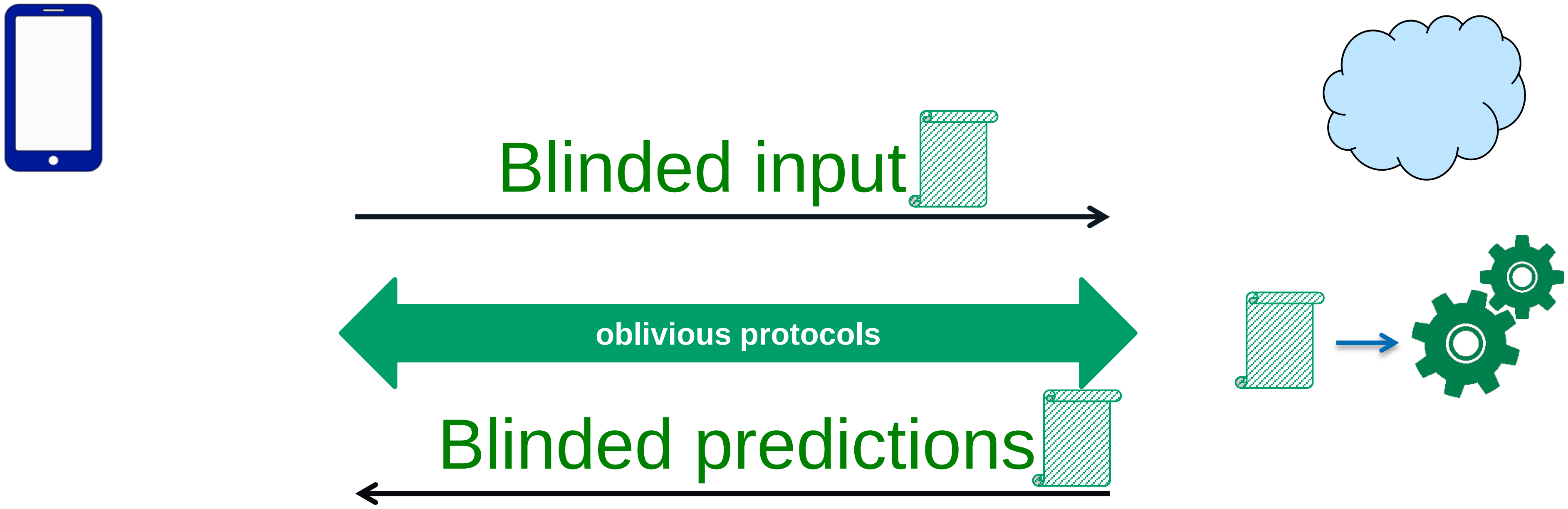


- High throughput for batch queries from same client
- High overhead for single queries: 297.5s and 372MB (MNIST dataset)
- Cannot support: high-degree polynomials, comparisons, …

[GDLLNW16] CryptoNets, ICML 2016

MiniONN: Overview

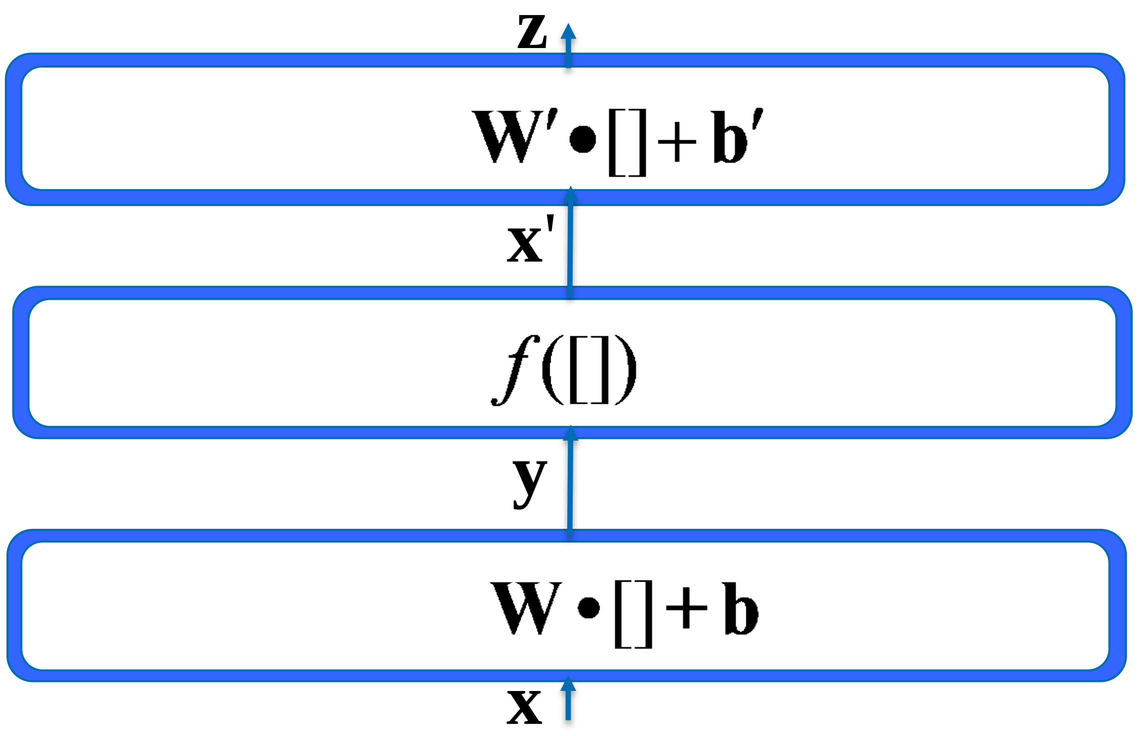
By Source, Fair use, https://en.wikipedia.org/w/index.php?curid=54119040



- Low overhead: ~1s
- Support all common neural networks

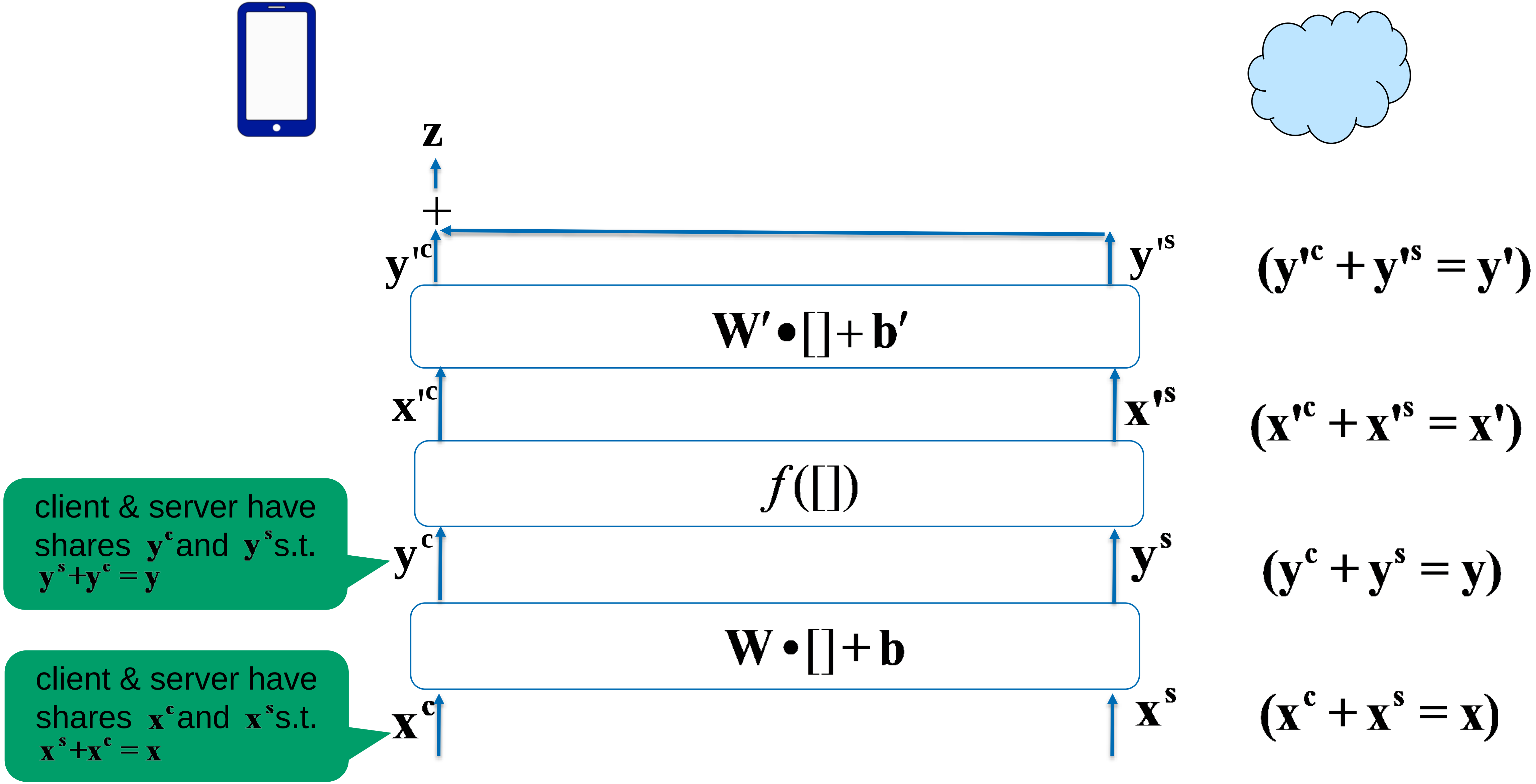
Example $\mathbf{z} = \mathbf{W}' \bullet f(\mathbf{W} \bullet \mathbf{x} + \mathbf{b}) + \mathbf{b}'$

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

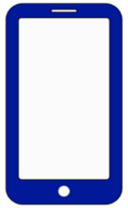


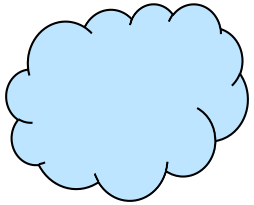
All operations are in a finite field Z_N

Core idea: use secret sharing for oblivious computation



Secret sharing initial input $\mathbf{x}$





$$x_1^c, x_2^c \xleftarrow{\$} Z_N$$

$$x_1^s := x_1 - x_1^c, \quad x_2^s := x_2 - x_2^c$$

Note that $\mathbf{x}^c$ is independent of $\mathbf{x}$. Can be **pre-chosen**

Oblivious linear transformation $\mathbf{W} \bullet \mathbf{x} + \mathbf{b}$

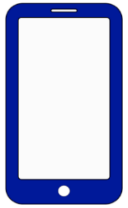
$$= \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix} \bullet \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix} \bullet \begin{bmatrix} x_1^s + x_1^c \\ x_2^s + x_2^c \end{bmatrix}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

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

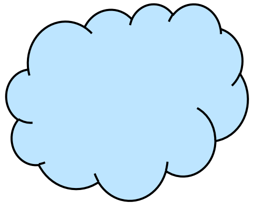
Compute locally by the server

Dot-product

Oblivious linear transformation: dot-product



Homomorphic Encryption with SIMD



$$r_{1,1}, r_{1,2}, r_{2,1}, r_{2,2} \xleftarrow{\$} Z_N \qquad \xleftarrow{\quad E(w_{1,1}), E(w_{1,2}), E(w_{2,1}), E(w_{2,2}) \quad}$$

$$c_{1,1} = E(w_{1,1}x_1^c - r_{1,1})$$

$$c_{1,2} = E(w_{1,2}x_2^c - r_{1,2}) \qquad \xrightarrow{\quad c_{1,1}, c_{1,2}, c_{2,1}, c_{2,2} \quad}$$

$$c_{2,1} = E(w_{2,1}x_1^c - r_{2,1}) \qquad D(c_{1,1}), D(c_{1,2}), D(c_{2,1}), D(c_{2,2})$$

$$c_{2,2} = E(w_{2,2}x_2^c - r_{2,2}) \qquad v_1 = r_{1,1} + r_{1,2} \qquad u_1 = w_{1,1}x_1^c + w_{1,2}x_2^c - (r_{1,2} + r_{1,1})$$

$$v_2 = r_{2,1} + r_{2,2} \qquad u_2 = w_{2,1}x_1^c + w_{2,2}x_2^c - (r_{2,1} + r_{2,2})$$

u + v = W•x^c; Note: **u, v,** and **W•x^c** are independent of **x**.
<**u,v,x^c** > generated/stored in a **precomputation phase**

Oblivious linear transformation $\mathbf{W} \bullet \mathbf{x} + \mathbf{b}$

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

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

$$= \begin{bmatrix} w_{1,1}x_1^s + w_{1,2}x_2^s + b_1 + u_1 \\ w_{2,1}x_1^s + w_{2,2}x_2^s + b_2 + u_2 \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

Oblivious linear transformation $\mathbf{W} \bullet \mathbf{x} + \mathbf{b}$

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

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

$$= \begin{bmatrix} w_{1,1}x_1^s + w_{1,2}x_2^s + b_1 + u_1 \\ w_{2,1}x_1^s + w_{2,2}x_2^s + b_2 + u_2 \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} := \begin{bmatrix} y_1^s \\ y_2^s \end{bmatrix} + \begin{bmatrix} y_1^c \\ y_2^c \end{bmatrix}$$

Oblivious activation/pooling functions $f(y)$

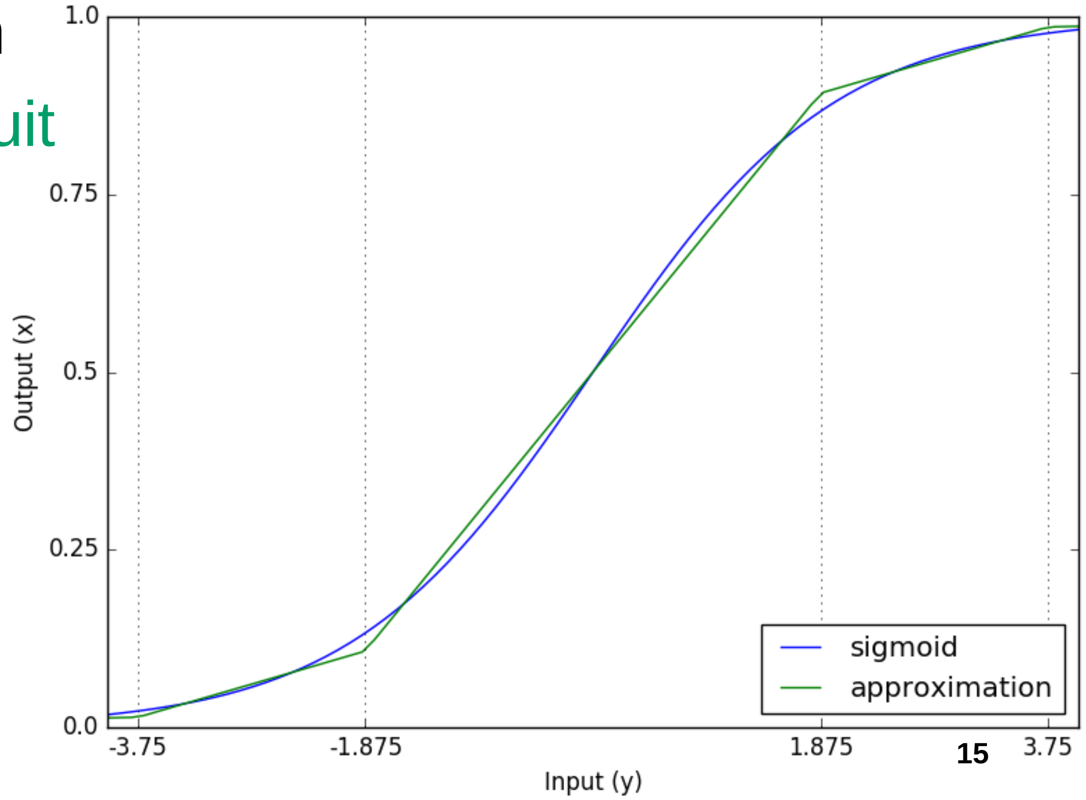
Piecewise linear functions e.g.,

- ReLU: $x := \max(y, 0)$
- Oblivious ReLU: $x^s + x^c := \max(y^s + y^c, 0)$
  - easily computed obliviously by a garbled circuit

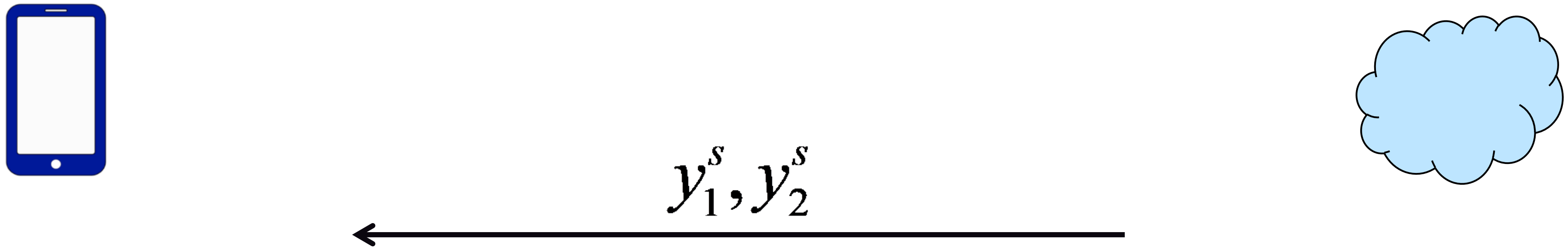
Oblivious activation/pooling functions $f(y)$

Smooth functions e.g.,

- Sigmoid: $x := 1/(1+e^{-y})$
- Oblivious sigmoid: $x^s + x^c := 1/(1+e^{-(y^s+y^c)})$
  - approximate by a piecewise linear function
  - then compute obliviously by a garbled circuit
  - empirically: ~14 segments sufficient



Combining the final result

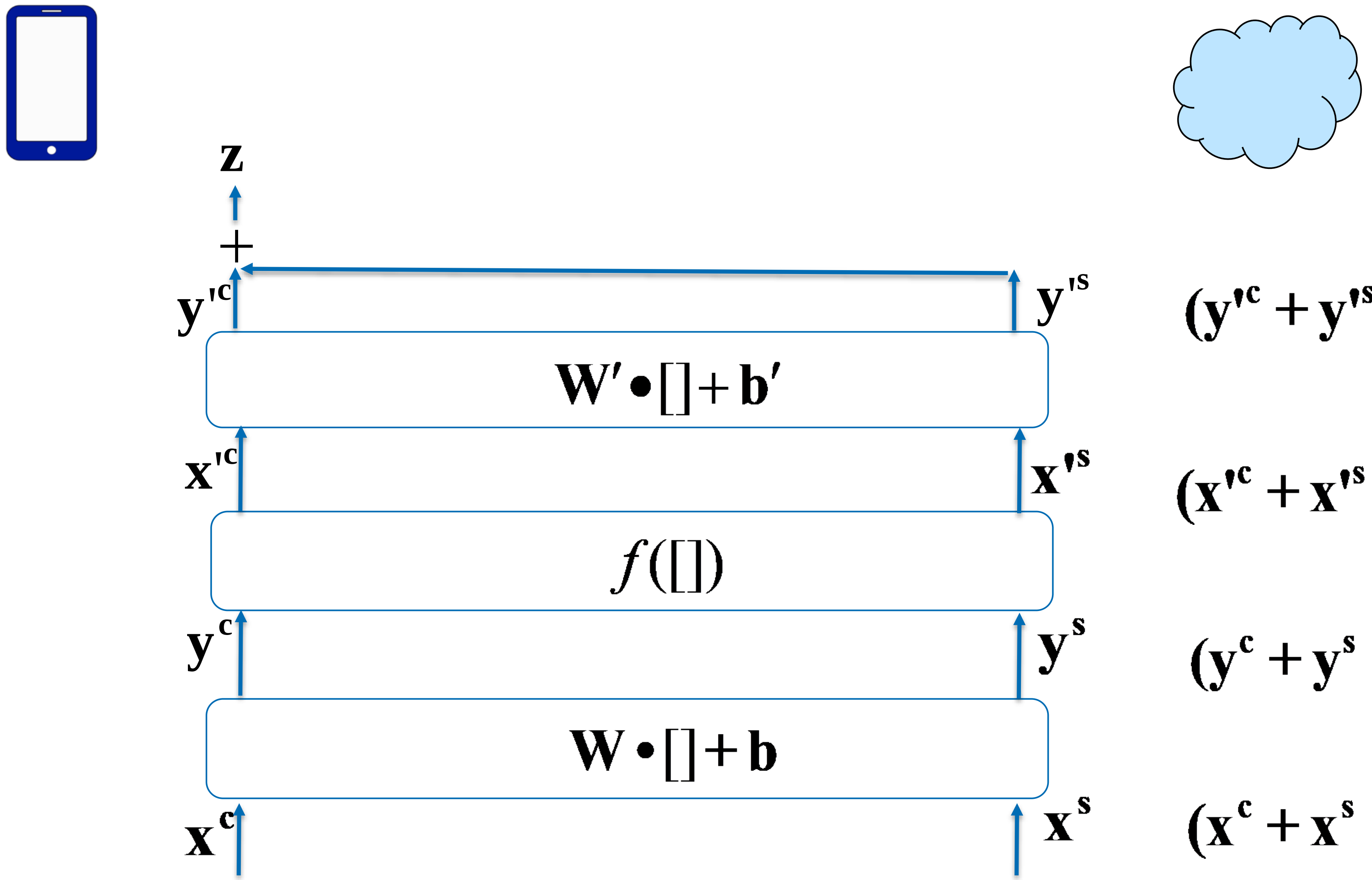


$$y_1 := y_1^s + y_1^c$$

$$y_2 := y_2^s + y_2^c$$

They can jointly calculate max(y_1,y_2)
(for minimizing information leakage)

Core idea: use secret sharing for oblivious computation



$(y'^c + y'^s = y')$

$(x'^c + x'^s = x')$

$(y^c + y^s = y)$

$(x^c + x^s = x)$

Performance (for single queries)

Model	Latency (s)	Msg sizes (MB)	Loss of accuracy
MNIST/Square	0.4 (+ 0.88)	44 (+ 3.6)	none
CIFAR-10/ReLU	472 (+ 72)	6226 (+ 3046)	none
PTB/Sigmoid	4.39 (+ 13.9)	474 (+ 86.7)	Less than 0.5% (cross-entropy loss)

Pre-computation phase timings in parentheses

PTB = Penn Treebank

MiniONN pros and cons

300-700x faster than CryptoNets

Can transform any given neural network to its oblivious variant

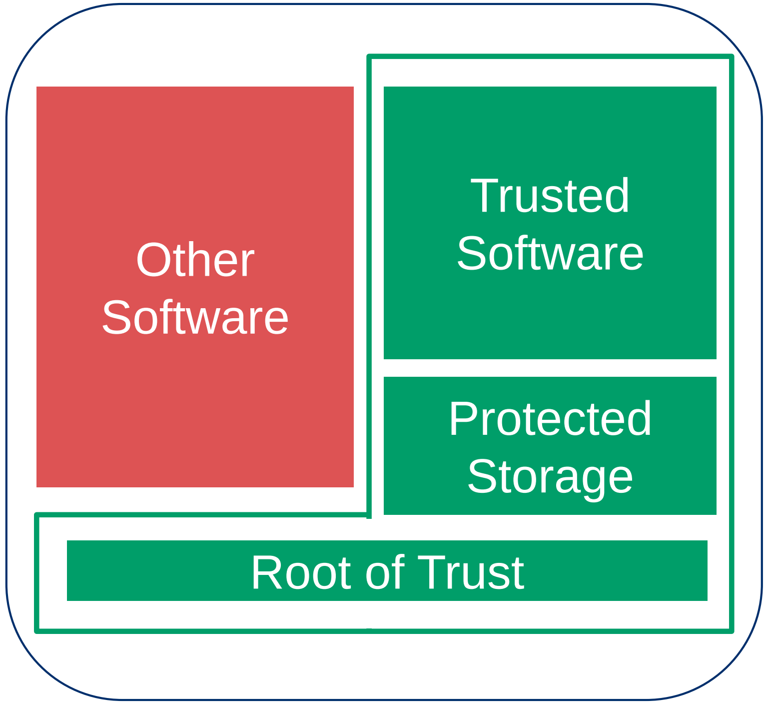
Still ~1000x slower than without privacy

Server can no longer filter requests or do sophisticated metering

Assumes online connectivity to server

Reveals structure (but not params) of NN

Can trusted computing help?



Hardware support for

- **Isolated execution: Trusted Execution Environment**
- **Protected storage: Sealing**
- **Ability to report status to a remote verifier: Attestation**

Cryptocards

https://www.ibm.com/security/cryptocards/

Trusted Platform Modules

https://www.infineon.com/tpm

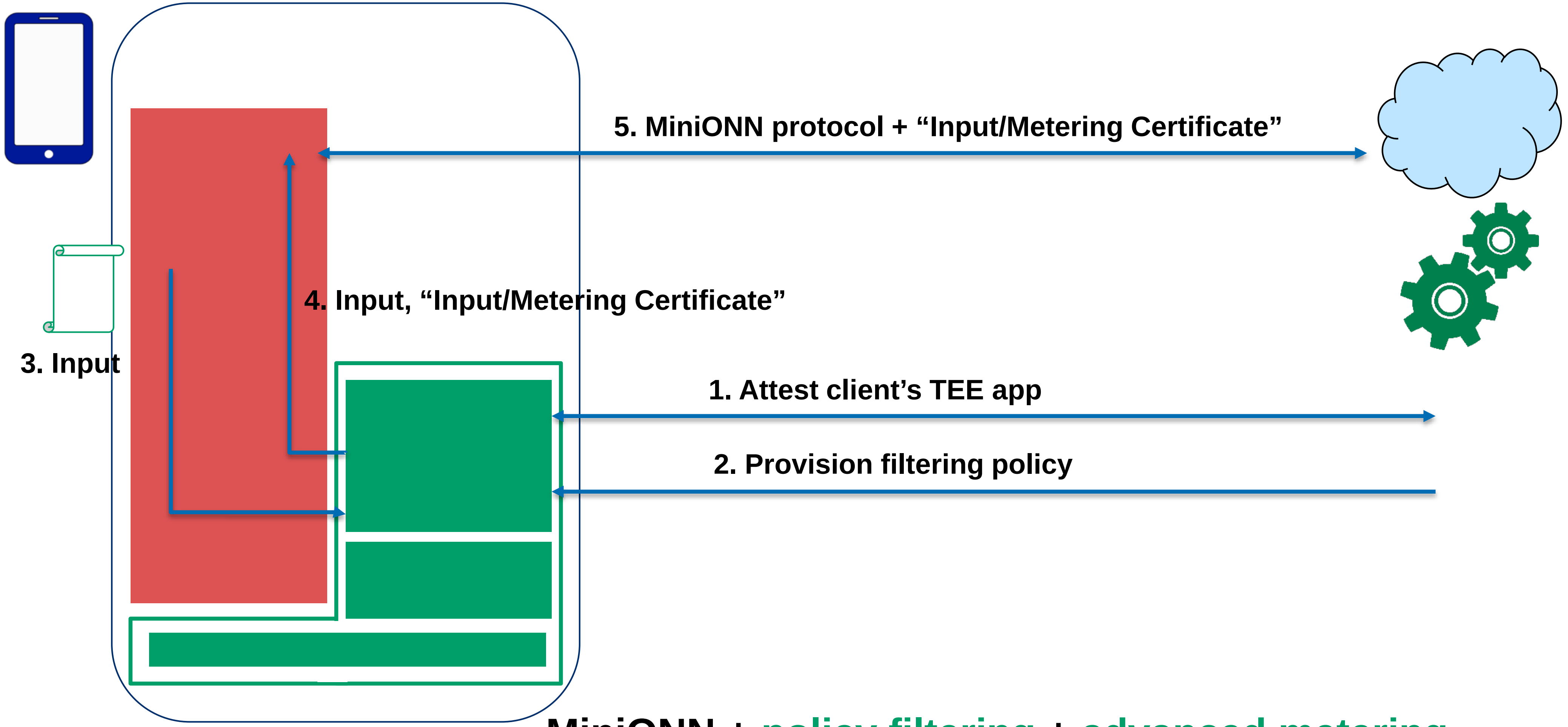
ARM TrustZone

https://www.arm.com/products/security-on-arm/trustzone

Intel Software Guard Extensions

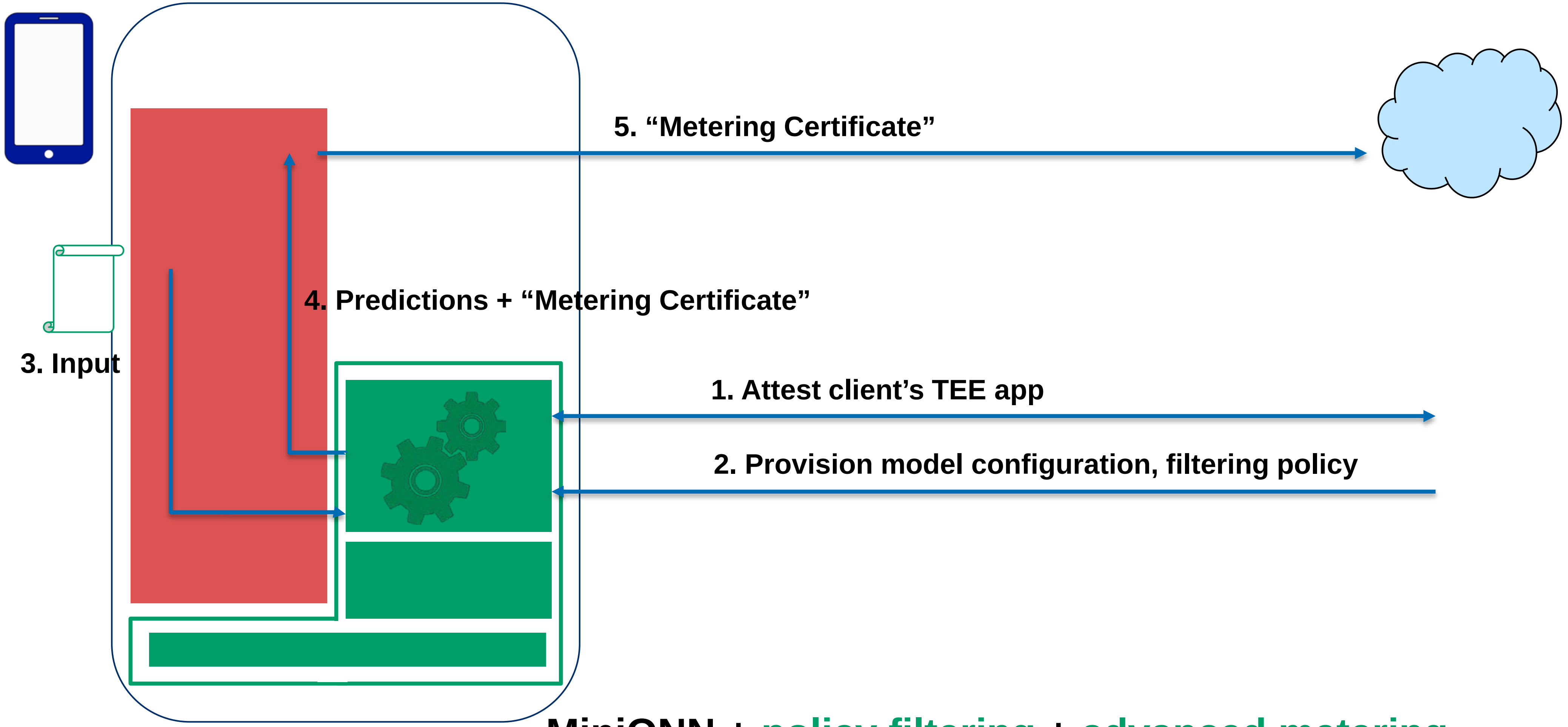
https://software.intel.com/en-us/sgx

Using a client-side TEE to vet input



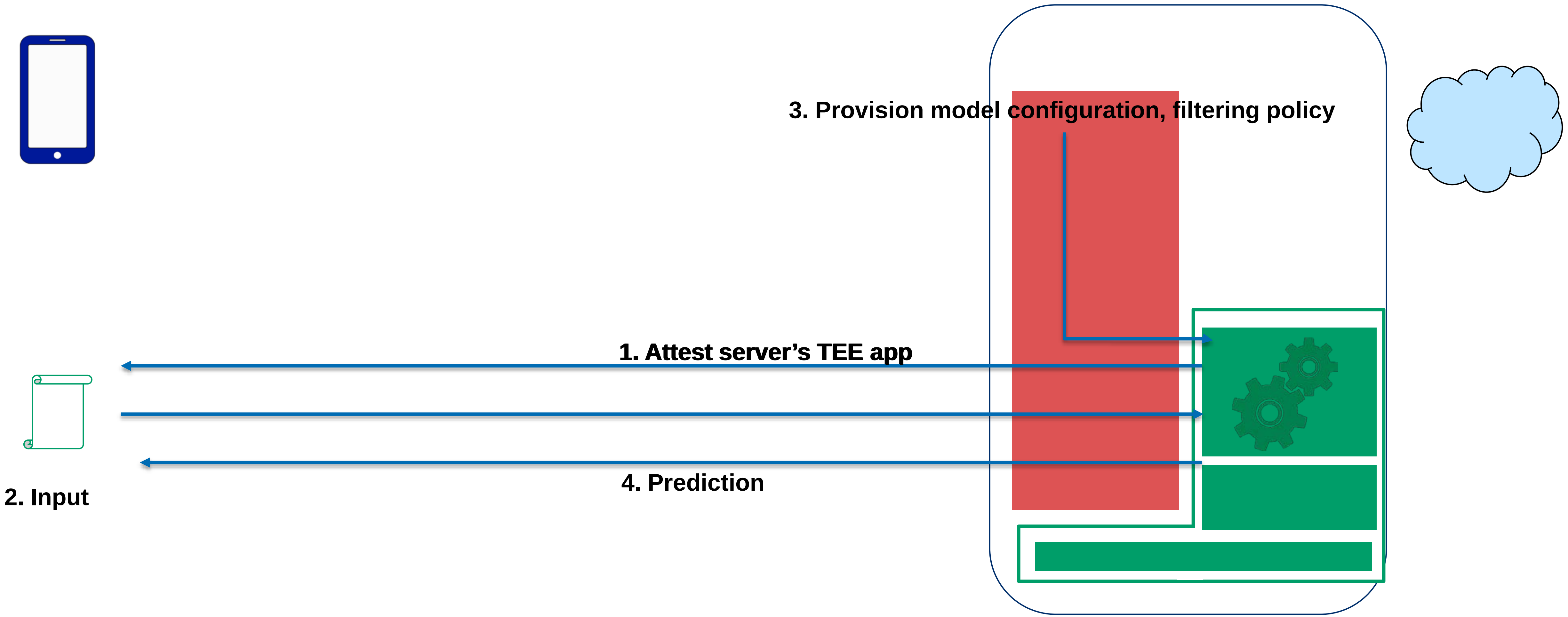
MiniONN + policy filtering + advanced metering

Using a client-side TEE to run the model



MiniONN + policy filtering + advanced metering
+ disconnected operation + performance + better privacy
- harder to reason about model secrecy

Using a server-side TEE to run the model



MiniONN + policy filtering + advanced metering
- disconnected operation + performance + better privacy

MiniONN: Efficiently transform any given neural network into oblivious form with no/negligible accuracy loss

Trusted Computing can help realize improved security and privacy for ML

ML is very fragile in adversarial settings

https://eprint.iacr.org/2017/452
CCS 2017