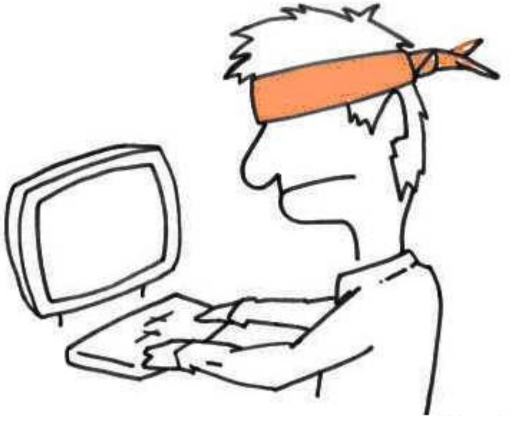
Blind Machine Learning



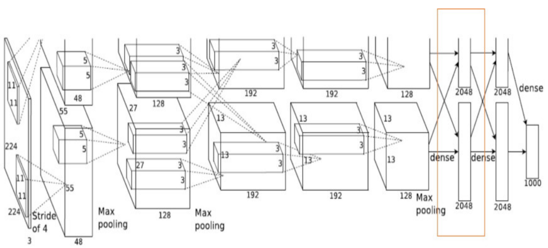
Vinod Vaikuntanathan

MIT

Joint work with Chiraag Juvekar
and Anantha Chandrakasan

Problem 1. Blind Inference
(application: Monetizing ML)

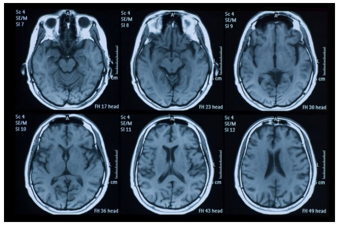
START UP



Convolutional NN

⌛✆☎♋⧫♎◻♋⧫♐⌘

$0.1

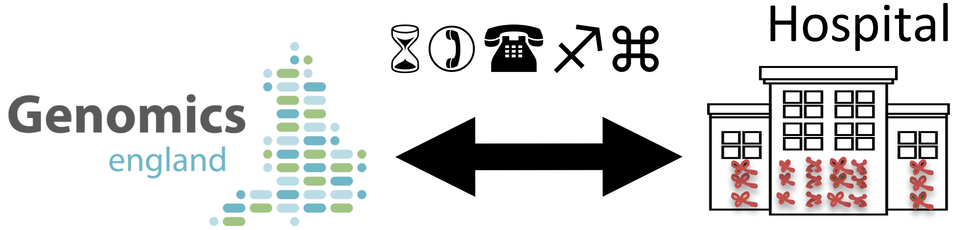


MRI Image

Secure Two-party Computation: “Alice should get (only) the inference result, and the startup should learn nothing”

Problem 2. Blind Training
(application: Collaborative ML)

ID	Genome

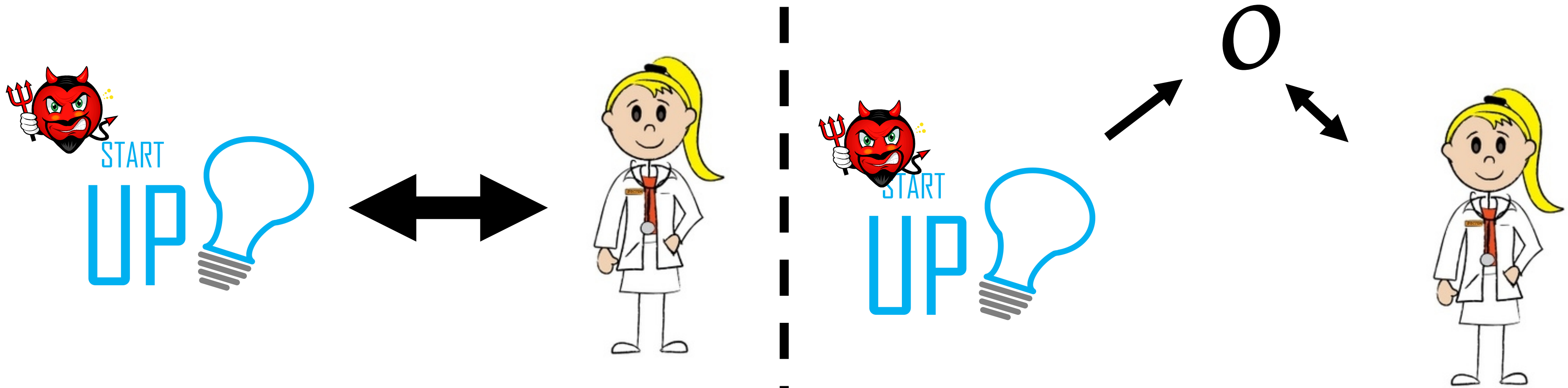


ID	Phenotype

Database could be horizontally or **vertically** partitioned

Secure Two-party Computation: “Parties should learn a classifier (genotype-phenotype correlations) but nothing else”

What does “blind” mean?



REAL WORLD **vs.** IDEAL (Imaginary) WORLD

Defining Security: the Simulation Paradigm [GMR’85]

“Anything learnt on the left could’ve been learnt on the right”

Adversarial capability = **honest-but-curious** vs malicious

Conventional Wisdom (?)

EITHER: Large Communication Overhead *or*

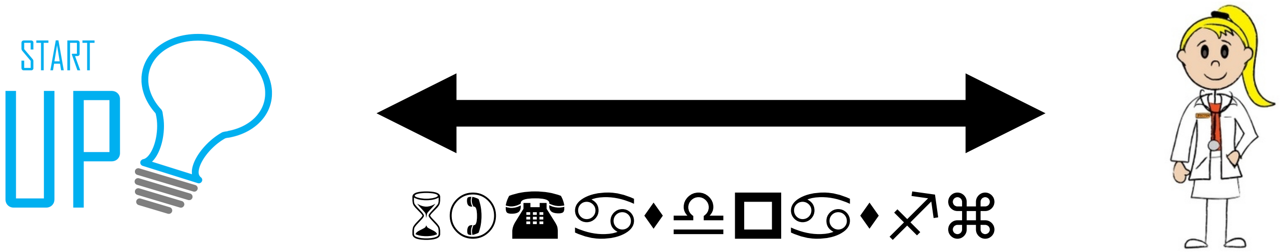
Large Computational Overhead *or*

Only support simple models*

[Lindell-Pinkas’00, Lauter-Naehrig-V.’11, Wu-Haven’12, Graepel-Lauter-Naehrig’12, Nikolaenko-Weinsberg-Ioannidis-Joye-Boneh-Taft’13a,13b, Bost-Popa-Tu-Goldwasser’15 *and many more*]

Secure Computing Techniques I

From the 1980s



- ☐ **Yao's Garbled Circuits** [1986]

  2 parties, lightweight crypto

- ☐ **Goldreich-Micali-Wigderson (GMW) Protocol** [1987]

  2 or more parties, lightweight crypto

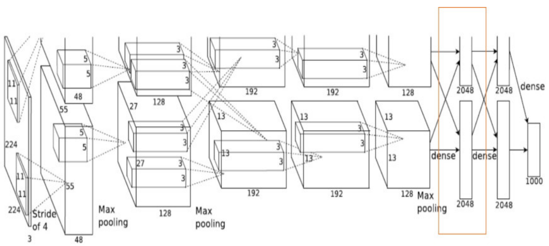
- ☐ **BenOr-Goldwasser-Wigderson (BGW) Protocol** [1988]

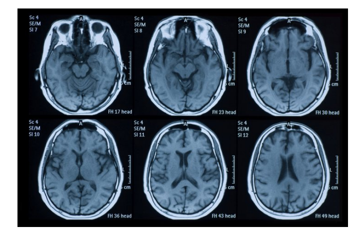
  3 or more parties, < ½ corruption, no crypto

Secure Computing Techniques I

From the 1980s

⌛✆☎♋♦♎◻♋♦♐⌘





PLUS. Efficient computationally.

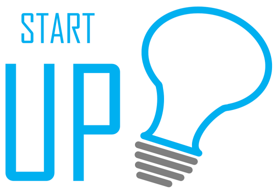
MINUS. Inefficient Communication (∝ Boolean circuit size)

MINUS. Computational efficiency only for Boolean (vs. arithmetic) computations*

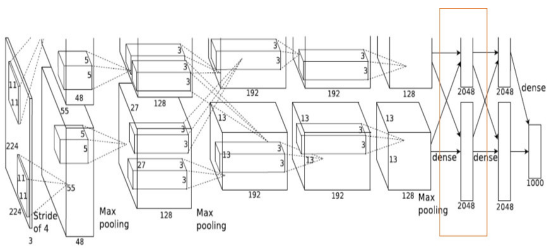
Secure Computing Techniques II

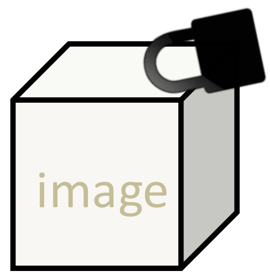
From this decade

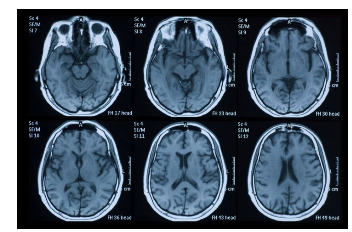
Fully Homomorphic Encryption [Gen’09, BV’11, BGV’12, GSW’13]



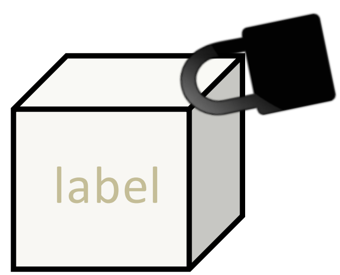
🔔✉✆🗎= $\mathbf{Enc}_{\mathbf{Key}}$**[image]**

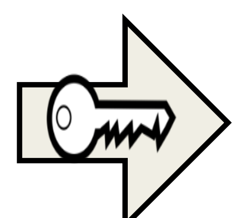






⌛✆☎⌘= $\mathbf{Enc}_{\mathbf{Key}}$**[label]**





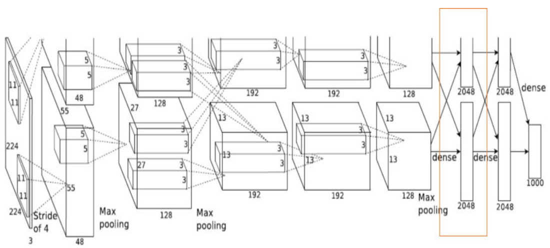
Label

Secure Computing Techniques II

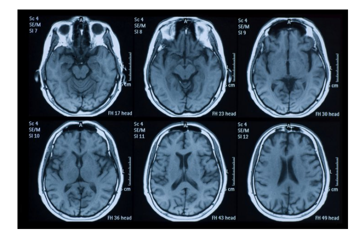
From this decade

Fully Homomorphic Encryption [Gen'09, BV'11, BGV'12, GSW'13]

🕭🖂✆🖹= $\mathbf{Enc_{Key}[image]}$



⌛✆☎⌘= $\mathbf{Enc_{Key}[label]}$



PLUS. Efficient Communication (∝ image size)

PLUS. Native Arithmetic (not just Boolean) Computations

MINUS. Inefficient Computation (∝ degree)

The Old vs The New: Which is Better?

or

How would you get from A to B?

(assume unlimited supply of Ferraris and Camels)

A

B

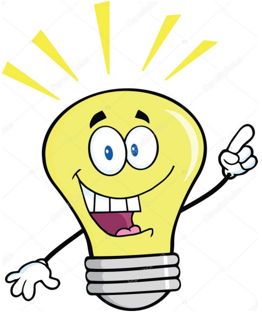
When is FHE Better?

(than garbled circuits/GMW/BGW etc.)

WHEN:

1. **Computation is linear (degree-1)**

   ~~(F~~HE is fast)



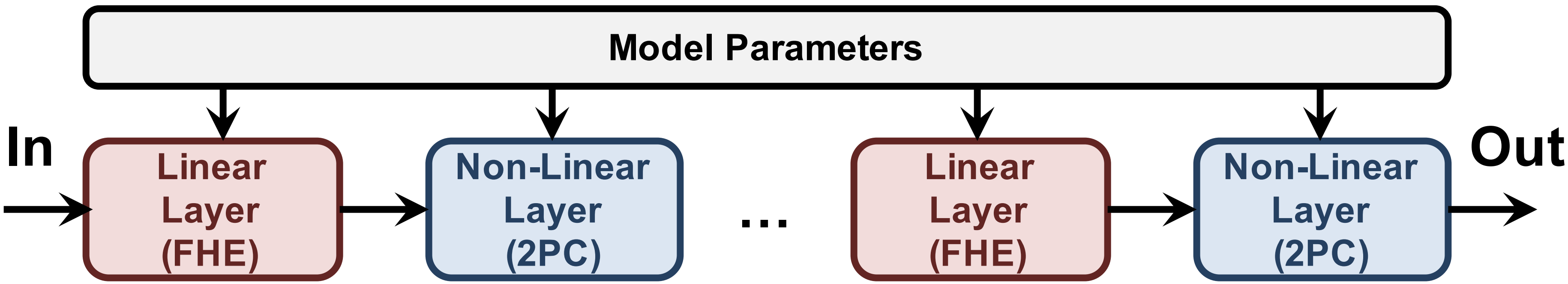
and

2. **Circuit-size is super-linear (say, quadratic)**

   (MPC costs in bandwidth)

Overview of Our Approach

Convolutional Neural Networks: Alternating Linear and Non-linear Layers



1000x lower BW

No Retraining

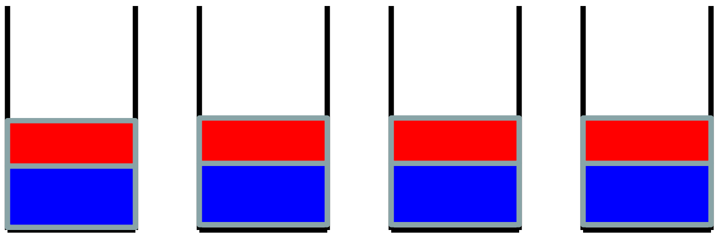
10-50x faster (SW)

Gazelle: Fast HE for CNNs

Fast Homomorphic Encryption Library with Native Support for Neural Network Layers

(extending the PALISADE lattice library)

Basic HE Operations



Plaintexts: 8 bits.

Ciphertexts:

2048 Slots, each 64 Bits

Each Slot: Plaintext & “Noise”

Addition: Add an encrypted vector v to another encrypted vector v’

Scalar Multiplication: Mult encrypted v with plaintext v’ (co-ordinate wise)

Rotation (Automorphism): Permute the slots (typically, rotate)

Gazelle: Fast HE for CNNs

Fast Homomorphic Encryption Library with Native Support for Neural Network Layers

(extending the PALISADE lattice library)

Homomorphic Addition*:

~ 6 µs or 18K clock cycles (for 2048 add)

Homomorphic Scalar Mult*:

~ 14 µs or 42K clock cycles (for 2048 mult)

Homomorphic Slot Rotation:

~ 300 µs or 900K clock cycles (non-amortized)

* single-threaded, no vectorization, 3GHz processor

* CT dimension: 2048, modulus: 64 bits, pt mod: 8 bits

Gazelle: Fast HE for CNNs

Fast Homomorphic Encryption Library with Native Support for Neural Network Layers

(extending the PALISADE lattice library)

Our Work: Homomorphic Matrix-Vector Mult

64 X 2048 matrix of 8-bit numbers

~ 16 ms, 47M clock cycles (ptxt: *at least* 128K)

Our Work: Homomorphic Convolutions

* single-threaded, no vectorization, 3GHz processor

* CT dimension: 2048, modulus: 64 bits, pt mod: 8 bits

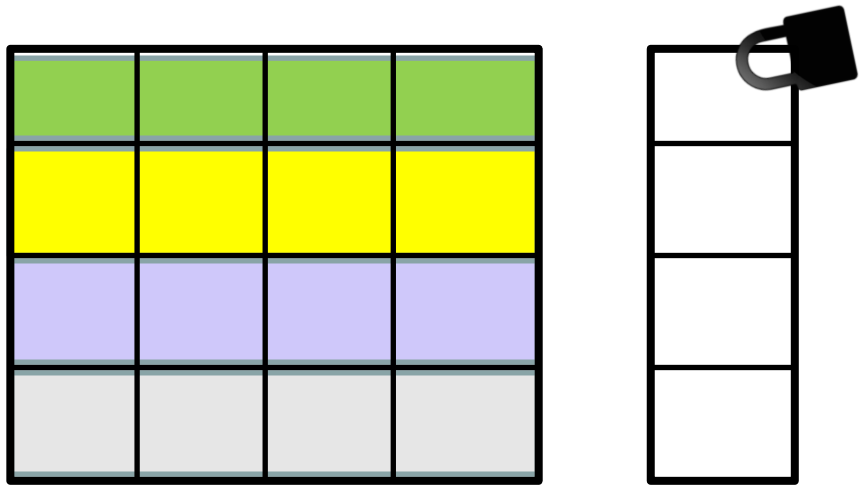
Gazelle: Fast HE for CNNs

Fast Homomorphic Encryption Library with Native Support for Neural Network Layers

(extending the PALISADE lattice library)

MNIST	2 conv, 2 FC, 32*32 input, 400K mult-add **100 ms** comp. + 111 Mb comm. = **111ms***
CIFAR-10	7 conv, 1 FC, 32*32 input, 61M mult-add **1.6s** comp., 2 Gb comm. = **2s***
ImageNet	5 conv, 3 FC, 256*256 input, 1.3G mult-add **20s** comp., 20 Gb comm. = **20s***

Fast Matrix Multiplications



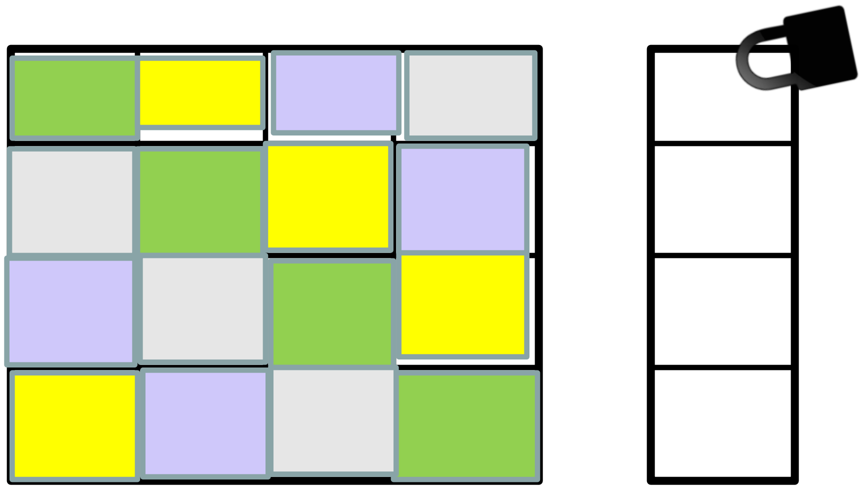
1. Simple Mult: Each matrix row with the encrypted vector

Lots of rotations (N log N)

Reasonable noise growth $\eta_0 \, X \, \eta_{mult} + \eta_{rot}$

Evaluated ciphertexts are not packed
(one number per ciphertext)

Fast Matrix Multiplications



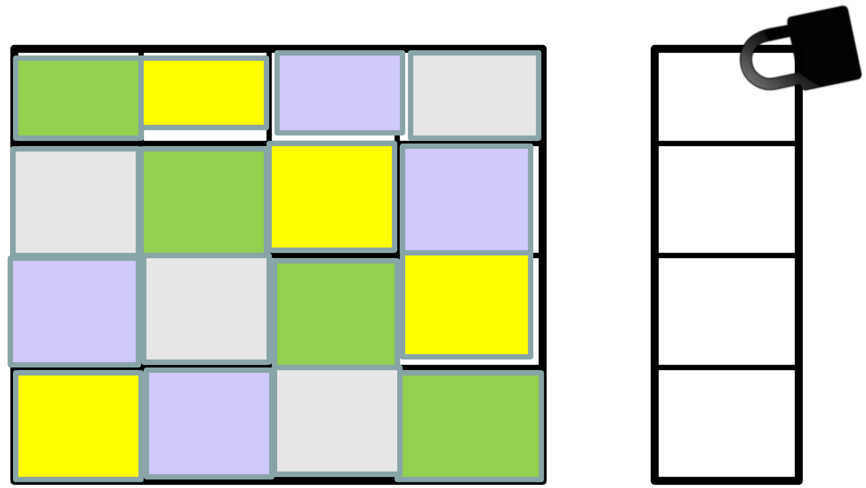
2. Diagonal Multiplication:

IDEA: Non-interacting numbers go into same ciphertext

Fewer rotations (O(N) on the encrypted vector)

Bigger noise growth $(\eta_0 + \eta_{rot})\ X\ \eta_{mult}$

Fast Matrix Multiplications



3. Interpolating between 1 & 2 (“Baby Step Giant Step”)

4. “Hoisting”: optimized [Halevi-Shoup’17]

“N *input* rotations (almost) for the price of one”

Ongoing & Future Work

- Programming Framework for Encrypted CNNs.

  Mostly handcoded + some automatic optimization

  Can we come up with the best homomorphic evaluation automatically?

- Beyond CNNs? Limits of encrypted computation

- Encrypted ML Training?

Thank you!