Privacy Preserving Machine Learning

百万富翁问题

两个富翁的财产是 1 到 10 之间的整数。

他们如何在不透露自己财产的情况下，比较谁更富有?

Yao, Andrew C. "Protocols for secure computations.“

Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on. IEEE, 1982.

Prediction as a Service

Medical

Genomic

Financial

Who else is going to see your DNA sequence and the prediction?

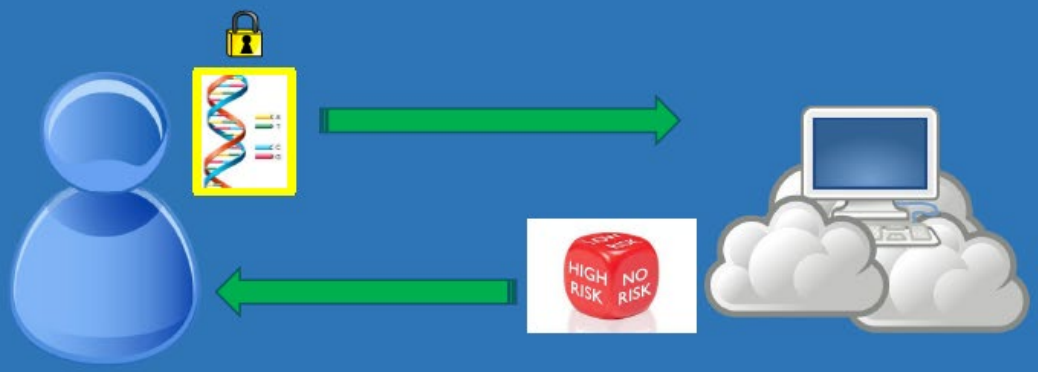
Who else is going to see your DNA sequence and the prediction?

"Sorry, your DNA does not match this job description."

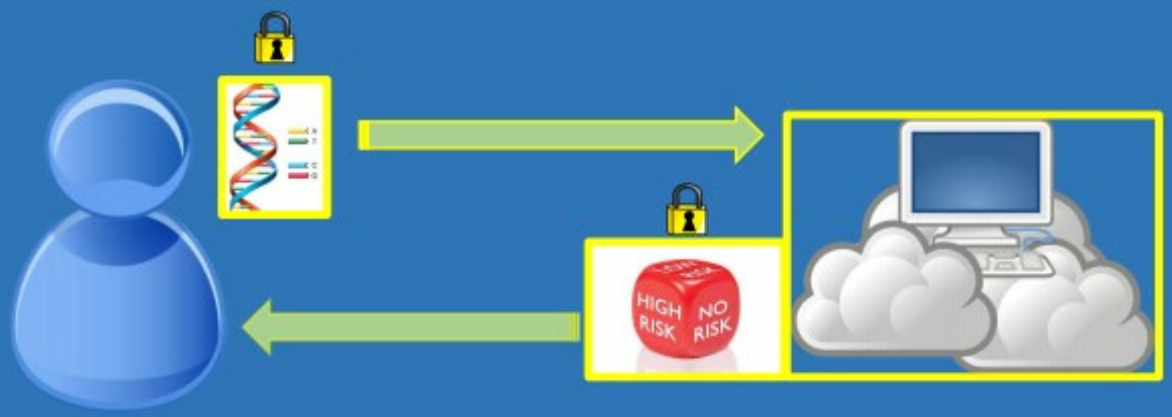
"Here is an advertisement that according to your DNA you will not be able to resist."

"We are not giving you this loan because it is not in your DNA to pay it back."

Inference Over Encrypted Data

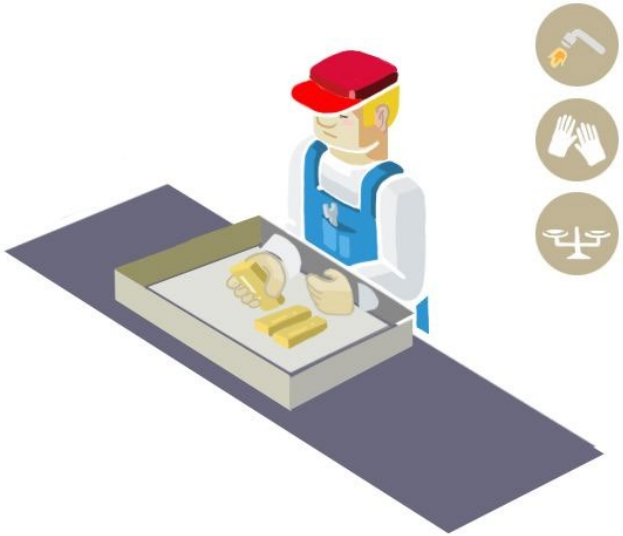


Inference Over Encrypted Data



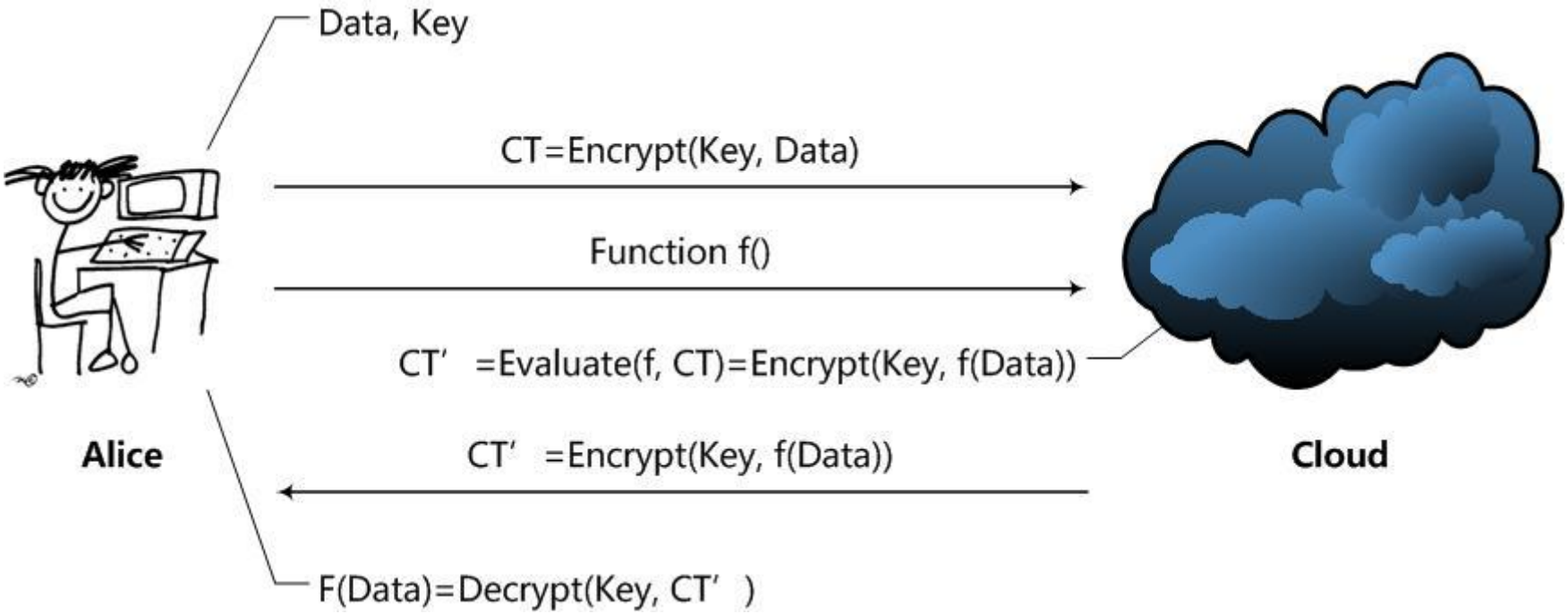
1. Homomorphic Encryption
2. Differential privacy

Homomorphic Encryption

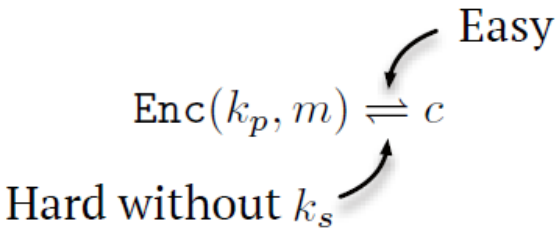


A way to delegate processing of your data, without giving away access to it.

[Craig Gentry 09]


Data, Key
CT=Encrypt(Key, Data)
Function f()
CT' =Evaluate(f, CT)=Encrypt(Key, f(Data))
Alice
CT' =Encrypt(Key, f(Data))
Cloud
F(Data)=Decrypt(Key, CT')

Fundamental point is ...



$$\texttt{Dec}(k_s, c) = m$$

The *security level* of an encryption scheme is the order of the number of operations required to crack it (decrypt without k_s).

What is Homomorphic Encryption (HE)?

- Allows for computation on encrypted data
- Enables outsourcing of data storage/processing

History of HE:

- Rivest, Adleman, Dertouzos (1978) -- “On Data Banks and Privacy Homomorphisms”
- Gentry (2009) -- “A Fully Homomorphic Encryption Scheme”
- Multiple HE schemes developed after 2009

Here are a few popular ones with characteristics we like:

- Efficient Homomorphic Encryption on Integer Vectors and Its Applications
- Yet Another Somewhat Homomorphic Encryption (YASHE)
- Somewhat Practical Fully Homomorphic Encryption (FV)
- Fully Homomorphic Encryption without Bootstrapping

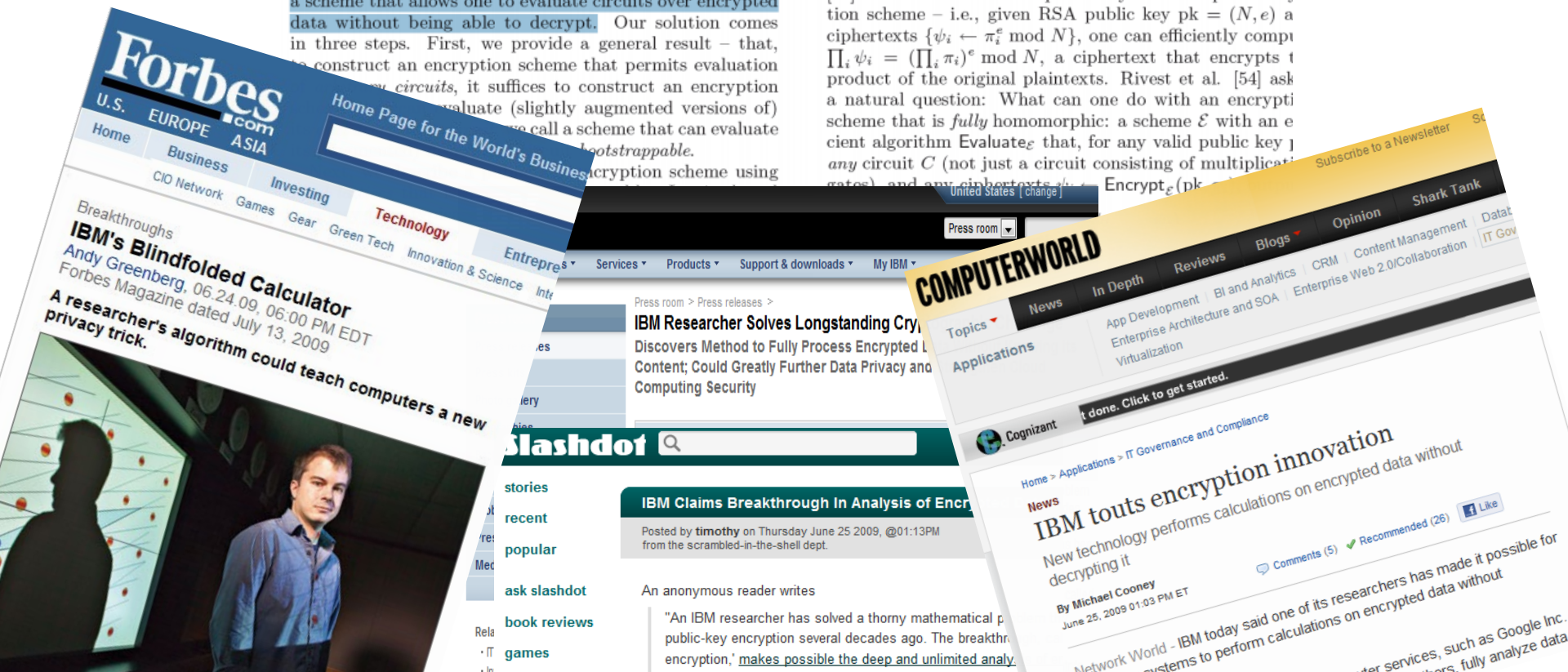
Fully Homomorphic Encryption Using Ideal Lattices

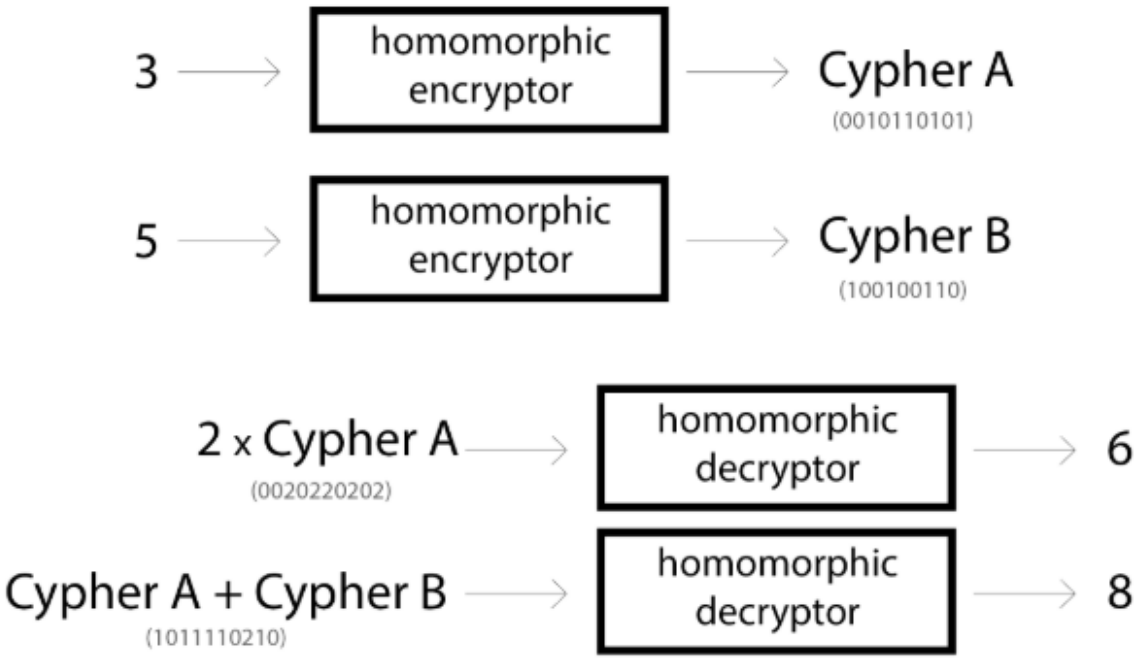
Craig Gentry
Stanford University and IBM Watson
cgentry@cs.stanford.edu

ABSTRACT

We propose a fully homomorphic encryption scheme – i.e., a scheme that allows one to evaluate circuits over encrypted data without being able to decrypt. Our solution comes in three steps. First, we provide a general result – that, to construct an encryption scheme that permits evaluation ... circuits, it suffices to construct an encryption ... evaluate (slightly augmented versions of) ... call a scheme that can evaluate ... bootstrappable. ... cryption scheme using ...

duced by Rivest, Adleman and Dertouzos [54] shortly ter the invention of RSA by Rivest, Adleman and Shar [55]. Basic RSA is a multiplicatively homomorphic encry tion scheme – i.e., given RSA public key $\text{pk} = (N, e)$ a ciphertexts $\{\psi_i \leftarrow \pi_i^e \bmod N\}$, one can efficiently compu $\prod_i \psi_i = (\prod_i \pi_i)^e \bmod N$, a ciphertext that encrypts t product of the original plaintexts. Rivest et al. [54] ask a natural question: What can one do with an encrypti scheme that is *fully* homomorphic: a scheme $\mathcal{E}$ with an e cient algorithm $\text{Evaluate}_{\mathcal{E}}$ that, for any valid public key *any* circuit C (not just a circuit consisting of multiplicat gates), and any ciphertexts ... $\text{Encrypt}_{\mathcal{E}}(\text{pk}, \ldots)$




3
homomorphic encryptor
Cypher A
(0010110101)
5
homomorphic encryptor
Cypher B
(100100110)
2 x Cypher A
(0020220202)
homomorphic decryptor
6
Cypher A + Cypher B
(1011110210)
homomorphic decryptor
8

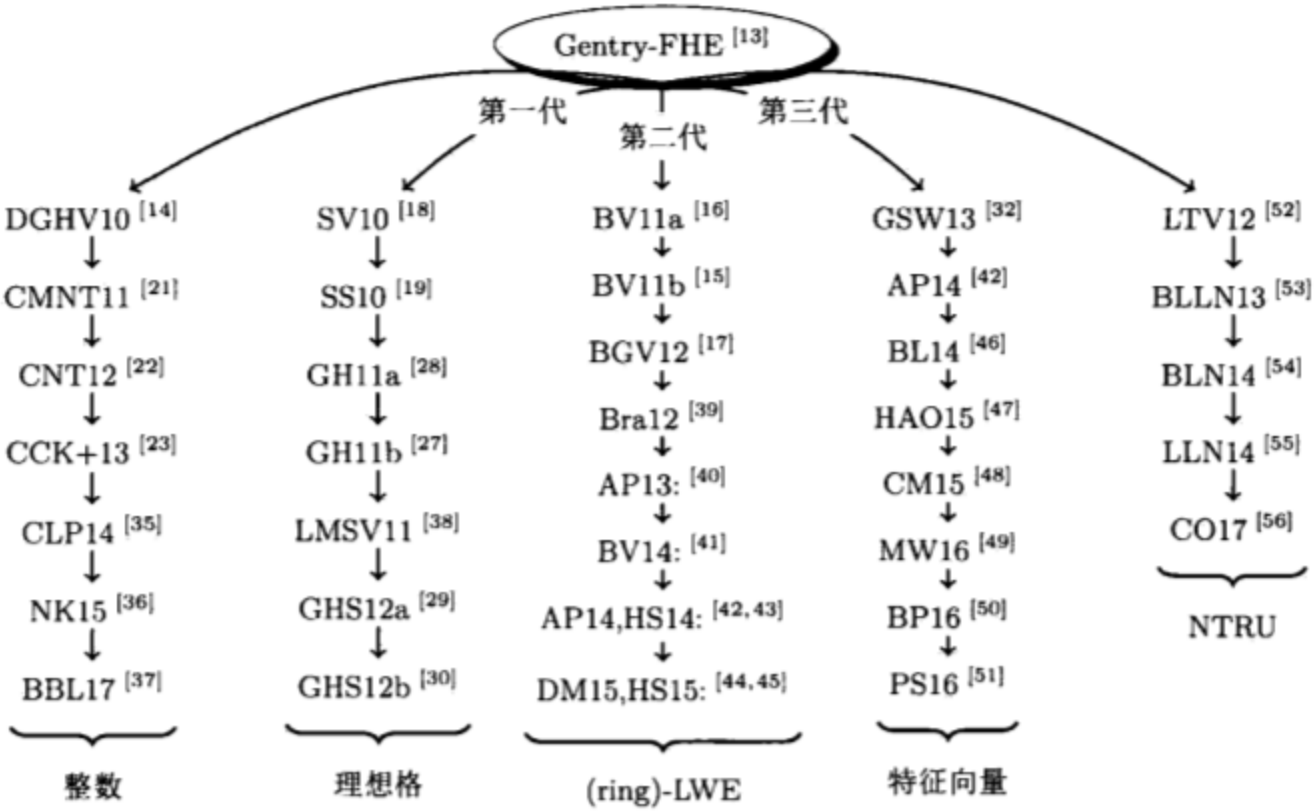
Definition (Homomorphic encryption scheme)

An encryption scheme is said to be *homomorphic* if there is a set of operations $\circ \in \mathcal{F}_M$ acting in message space (such as addition) that have corresponding operations $\diamond \in \mathcal{F}_C$ acting in cipher text space satisfying the property:

$$\mathsf{Dec}(k_s, \mathsf{Enc}(k_p, m_1) \diamond \mathsf{Enc}(k_p, m_2)) = m_1 \circ m_2 \quad \forall\, m_1, m_2 \in M$$

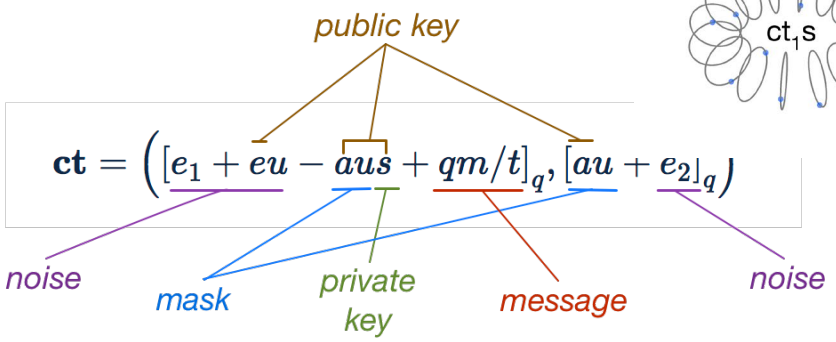
A scheme is *fully homomorphic* if $\mathcal{F}_M = \{+, \times\}$ and an arbitrary number of such operations are possible.

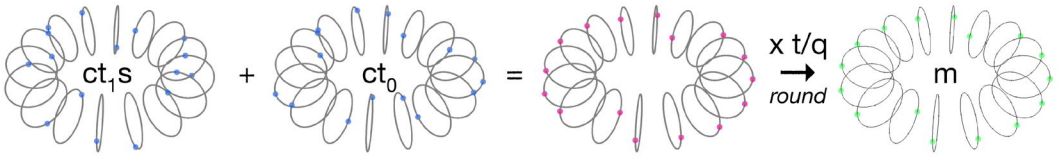
Library	URL	License	Language
HEANN	https://github.com/kimandrik/HEAAN	CC non commercial	C++
NFLib	https://github.com/quarkslab/NFLlib	GPLv3	C++
FV-NFLib	https://github.com/quarkslab/NFLlib	GPLv3	C++
cuHE	https://github.com/vernamlab/cuHE	MIT	CUDA C++
PALISADE	https://git.njit.edu/palisade/PALISADE/tree/master	Liberal	C++
SEAL	https://www.microsoft.com/en-us/research/project/simple-encrypted-arithmetic-library/	Microsoft research only	C++
HELib	https://github.com/shaih/HElib	Apache 2.0	C++
jLBC	http://gas.dia.unisa.it/projects/jlbc/download.html	LGPLv3	Java


Gentry-FHE [13]
第一代
第二代
第三代
DGHV10 [14]
CMNT11 [21]
CNT12 [22]
CCK+13 [23]
CLP14 [35]
NK15 [36]
BBL17 [37]
整数
SV10 [18]
SS10 [19]
GH11a [28]
GH11b [27]
LMSV11 [38]
GHS12a [29]
GHS12b [30]
理想格
BV11a [16]
BV11b [15]
BGV12 [17]
Bra12 [39]
AP13: [40]
BV14: [41]
AP14,HS14: [42,43]
DM15,HS15: [44,45]
(ring)-LWE
GSW13 [32]
AP14 [42]
BL14 [46]
HAO15 [47]
CM15 [48]
MW16 [49]
BP16 [50]
PS16 [51]
特征向量
LTV12 [52]
BLLN13 [53]
BLN14 [54]
LLN14 [55]
CO17 [56]
NTRU

2.0 – Understanding HE

- Plaintext: elements and operations of a polynomial ring (mod x^n+1, mod p).
  - Example: $3x^5 + x^4 + 2x^3 + ...$
- Ciphertext: elements and operations of a polynomial ring (mod x^n+1, mod q).
  - Example: $7862x^5 + 5652x^4 + ...$





morphism is a *structure-preserving* transformation. For example, consider the map $\Phi : \mathbb{Z} \to \mathbb{Z}_7$ such that $\Phi(z) := z \pmod 7$. This map Φ preserves both the additive and multiplicative structure of the integers in the sense that for every $z_1, z_2 \in \mathbb{Z}$, we have that $\Phi(z_1 + z_2) = \Phi(z_1) \oplus \Phi(z_2)$ and $\Phi(z_1 \cdot z_2) = \Phi(z_1) \otimes \Phi(z_2)$ where $\oplus$ and $\otimes$ are the addition and multiplication operations in $\mathbb{Z}_7$. The map Φ is a ring homomorphism between the rings $\mathbb{Z}$ and $\mathbb{Z}_7$.

Introduction to the API

Let's start with a quick demo of the API. First thing, let's create public and private keys by using a key length in bits long enough to get decent cryptographic guarantees:

```
>>> import phe as paillier
>>> pubkey, privkey = paillier.generate_paillier_keypair(n_length=1024)
```

Paillier is an asymmetric cryptoscheme (like RSA), where the public key is used for encryption and the private key for decryption.

```
>>> secret_numbers = [3.141592653, 300, -4.6e-12]
>>> encrypted_numbers = [pubkey.encrypt(x) for x in secret_numbers]
>>> [privkey.decrypt(x) for x in encrypted_numbers]
[3.141592653, 300, -4.6e-12]
```

But what do these encrypted numbers look like? You can open up the object and look inside at the integer representation.

```
>>> print(encrypted_numbers[0])
<phe.paillier.EncryptedNumber object at 0x7f02f2849dd8>
>>> print(encrypted_numbers[0].ciphertext())
50727523990589201897301825868119129024634744806677124327179597748195870744893252252142409987781503731971126374488166629310169703734073890342751905581823438587211139408707094099240171665974075433551018157079366369056407499635750279632160116464975647241537291031471387475118541219343274068776292947602782415543164098595730658936817678022192027717289631915231522549748084512692629324263583397073610347387379408438679715777728991911778903338803570615181347451465132285058137852689019916472620583557940728497906324186799612131622394956002911272084080828823052193633303271548901725390879184773782119863238868147274805570 38
```

Paillier encryption is a great tool for preserving simple arithmetic operations in the encrypted space. We can sum two encrypted numbers and decrypt the result and that will be equal to the sum of the original numbers.

```
>>> x, y = 2, 0.5
>>> encrypted_x = pubkey.encrypt(x)
>>> encrypted_y = pubkey.encrypt(y)
>>> encrypted_sum = encrypted_x + encrypted_y
>>> privkey.decrypt(encrypted_sum)
2.5
```

In the same way, multiplication of an encrypted number by a number in the clear works.

```
>>> z = 10
>>> privkey.decrypt(z * encrypted_x)
20
```

1. Message space
   - Commonly only easy to encrypt binary/integers
2. Cipher text size
   - Present schemes all inflate the size of data substantially (e.g. 1MB → 16.4GB)
3. Computational cost
   - 1000's additions per sec
   - ≈ 50 multiplications per sec
4. Division and comparison operations
   - Impossible!
5. Depth of operations
   - After a certain depth of multiplications, need to 'refresh' cipher text: hugely time consuming, so avoid!

Dimension	KeyGen	PK size	ReCrypt
2048	40 sec	70 MB	31 sec
8192	8 min	285 MB	3 min
32768	2 hours	2.3 GB	0.5 hours

Gentry的FHE方案

Dimension	KeyGen	Encrypt	Mult. / Dec	Degree
2048 (800,000bit)	1.25s	0.06s	0.023s	~200
8192 (3,200,000bit)	10s	0.7s	0.12s	~200
32768 (13,000,000bit)	95s	5.3s	0.6s	~200

Smart-Vercauteren的SWHE方案效率

2011年，Gentry和Halevi在IBM尝试实现了两个HE方案：Smart-Vercauteren的SWHE方案[SV10]以及Gentry的FHE方案[Gen09]，并公布了效率。

（原始数据可以在2nd Bar-Ilan Winter School on Cryptography找到）

Homomorphic Encryption

Pros

- Protects the privacy of the data owner
- Protects the privacy of the model owner (evaluator)
- Multiple data owners possible
- In best case can use CRT batching for good performance

Cons

- Currently needs to be applied and tuned on case-by-case basis
- Performance depends hugely on model
- Difficulty of integer comparisons is a major issue

- Guide to Applications of Homomorphic Encryption
  - https://www.youtube.com/watch?v=PeE4C7Opcsw
- A Homomorphic Encryption Illustrated Primer
  - https://blog.n1analytics.com/homomorphic-encryption-illustrated-primer/#homomorphic-operations
- Encrypt your Machine Learning

  How Practical is Homomorphic Encryption for Machine Learning?
  - https://medium.com/corti-ai/encrypt-your-machine-learning-12b113c879d6
- Distributed machine learning and partially homomorphic encryption (Part 1)
  - https://blog.n1analytics.com/distributed-machine-learning-and-partially-homomorphic-encryption-1/

- **https://github.com/OpenMined/private-ai-resources**

Secure Deep Learning

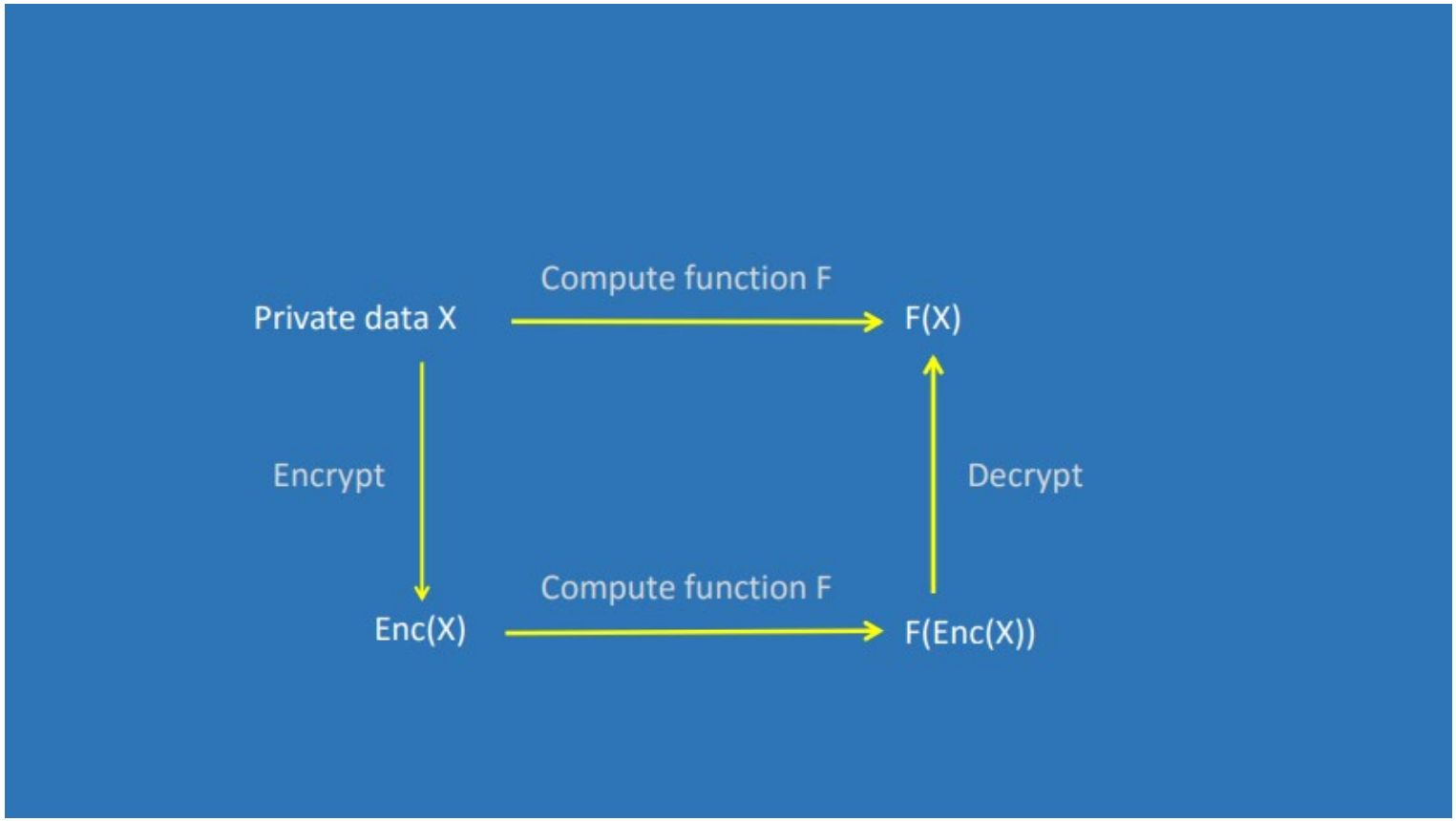
- Gazelle: A Low Latency Framework for Secure Neural Network Inference, January 16, 2018
- Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications, November 29, 2017
- CryptoDL: Deep Neural Networks over Encrypted Data, November 14, 2017
- MiniONN: Oblivious Neural Network Predictions via MiniONN Transformations, November 3, 2017
- DeepSecure: Scalable Provably-Secure Deep Learning, May 24, 2017
- SecureML: A System for Scalable Privacy-Preserving Machine Learning, April 19, 2017
- CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy, February 24, 2016
- Privacy-Preserving Deep Learning, October 12, 2015

Libraries and Frameworks

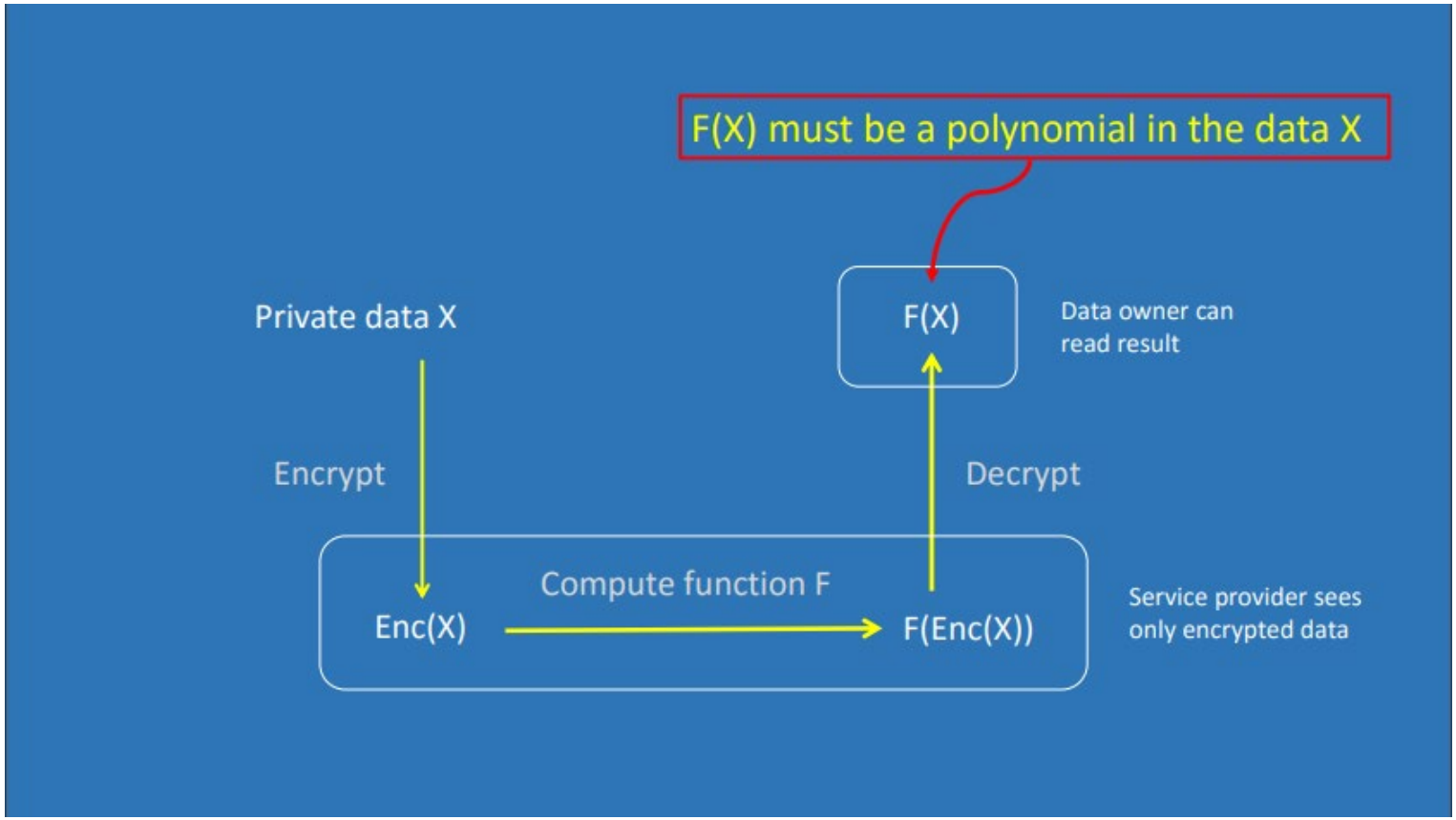
- TinyGarble: Logic Synthesis and Sequential Descriptions for Yao's Garbled Circuits
- SPDZ-2: Multiparty computation with SPDZ and MASCOT offline phase
- ABY: A Framework for Efficient Mixed-Protocol Secure Two-Party Computation
- Obliv - C: C compiler for embedding privacy preserving protocols:
- TFHE: Fast Fully Homomorphic Encryption Library over the Torus
- SEAL: Simple Encypted Arithmatic Library
- PySEAL: Python interface to SEAL
- HElib: An Implementation of homomorphic encryption

General Research

- Overdrive: Making SPDZ Great Again
- Privacy-Preserving Logistic Regression Training
- Between a Rock and a Hard Place: Interpolating Between MPC and FHE
- Privacy-Preserving Boosting with Random Linear Classifiers for Learning from User-Generated Data
- The Secret Sharer: Measuring Unintended Neural Network Memorization & Extracting Secrets
- Improvements for Gate-Hiding Garbled Circuits
- Practical Secure Aggregation for Privacy-Preserving Machine Learning
- CryptoRec: Secure Recommendations as a Service
- Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data
- Communication-Efficient Learning of Deep Networks from Decentralized Data
- Differentially Private Generative Adversarial Network
- Doing Real Work with FHE: The Case of Logistic Regression
- ADSNARK: Nearly Practical and Privacy-Preserving Proofs on Authenticated Data
- Scalable Private Learning with PATE
- Doing Real Work with FHE: The Case of Logistic Regression
- Reading in the Dark: Classifying Encrypted Digits with Functional Encryption
- Stealing Hyperparameters in Machine Learning


Compute function F
Private data X
F(X)
Encrypt
Decrypt
Compute function F
Enc(X)
F(Enc(X))

Homomorphic Encryption + Machine learning


F(X) must be a polynomial in the data X
Private data X
F(X)
Data owner can read result
Encrypt
Decrypt
Compute function F
Enc(X)
F(Enc(X))
Service provider sees only encrypted data

HE+ LR

Fast for certain types of models

Linear regression, linear mixed models

- Useful in genomics
- Super fast to evaluate on homomorphically encrypted data
- Demonstrated on realistic size examples
- Easy to utilize batching even for single prediction

Other intelligible polynomial models

- Demonstrated for realistic medical risk prediction
- Batching may or may not be easy to utilize

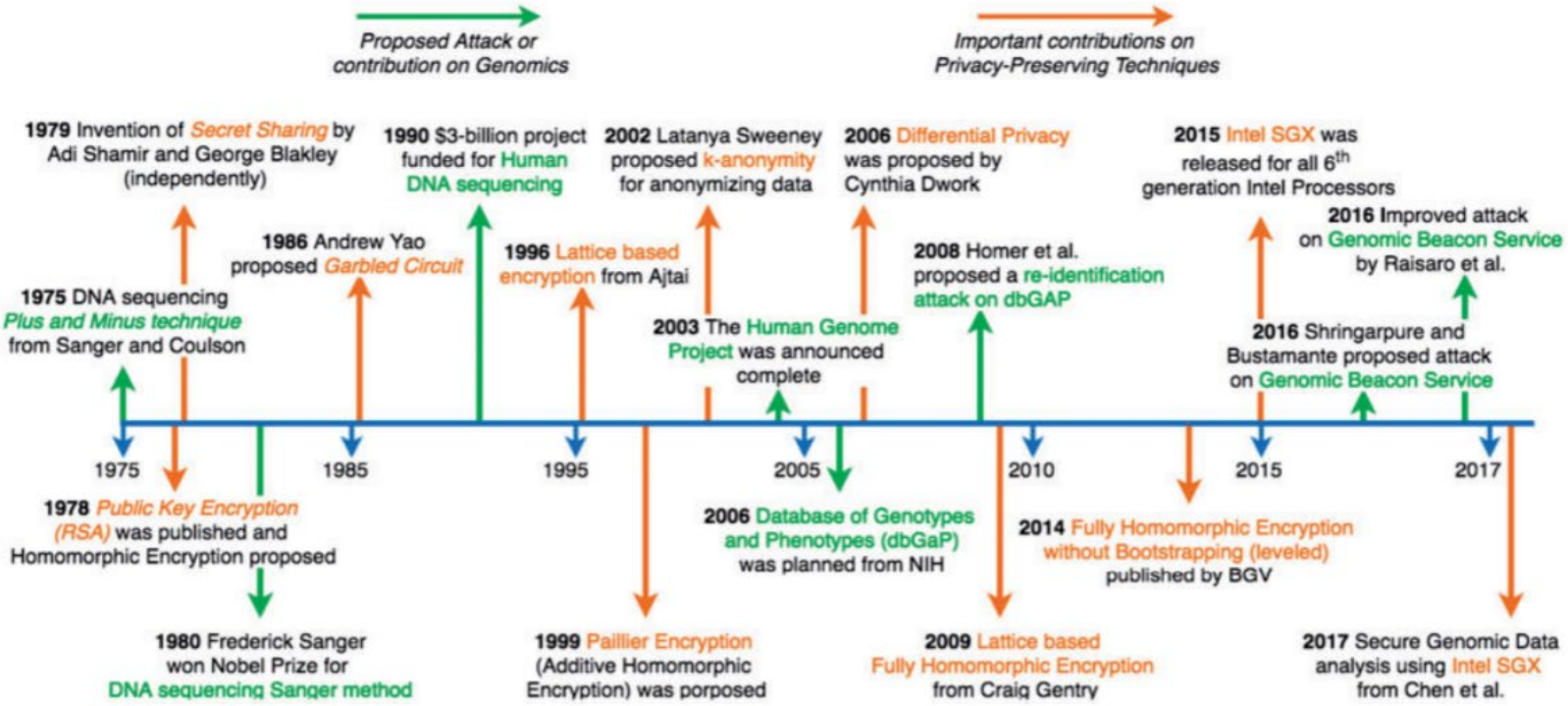


Figure 1. Timeline of the evolution of genomic data studies and seminal development of different privacy-preserving techniques.

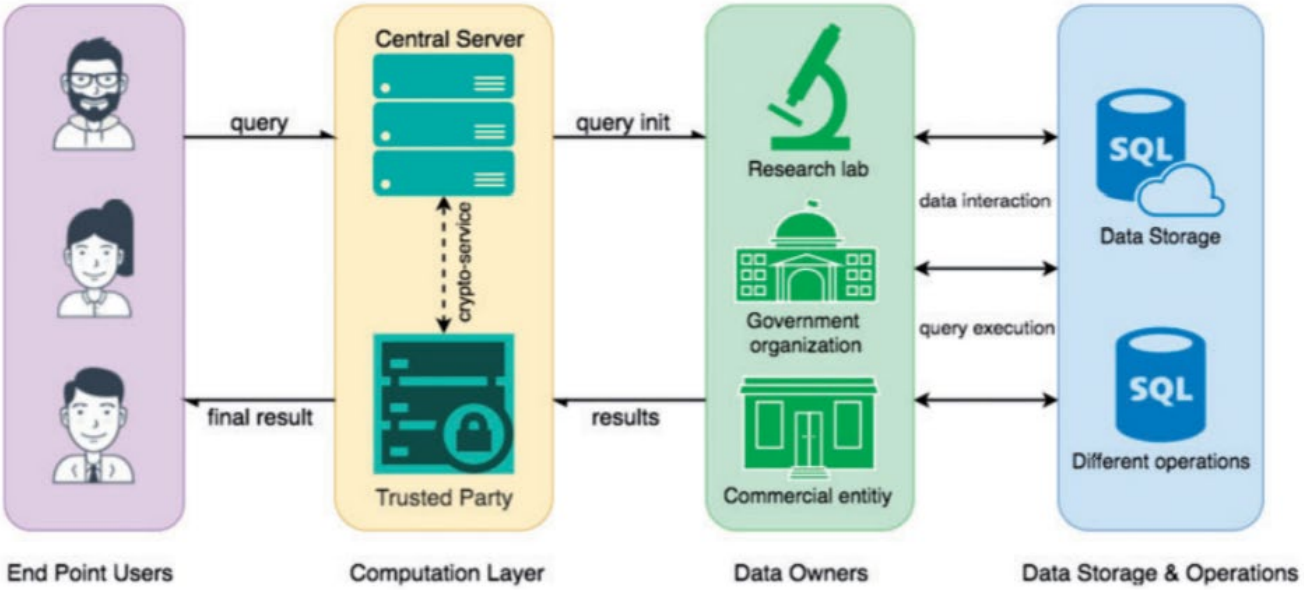


Figure 2. Different entities involved in a genomic data computation.

➢ Privacy-Preserving Ridge Regression with only Linearly-Homomorphic Encryption

Ridge Regression

Training: given $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1,\dots,n}$

$$\text{find argmin of } F(\mathbf{w}) = \underbrace{\sum_{i=1}^{n}(y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle)^2}_{\text{mean squared error}} + \lambda \underbrace{||\mathbf{w}||_2^2}_{\text{regularization}}$$

This can be done in two steps:

- Step 1: Compute the matrix $A = \sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i + \lambda I$ and the vector $\mathbf{b} = \sum_{i=1}^{n} y_i \mathbf{x}_i$
- Step 2: Solve the linear system $A \cdot \mathbf{w} = \mathbf{b}$

Phase 1: merging the local data silos

Goal: compute the encryption of $A = \sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i + \lambda I$ and $\mathbf{b} = \sum_{i=1}^{n} y_i \mathbf{x}_i$

It depends on the distributed setting:

- horizontally-partitioned datasets

$$\boxplus_{i=1}^{n} \text{Enc}_{pk}(\mathbf{x}_i^\top \mathbf{x}_i)$$

- arbitrarily-partitioned datasets

$$\text{Enc}_{pk}(\mathbf{x}_i(j)) \cdot \text{Enc}_{pk}(\mathbf{x}_k(h))$$

$$\begin{pmatrix} - & \mathbf{x}_1 & - & y_1 \\ - & \mathbf{x}_2 & - & y_2 \\ - & \mathbf{x}_3 & - & y_3 \\ \vdots & \vdots & & \vdots \\ - & \mathbf{x}_n & - & y_n \end{pmatrix}$$

(1 multiplication done via Labeled Encryption, Barbosa et al. ESORICS 2017)

Phase 2: solving $A \cdot \mathbf{w} = \mathbf{b}$

18

ML Server: $\text{Enc}_{pk}(A), \text{Enc}_{pk}(\mathbf{b})$
Crypto Provider: sk

Interactive protocol:

1. ML Server "masks inside the encryption"
   $\text{Enc}_{pk}(A) \rightarrow \text{Enc}_{pk}(A \cdot R)$
   $\text{Enc}_{pk}(\mathbf{b}) \rightarrow \text{Enc}_{pk}(\mathbf{b} + A \cdot \mathbf{r})$
2. Crypto Provider decrypts, gets $\tilde{A} = A \cdot R$, $\tilde{\mathbf{b}} = \mathbf{b} + A \cdot \mathbf{r}$ and computes a "masked model", $\tilde{\mathbf{w}} = \tilde{A}^{-1}\tilde{\mathbf{b}}$
3. ML Server computes the real model $\mathbf{w}$ from the masked one

$$\mathbf{w} = R \cdot \tilde{\mathbf{w}} - \mathbf{r}$$

➢ Privacy-preserving logistic regression with distributed data sources via homomorphic encryption

$$J(\theta) = \frac{\lambda}{2N_{\text{data}}}\sum_{j=1}^{d}\theta_j^2 + \frac{1}{N_{\text{data}}}\sum_{i=1}^{N_{\text{data}}}\Big[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\Big] \quad (1)$$

in which $0 < h_\theta(x) < 1$ is the value of the sigmoid function $h_\theta : \mathbb{R}^{d+1} \to \mathbb{R}$ given by

$$h_\theta(x) = \frac{1}{1+\exp(-\theta^T x)} = \frac{1}{1+\exp(-\sum_{j=0}^{d}\theta_j x_j)}.$$

$$\log(1-h_\theta(x)) \approx \sum_{j=0}^{k} a_j(\theta^T x)^j \quad (4)$$

$$\log(h_\theta(x)) \approx \sum_{j=0}^{k}(-1)^j a_j(\theta^T x)^j \quad (5)$$

The approximate function of $J(\theta)$ is formed by using the polynomials at (4) and (5) to replace the corresponding logarithm functions in (1). Namely, define the following cost function

$$J_{\text{approx}}(\theta) = \frac{\lambda}{2N_{\text{data}}}\sum_{j=1}^{d}\theta_j^2 + J^*_{\text{approx}}(\theta) \quad (6)$$

we can finally express

$$J^*_{\text{approx}}(\theta) = \frac{1}{N_{\text{data}}}\sum_{j=1}^{k}\sum_{r_1,\ldots,r_j=0}^{d} a_j(\theta_{r_1}\cdots\theta_{r_j})A_{j,r_1,\ldots,r_j} - a_0 \quad (8)$$

Since degree $k = 2$ is mainly used in later sections, we give the explicit formulas of (7) as

$$A_{1,r_1} = \sum_{i=1}^{N_{\text{data}}} \underbrace{\left(2y^{(i)}-1\right)\left(x_{r_1}^{(i)}\right)}_{\text{owned by the } i^{\text{th}} \text{ data source}} \quad (9)$$

$$A_{2,r_1,r_2} = \sum_{i=1}^{N_{\text{data}}} \underbrace{(-1)\left(x_{r_1}^{(i)}x_{r_2}^{(i)}\right)}_{\text{owned by the } i^{\text{th}} \text{ data source}} \quad (10)$$

➢ Secure Logistic Regression Based on Homomorphic Encryption: Design and Evaluation

Let $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \{\pm 1\}$ be the supervised learning samples for $i = 1, \cdots, n$. If we write $\boldsymbol{z}_i = y_i \cdot (1, \boldsymbol{x}_i) \in \mathbb{R}^{d+1}$, then the cost function for logistic regression is defined by $\frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(-\boldsymbol{z}_i^T \boldsymbol{\beta}))$. Its gradient with respect to $\boldsymbol{\beta}$ is computed by $-\frac{1}{n}\sum_{i=1}^{n} \sigma(-\boldsymbol{z}_i^T \boldsymbol{\beta}) \cdot \boldsymbol{z}_i$. To find a local minimum point, the gradient descent method updates the regression parameters using the following formula until $\boldsymbol{\beta}$ converges:

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \frac{\alpha}{n} \sum_{i=1}^{n} \sigma(-\boldsymbol{z}_i^T \boldsymbol{\beta}) \cdot \boldsymbol{z}_i,$$

In our implementation, we used the degree 3 and 7 least squares approximations of the sigmoid function over the interval $[-8, 8]$, which contains all of the input values $(-\boldsymbol{z}_i^T \boldsymbol{\beta})$ during iterations. The least square polynomials are computed as:

$$g_3(x) = a_0 + a_1(x/8) + a_3(x/8)^3,$$
$$g_7(x) = b_0 + b_1(x/8) + b_3(x/8)^3 + b_5(x/8)^5 + b_7(x/8)^7,$$

➢ SecureML：A System for Scalable Privacy-Preserving Machine Learning

The SGD algorithm works as follows: $\mathbf{w}$ is initialized as a vector of random values or all 0s. In each iteration, a sample $(\mathbf{x}_i, y_i)$ is selected randomly and a coefficient w_j is updated as

$$w_j := w_j - \alpha \frac{\partial C_i(\mathbf{w})}{\partial w_j}. \tag{1}$$

➢ Logistic regression over encrypted data from fully homomorphic encryption

Gradient descent. The standard method for training logistic regression is gradient descent. To fix notation, let D be the number of (binary) features, and N the number of training records of the form (X, y), where $X \in \mathbb{R}^{N \times D}$, $y \in \mathbb{R}^N$. In this case the weight vector w is in $\mathbb{R}^D$.

Gradient descent proceeds in iterations, where in each iteration the weight vector w is updated as

$$w \leftarrow w - \alpha(\sigma(Xw) - y)X^T,$$

HE+ NN

Less fast for other types of models

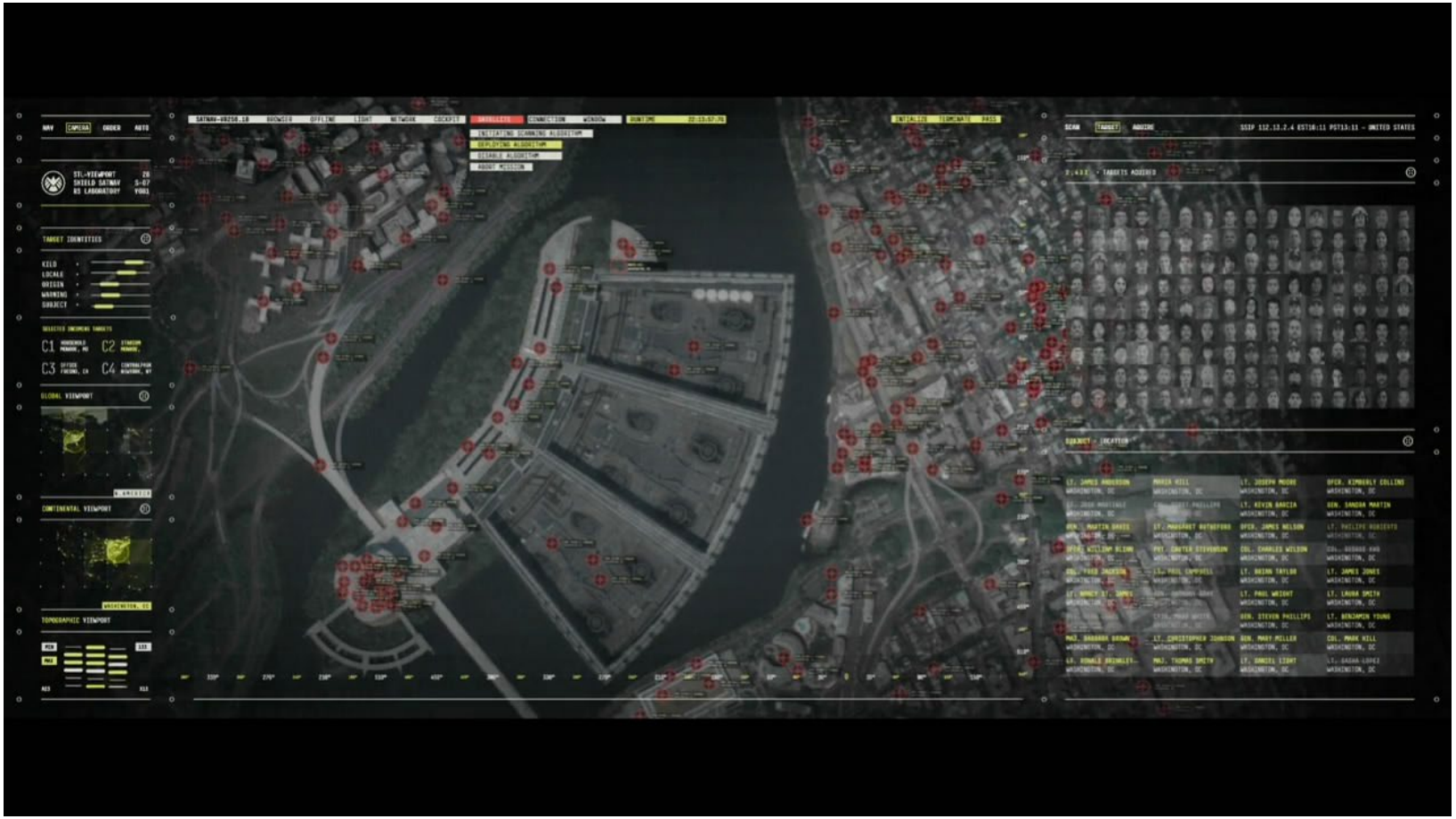
Neural networks

- Very powerful, high accuracy for difficult problems
- Homomorphic evaluation challenging due to size
- Depth limited with leveled fully homomorphic encryption
- Activation functions need to be low-degree polynomials
- Demonstrated (C)NN on MNIST data (hand-written digit recognition)
  - CryptoNets
  - O(10 seconds) for 4096 simultaneous predictions on encrypted data
  - Great *amortized* performance
  - Problem: How to use batching more efficiently?
  - Problem: How to extend to bigger NNs?

➢ Building Safe A.I.

A Tutorial for Encrypted Deep Learning

https://iamtrask.github.io/2017/03/17/safe-ai/



- **Addition** $+$
- **Multiplication** $\times$
- **Division** $\div$
- **Subtraction** $-$
- **Sigmoid** $^{1}/_{1+e^{-x}}$
- **Tanh** $\tan x$
- **Exponential** a^x

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots \quad for\,|x| < 1$$

$$\tan^{-1}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots \quad for\,|x| < 1$$

$$S\mathbf{c} = w\mathbf{x} + \mathbf{e}$$

$$\mathbf{x} = \left\lceil \frac{S\mathbf{c}}{w} \right\rfloor$$

```
In [2]: T = get_T(n)

In [3]: c,S = encrypt_via_switch(x,w,m,n,T)

In [4]: decrypt(c,S,w)
Out[4]: array([0, 1, 2, 5])

In [5]: decrypt(c + c, S,w)
Out[5]: array([ 0,  2,  4, 10])

In [6]: decrypt(c * 10,S,w)
Out[6]: array([ 0, 10, 20, 50])
```

```
 Iter:110 Encrypted Loss:59148014 Decrypted Loss:1.763 Alpha:0.015
 Iter:120 Encrypted Loss:58934571 Decrypted Loss:1.757 Alpha:0.015
 Iter:130 Encrypted Loss:58724873 Decrypted Loss:1.75 Alpha:0.0155
 Iter:140 Encrypted Loss:58516008 Decrypted Loss:1.744 Alpha:0.015
 Iter:150 Encrypted Loss:58307663 Decrypted Loss:1.739 Alpha:0.015
 Iter:160 Encrypted Loss:58102049 Decrypted Loss:1.732 Alpha:0.015
 Iter:170 Encrypted Loss:57863091 Decrypted Loss:1.725 Alpha:0.015
 Iter:180 Encrypted Loss:55470158 Decrypted Loss:1.653 Alpha:0.015
 Iter:190 Encrypted Loss:54650383 Decrypted Loss:1.629 Alpha:0.015
 Iter:200 Encrypted Loss:53838756 Decrypted Loss:1.605 Alpha:0.015
 Iter:210 Encrypted Loss:51684722 Decrypted Loss:1.541 Alpha:0.015
 Iter:220 Encrypted Loss:54408709 Decrypted Loss:1.621 Alpha:0.015
 Iter:230 Encrypted Loss:54946198 Decrypted Loss:1.638 Alpha:0.015
 Iter:240 Encrypted Loss:54668472 Decrypted Loss:1.63 Alpha:0.0155
 Iter:250 Encrypted Loss:55444008 Decrypted Loss:1.653 Alpha:0.015
 Iter:260 Encrypted Loss:54094286 Decrypted Loss:1.612 Alpha:0.015
 Iter:270 Encrypted Loss:51251831 Decrypted Loss:1.528 Alpha:0.015
 Iter:276 Encrypted Loss:24543890 Decrypted Loss:0.732 Alpha:0.015
 Final Prediction:
True Pred:[0] Encrypted Prediction:[-3761423723.0718255 0.0] Decrypted Prediction:[-0.112]
True Pred:[1] Encrypted Prediction:[24204806753.166267 0.0] Decrypted Prediction:[ 0.721]
True Pred:[1] Encrypted Prediction:[23090462896.17028 0.0] Decrypted Prediction:[ 0.688]
True Pred:[0] Encrypted Prediction:[1748380342.4553354 0.0] Decrypted Prediction:[ 0.052]
```

➢ CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy

ICML-2016

$$CryptoNets = homomorphic\ encryption + neural\ networks$$

Several common functions can be computed at the nodes:

1. ***Weighted-Sum*** (convolution layer)

2. *Max Pooling* $\Rightarrow max(x_1, \dots, x_n) = \lim_{d\to\infty}\left(\sum_i {x_i}^d\right)^{\frac{1}{d}}, d = 1$

   Compute the maximal value of some of the components of the feeding layer.

3. *Mean Pooling* $\Rightarrow mean(x_1, \dots, x_n) = \lim_{d\to\infty}\left(\sum_i {x_i}^d\right)^{\frac{1}{d}}, d = 1$

   Compute the average value of some of the components of the feeding layer.

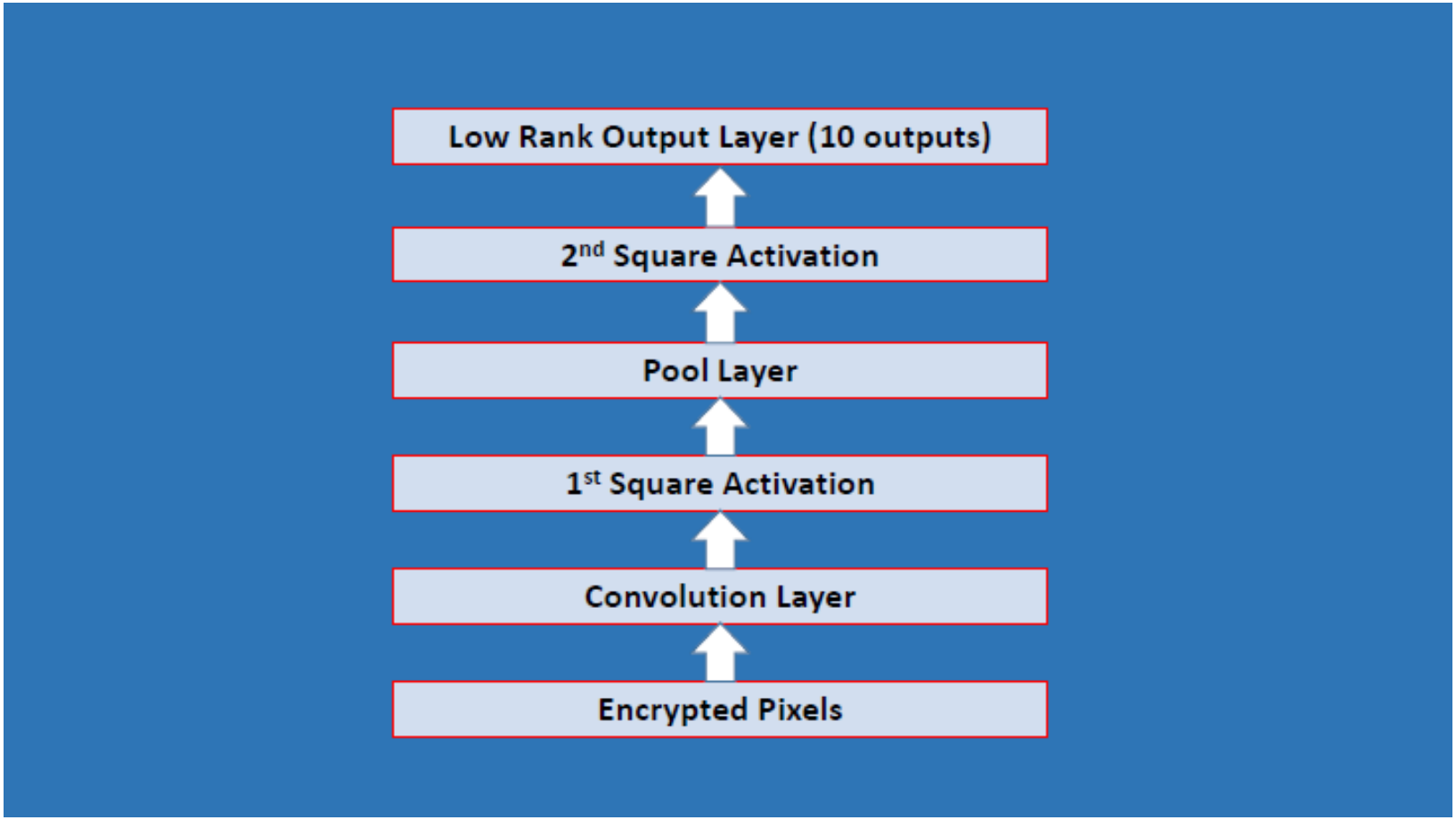
4. *Sigmoid* $\Rightarrow square\ function: sqr(z) := z^2.$

   $z \mapsto 1/(1 + exp(-z))$

5. *Rectified Linear* $\Rightarrow square\ function: sqr(z) := z^2.$

   $z \mapsto max(0, z)$

Livni et al. (2014) have recently suggested a theoretical analysis of the problem of learning neural networks with polynomial activation functions and devoted much of their study to the square activation function.


Low Rank Output Layer (10 outputs)
2nd Square Activation
Pool Layer
1st Square Activation
Convolution Layer
Encrypted Pixels

MNIST dataset

60,000 images ; 10 response.

28x28 pixel, 0-255.

99% (105 out of 10,000 test examples)

Each image requires:

$$28 \times 28 \times 4096 \times 5 \times 24\text{bytes} = 367.5MB$$

The response of the message size:

$$10 \times 4096 \times 5 \times 24\text{bytes} = 4.7MB$$

Table 3. Message sizes of CryptoNet for MNIST

	Message size	Size per instance
Owner → Cloud	367.5 MB	91.875 KB
Cloud → Owner	4.7 MB	1.17 KB

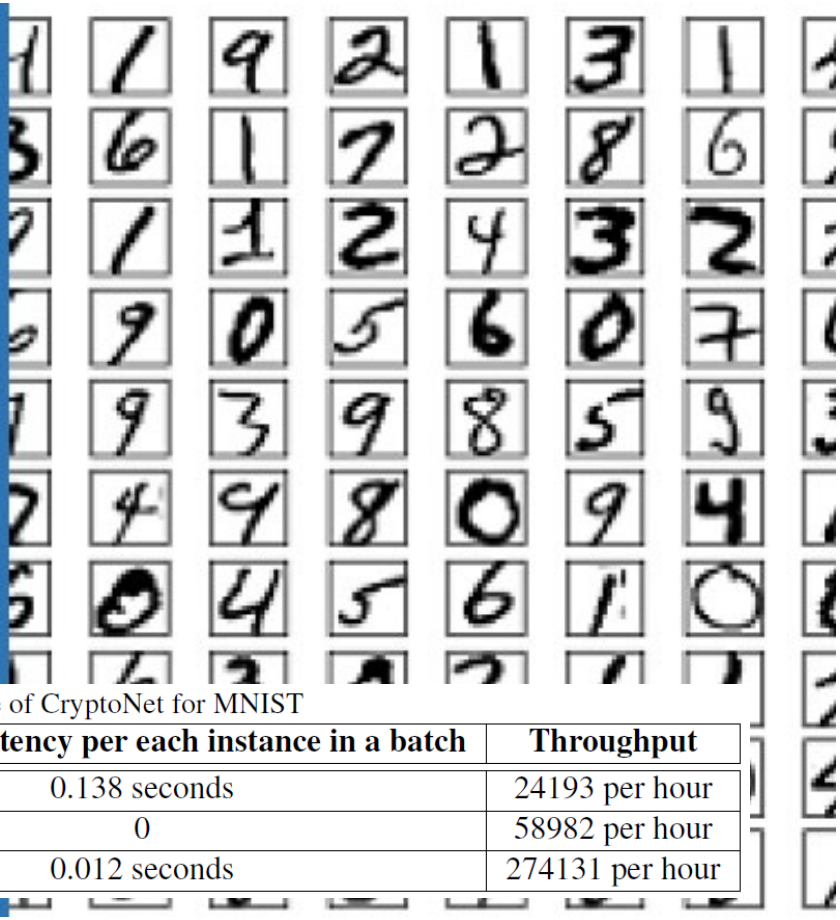


Table 2. The performance of CryptoNet for MNIST

Stage	Latency	Additional latency per each instance in a batch	Throughput
Encoding+Encryption	44.5 seconds	0.138 seconds	24193 per hour
Network application	250 seconds	0	58982 per hour
Decryption+Decoding	3 seconds	0.012 seconds	274131 per hour

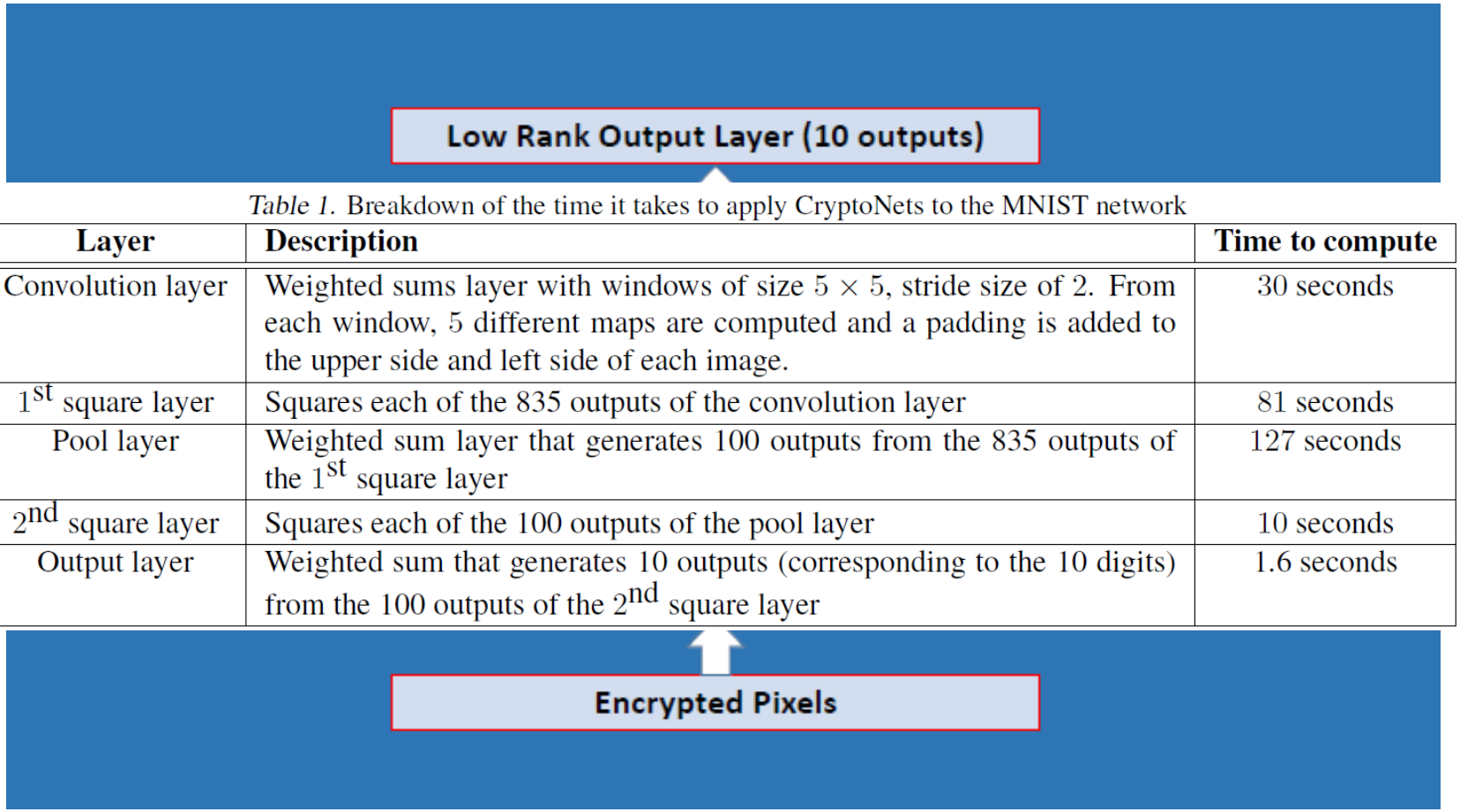


Table 1. Breakdown of the time it takes to apply CryptoNets to the MNIST network

Layer	**Description**	**Time to compute**
Convolution layer	Weighted sums layer with windows of size 5×5, stride size of 2. From each window, 5 different maps are computed and a padding is added to the upper side and left side of each image.	30 seconds
1^{st} square layer	Squares each of the 835 outputs of the convolution layer	81 seconds
Pool layer	Weighted sum layer that generates 100 outputs from the 835 outputs of the 1^{st} square layer	127 seconds
2^{nd} square layer	Squares each of the 100 outputs of the pool layer	10 seconds
Output layer	Weighted sum that generates 10 outputs (corresponding to the 10 digits) from the 100 outputs of the 2^{nd} square layer	1.6 seconds

➢ GELU-Net: A Globally Encrypted, Locally Unencrypted Deep Neural Network for Privacy-Preserved Learning

IJCAI-2018

GELU-Net = *Globally Encrypted* + *Locally Unencrypted*

For an n-layer network, w_i and b_i are the weights and biases corresponding to the i-th layer.

The forward propagation calculates weighted-sums

$$w_1 a_{i-1} + b$$

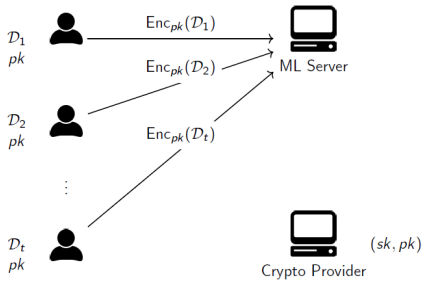
The weight and bias are then updated based on the backpropagation

$$w_i = w_i - \eta \Delta w_i, b_i = b_i - \eta \Delta b_i$$

The gradients Δw_i and Δb_i are calculated as follows:

$$\Delta w_i = a_{i-1}\delta_i, \Delta b_i = \delta_i$$

Where $\delta_i = \delta_{i-1} w_{i+1} a_i (1 - a_i)$ for sigmoid activation and $\delta_{i-1} = y - t$ for cross-entropy loss.



Scheme	Communication	Crypto	Activation	Total
Square	0	0	90.6	90.6
5-th order	0	0	1619.6	1619.6
GELU-Net	5	3.7	0.2	**8.9**

Table 1: Computation time of activation in different schemes (ms).

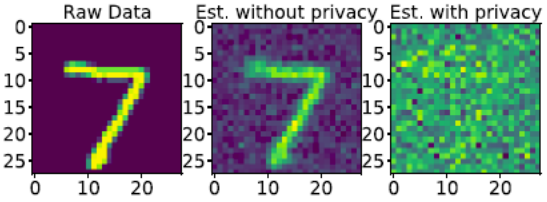


Figure 2: Server reconstruction with/without gradient protection.

Algorithm 1: Privacy Preserved Forward Propagation

Input: *Client*: Data $\boldsymbol{x}$, set gradient/random accumulator $\Delta \boldsymbol{w}_i^c, \Delta \boldsymbol{b}_i^c, \boldsymbol{r}_i^{wc}, \boldsymbol{r}_i^{bc}$ to $\mathbf{0}$. *Server*: Initialize model, training bound d_{max}. Record initial parameters $\boldsymbol{w}_i^{\mathrm{I}}$ and $\boldsymbol{b}_i^{\mathrm{I}}$ (Section 4.4)

Output: Softmax output $\boldsymbol{y}$

for $d = 1, 2, 3, \cdots, d_{max}$ **do**
- Client: $\boldsymbol{a}_0 \leftarrow \boldsymbol{x}_d$
- **for** $i = 1, 2, \cdots, n-1$ **do**
  - Client: Encrypt $\boldsymbol{a}_{i-1}$ with PK_c as $[\![\boldsymbol{a}_{i-1}]\!]_c$ and send it to server
  - Server: Compute $[\![\widetilde{\boldsymbol{z}}_i]\!]_c \leftarrow (\widetilde{\boldsymbol{w}}_i \otimes [\![\boldsymbol{a}_{i-1}]\!]_c) \oplus \widetilde{\boldsymbol{b}}_i$ and send $[\![\widetilde{\boldsymbol{z}}_i]\!]_c$ to client
  - Client: Decrypt $[\![\widetilde{\boldsymbol{z}}_i]\!]_c$ with SK_c, call Algorithm 4 to remove randomness in $\widetilde{\boldsymbol{z}}_i$
  - **if** $i = n-1$ **then**
    - Client: $\boldsymbol{y} \leftarrow e^{\boldsymbol{z}_i} / \sum_j e^{\boldsymbol{z}_i}$ (softmax)
  - **else**
    - Client: $\boldsymbol{a}_i \leftarrow f(\boldsymbol{z}_i)$ (next layer)
- Call Algorithm 2 for backpropagation

Algorithm 2: Privacy Preserved Backpropagation

Server: Encrypt $\frac{1}{\eta}$ with PK_s as $[\![\frac{1}{\eta}]\!]_s$ and send it to client

Client: For the last layer $(n-1)$, compute $\boldsymbol{\delta}_{n-1} \leftarrow \boldsymbol{y} - \boldsymbol{t}$, $\Delta \boldsymbol{w}_{n-1} \leftarrow \boldsymbol{a}_{n-2}\boldsymbol{\delta}_{n-1}$, $\Delta \boldsymbol{b}_{n-1} \leftarrow \boldsymbol{\delta}_{n-1}$,

$\Delta \boldsymbol{w}_{n-1}^c \leftarrow \Delta \boldsymbol{w}_{n-1}^c + \Delta \boldsymbol{w}_{n-1}, \Delta \boldsymbol{b}_{n-1}^c \leftarrow \Delta \boldsymbol{b}_{n-1}^c + \Delta \boldsymbol{b}_{n-1}$

for $i = n-2, n-3, \cdots, 1$ **do**
- Client: Encrypt $\boldsymbol{\delta}_{i+1}$ with PK_c as $[\![\boldsymbol{\delta}_{i+1}]\!]_c$ and send it to server
- Server: Compute $[\![\widetilde{\boldsymbol{q}}_{i+1}]\!]_c \leftarrow [\![\boldsymbol{\delta}_{i+1}]\!]_c \otimes \widetilde{\boldsymbol{w}}_{i+1}$, and send it to client
- Client: Decrypt $[\![\widetilde{\boldsymbol{q}}_{i+1}]\!]_c$ with SK_c, call Algorithm 4 to remove randomness in $\widetilde{\boldsymbol{q}}_{i+1}$,and calculate $\boldsymbol{\delta}_i \leftarrow \boldsymbol{\delta}_{i+1}\boldsymbol{w}_{i+1}\boldsymbol{a}_i(1-\boldsymbol{a}_i)$, $\Delta \boldsymbol{w}_i \leftarrow \boldsymbol{a}_{i-1}\boldsymbol{\delta}_i$, $\Delta \boldsymbol{b}_i \leftarrow \boldsymbol{\delta}_i$.
- Update $\Delta \boldsymbol{w}_i^c \leftarrow \Delta \boldsymbol{w}_i^c + \Delta \boldsymbol{w}_i, \Delta \boldsymbol{b}_i^c \leftarrow \Delta \boldsymbol{b}_i^c + \Delta \boldsymbol{b}_i$
- **if** $d < d_{max}$ **then**
  - Client: Call Algorithm 3 to mask $\Delta \boldsymbol{w}_i$ and $\Delta \boldsymbol{b}_i$ as $[\![\Delta \widetilde{\boldsymbol{w}}_i]\!]_s$ and $[\![\Delta \widetilde{\boldsymbol{b}}_i]\!]_s$, send to server
  - Server: Decrypt $[\![\Delta \widetilde{\boldsymbol{w}}_i]\!]_s$ and $[\![\Delta \widetilde{\boldsymbol{b}}_i]\!]_s$ with SK_s, and update $\widetilde{\boldsymbol{w}}_i \leftarrow \widetilde{\boldsymbol{w}}_i - \eta \Delta \widetilde{\boldsymbol{w}}_i$ and $\widetilde{\boldsymbol{b}}_i \leftarrow \widetilde{\boldsymbol{b}}_i - \eta \Delta \widetilde{\boldsymbol{b}}_i$

Call Algorithm 1 for the next iteration or call Algorithm 5 to update model parameter on server when finish

Approach	Computation	Communication
GELU-Net	$\mathcal{O}(nm^2p)$	$\mathcal{O}(nm^2)$
BGN-Net	$\mathcal{O}(Cnmb)$	$\mathcal{O}(nm^2)$
ElGamal-Net	$\mathcal{O}(Z^2m^3ne)$	$\mathcal{O}(Z^2m^2n)$

Table 2: Complexity comparison (per iteration).

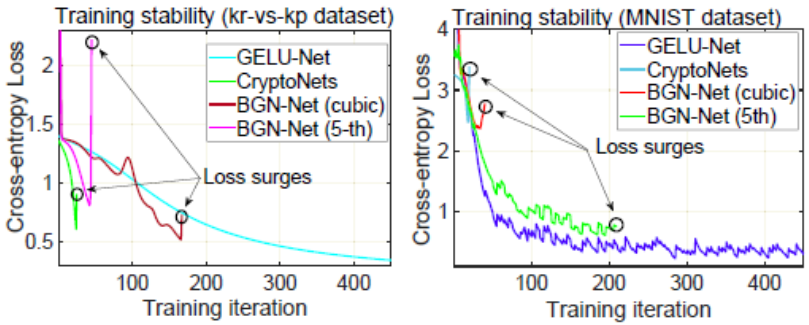


Figure 3: Training stability of different schemes.

Architecture	Time (s)	Accuracy
GELU-Net (Conv-1)	**67.5±2.8**	0.936±0.006
CryptoNets (Conv-1)	1271.8±1.9	0.909±0.002
GELU-Net (LeNet-5)	**85.5±2.1**	0.989±0.001
CryptoNets (LeNet-5)	3009.6±1.7	0.967±0.003

Table 4: Computation speed in different networks (s).

Architecture	Cloud-Local	Cloud-Cloud	Local-Local
Dense(12)	0.381±0.002	**0.156±0.005**	0.281±0.01
Dense(300)	147.5±9.8	**59.7±1.9**	107.4±3.5
Conv1	91.3±5.4	**37.5±1.6**	67.5±2.8
LeNet-5	126.7±6.3	**47.5±1.2**	85.5±2.1

Table 5: GELU-Net speed in different environments (s).

Not currently feasible for some models

Decision Trees

- Very powerful
- Need to always evaluate entire tree homomorphically
- Conditional statements based on integer comparison hard
  - Bit-wise encoding: comparisons easy, arithmetic hard
  - Integer-wise encoding: comparisons hard, arithmetic easy
  - Trees require both comparisons and arithmetic
- Could some pre-computation help with evaluating tree?
- Simplify by assuming structure of tree public but thresholds not?

- Regression: (Linear Regression, Ridge Regression)

  Privacy-preserving ridge regression on hundreds of millions of records. 2013
  Privacy-preserving linear regression on distributed data. 2017
  Using homomorphic encryption for large scale statistical analysis. 2012
  Fast and secure linear regression and biometric authentication with security update. 2015

- Classification: (Linear Means Classifier, Naive Bayes, Decision Trees, SVM)

  Ml confidential: Machine learning on encrypted data. 2012
  Machine learning classification over encrypted data. 2015

- Clustering: (K-Means, K-NN)

  Privacy preserving k-means clustering in multi-party environment. 2007
  Privacy-preserving k-means clustering over vertically partitioned data. 2003

- Neural networks: (Crypto-Nets)

  Crypto-nets: Neural networks over encrypted data. 2014

Differential Privacy

Our world is increasingly data driven

Source (http://www.agencypja.com/site/assets/files/1826/marketingdata-1.jpg)

Why 'Anonymous' Data Sometimes Isn't

By Bruce Schneier 12.13.07

Last year, Netflix published 10 million movie rankings by 500,000 customers, as part of a challenge for people to come up with better recommendation systems than the one the company was using.

The Scientist » The Nutshell

"Anonymous" Genomes Identified

The names and addresses of people participating in the Personal Genome Project can be easily tracked down despite such data being left off their online profiles.

By Dan Cossins | May 3, 2013

- Real world deployments of differential privacy

– OnTheMap  United States Census Bureau

RAPPOR  chrome

High-dimensional Data is Unique

Example: UCSD Employee Salary Table

Position	Gender	Department	Ethnicity	Salary
Faculty	Female	CSE	SE Asian	-

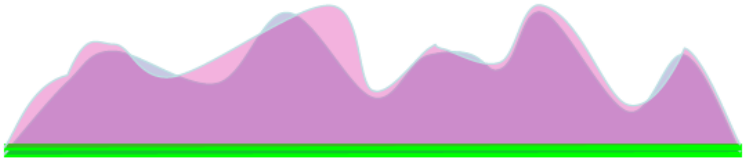
One employee (Kamalika) fits description!

Differential Privacy

E.g., want to release average while preserving privacy.

High level idea:

- What we want is a protocol that has a probability distribution over outputs:



such that if person i changed their input from x_i to any other allowed x_i', the relative probabilities of any output do not change by much.

Differential Privacy

Ingredients

- Input space X (with symmetric neighbouring relation $\simeq$)
- Output space Y (with σ-algebra of measurable events)
- Privacy parameter $\varepsilon \geqslant 0$

Differential Privacy **[Dwork et al., 2006, Dwork, 2006]**

A randomized mechanism $\mathcal{M} : X \to Y$ is ε-differentially private if for all neighbouring inputs $x \simeq x'$ and for all sets of outputs $E \subseteq Y$ we have

$$\mathbb{P}[\mathcal{M}(x) \in E] \leqslant e^{\varepsilon} \mathbb{P}[\mathcal{M}(x') \in E]$$

Intuitions behind the definition:

- The neighbouring relation $\simeq$ captures *what* is protected
- The probability bounds capture *how much* protection we get

research

“你有没有来过北海道”回答为“是”或者“否”写在纸条上传给统计分析者，回答的青蛙需要掷骰子，如果骰子显示1，3，5，即单数，那么如实回答，如果是2，4，6，即双数的话，那么再掷一次骰子，然后不管事实是什么，显示单数就写“是”，双数就写“否”。

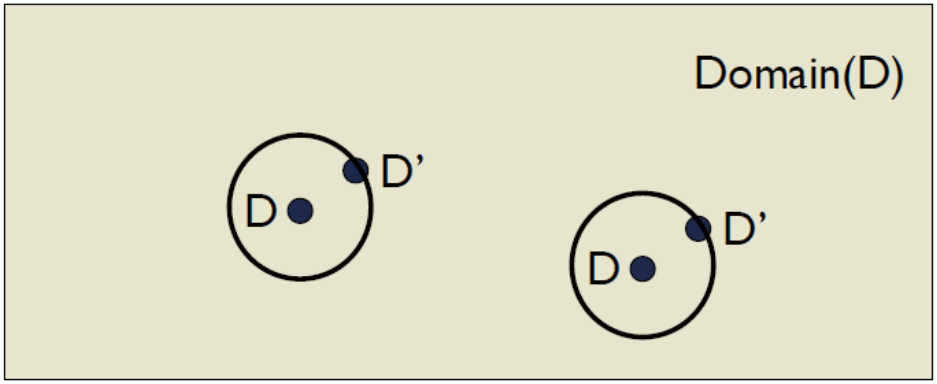
The Global Sensitivity Method [DMNS06]

Given: A function f, sensitive dataset D

Define: dist(D, D') = #records that D, D' differ by

Global Sensitivity of f:

$$S(f) = \max_{dist(D, D') = 1} | f(D) - f(D')|$$



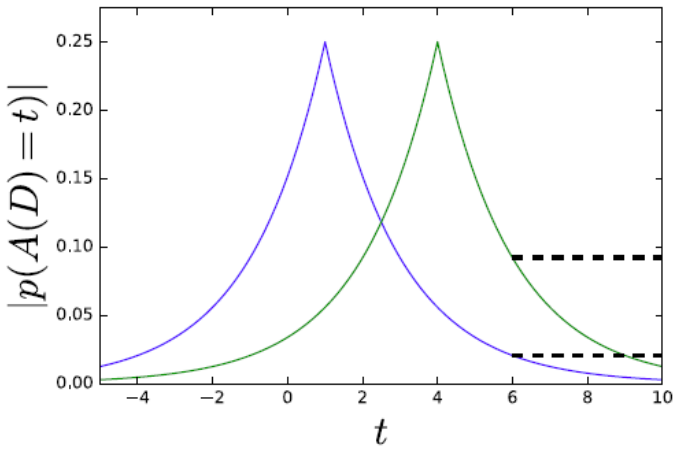
Laplace Mechanism

Global Sensitivity of f is S(f) = $\max_{\text{dist}(D, D') = 1}$ | f(D) - f(D')|

Output $f(D) + Z$, where

$$Z \sim \frac{S(f)}{\epsilon}\text{Lap}(0, 1)$$

ϵ-differentially private



Laplace distribution:

$$p(z|\mu, b) = \frac{1}{2b}\exp\left(-\frac{|z - \mu|}{b}\right)$$

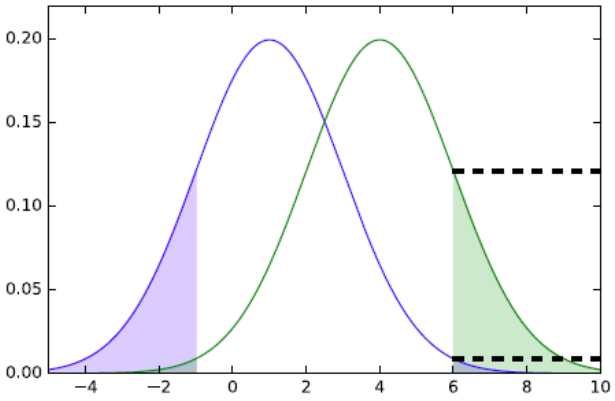
Gaussian Mechanism

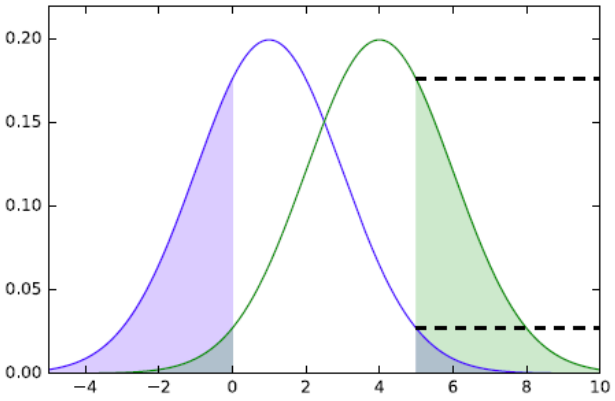
Global Sensitivity of f is S(f) = $\max_{\text{dist}(D, D') = 1} |f(D) - f(D')|$

Output $f(D) + Z$, where

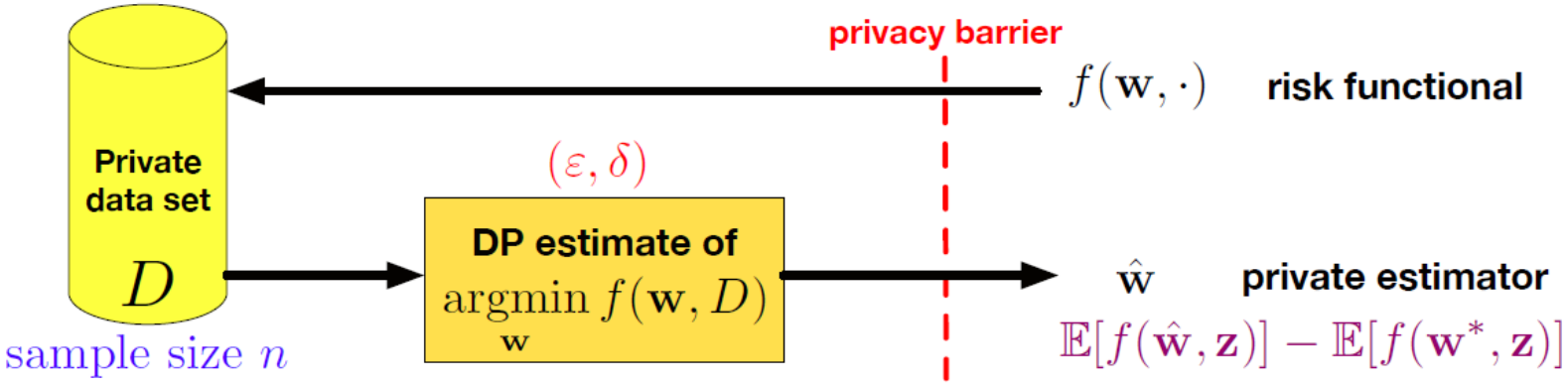
$$Z \sim \frac{S(f)}{\epsilon}\mathcal{N}(0, 2\ln(1.25/\delta))$$

(ϵ, δ)-differentially private





Privacy and accuracy make different assumptions about the data



Privacy – differential privacy makes *no assumptions on the data distribution*: privacy holds unconditionally.

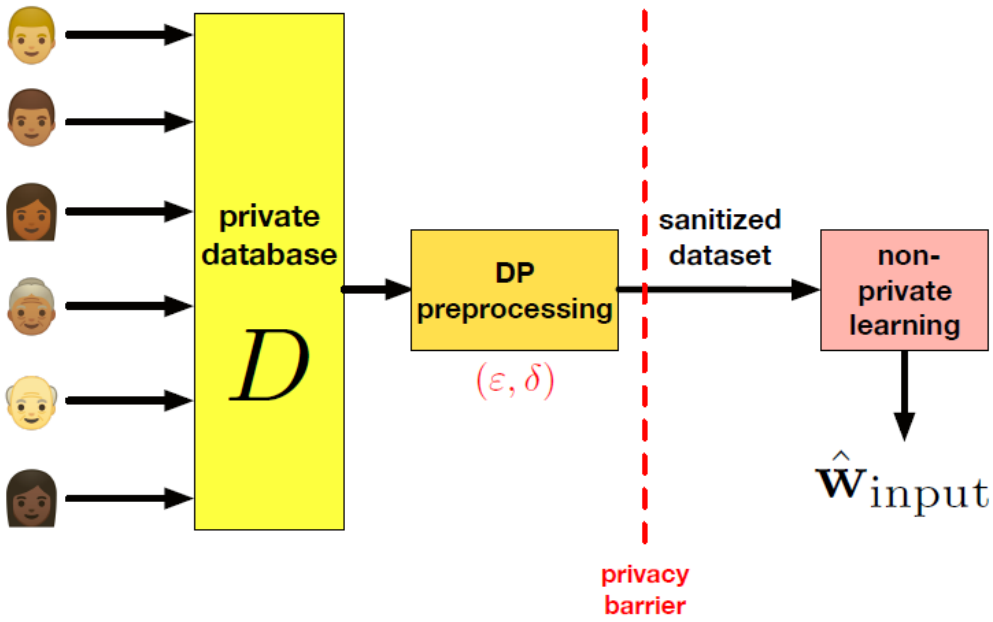
Accuracy – accuracy measured *w.r.t a "true population distribution"*: expected excess statistical risk.

Revisiting ERM

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{w}, (\mathbf{x}_i, y_i)) + \lambda R(\mathbf{w})$$

- Learning using (convex) optimization uses three steps:
  1. read in the data — **input perturbation**
  2. form the objective function — **objective perturbation**
  3. perform the minimization — **output perturbation**
- We can try to introduce privacy in each step!

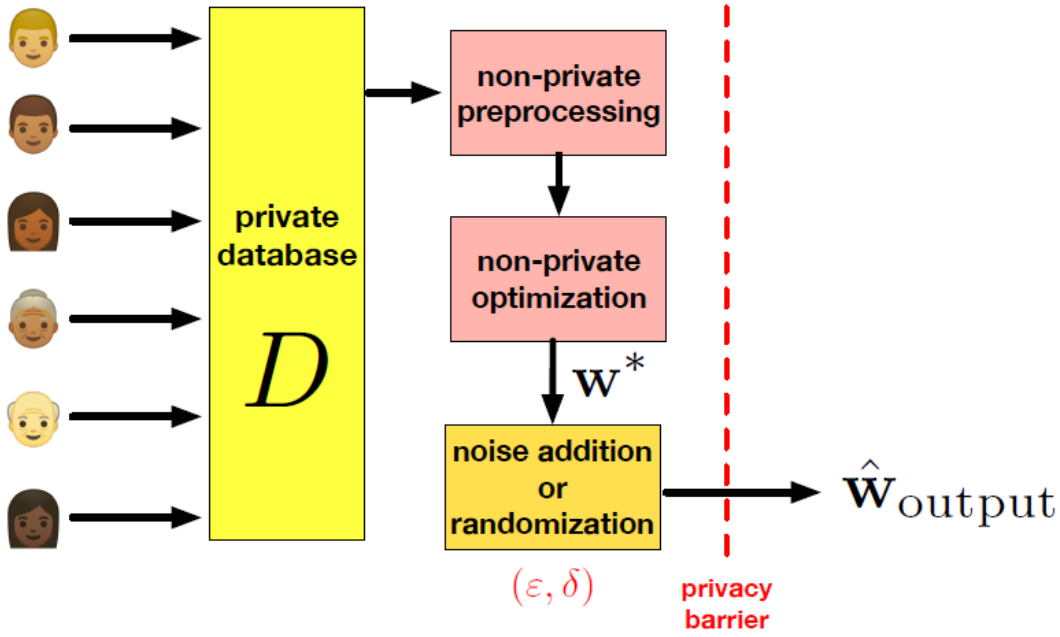
Input Perturbation



- Input perturbation: add noise to the input data.
- Advantages: easy to implement, results in reusable sanitized data set.

[DJW13,FTS17]

Output Perturbation

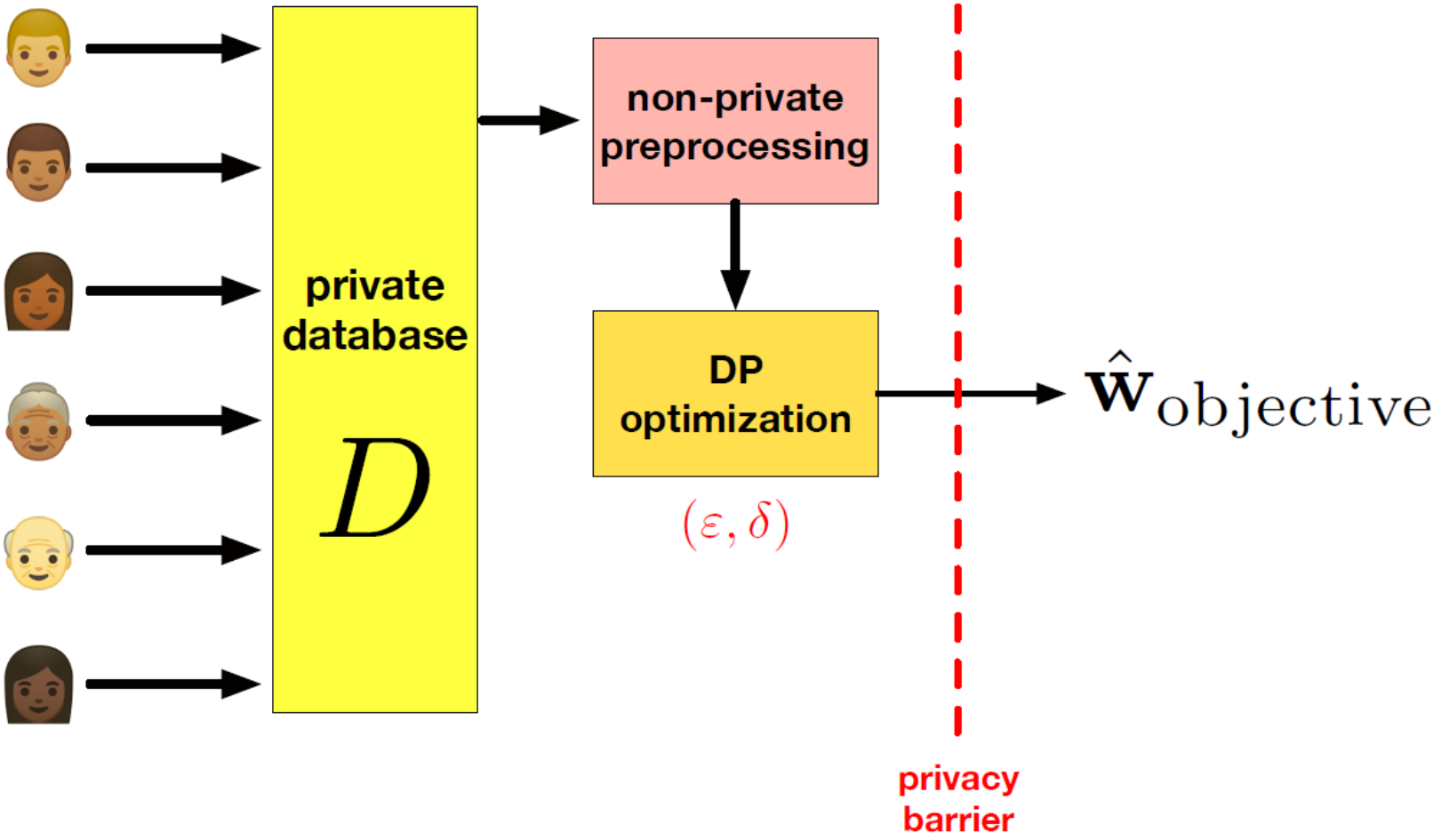


- Compute the minimizer and add noise.
- Does not require re-engineering baseline algorithms

Noise depends on the sensitivity of the argmin.

[CMS11, RBHT12]

Objective Perturbation



Objective Perturbation

$$J(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}, (\mathbf{x}_i, y_i)) + \lambda R(\mathbf{w})$$

A. Add a random term to the objective:

$$\hat{\mathbf{w}}_{\text{priv}} = \underset{\mathbf{w}}{\operatorname{argmin}} \left(J(\mathbf{w}) + \mathbf{w}^\top \mathbf{b} \right)$$

B. Do a randomized approximation of the objective:

$$\hat{\mathbf{w}}_{\text{priv}} = \underset{\mathbf{w}}{\operatorname{argmin}} \hat{J}(\mathbf{w})$$

Randomness depends on the sensitivity properties of *J(w)*.

[CMS11, ZZX+12]

Scaling up private optimization

- Large data sets are challenging for optimization:
  ⇒ batch methods not feasible

- Using more data can help our tradeoffs look better:
  ⇒ better privacy and accuracy

- Online learning involves multiple releases:
  ⇒ potential for more privacy loss

Goal: guarantee privacy using the optimization *algorithm*.

Non-private SGD

$$J(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}, (\mathbf{x}_i, y_i)) + \lambda R(\mathbf{w})$$

$\mathbf{w}_0 = \mathbf{0}$

For $t = 1, 2, \ldots, T$

$i_t \sim \text{Unif}\{1, 2, \ldots, n\}$

$\mathbf{g}_t = \nabla \ell(\mathbf{w}_{t-1}, (\mathbf{x}_{i_t}, y_{i_t})) + \lambda \nabla R(\mathbf{w}_{t-1})$

$\mathbf{w}_t = \Pi_{\mathcal{W}}(\mathbf{w}_{t-1} - \eta_t \mathbf{g}_t)$

$\hat{\mathbf{w}} = \mathbf{w}_T$

- select a random data point
- take a gradient step

Private SGD with noise

$$J(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{w}, (\mathbf{x}_i, y_i)) + \lambda R(\mathbf{w})$$

$\mathbf{w}_0 = \mathbf{0}$

For $t = 1, 2, \ldots, T$

$i_t \sim \text{Unif}\{1, 2, \ldots, n\}$

$\mathbf{z}_t \sim p_{(\varepsilon,\delta)}(\mathbf{z})$

$\hat{\mathbf{g}}_t = \mathbf{z}_t + \nabla \ell(\mathbf{w}_{t-1}, (\mathbf{x}_{i_t}, y_{i_t})) + \lambda \nabla R(\mathbf{w}_{t-1})$

$\mathbf{w}_t = \Pi_{\mathcal{W}}(\mathbf{w}_{t-1} - \eta_t \hat{\mathbf{g}}_t)$

$\hat{\mathbf{w}} = \mathbf{w}_T$

- select random data point
- add noise to gradient

[SCS15]

Making DP-SGD more practical

"SGD is robust to noise"

- True up to a point — for small epsilon (more privacy), the gradients can become too noisy.
- **Solution 1:** more iterations ([BST14]: need $O(n^2)$)
- **Solution 2:** use standard tricks: mini-batching, etc. [SCS13]
- **Solution 3:** use better analysis to show the privacy loss is not so bad [BST14][ACG+16]

Differential privacy in practice

Google: RAPPOR for tracking statistics in Chrome.

Apple: various iPhone usage statistics.

Census: 2020 US Census will use differential privacy.

mostly focused on count and average statistics

Challenges for machine learning applications

Differentially private ML is complicated because real ML algorithms are complicated:

- Multi-stage pipelines, parameter tuning, etc.
- Need to "play around" with the data before committing to a particular pipeline/algorithm.
- "Modern" ML approaches (= deep learning) have many parameters and less theoretical guidance.

Summary

Encryption

All

看山不是山

privacy

One

桃花依旧笑春风

THX~

百万富翁问题

假设富翁王有i亿资产，李有j亿资产。王选取一个公钥，使得李可以传递加密的信息。

首先，李选取一个随机的大整数x，把x用王的公钥加密，得到密文K。 李发送$K-j$给王。

王收到密文$c=K-j$之后， 对$c+1, c+2, \dots, c+10$进行解密，得到十个数字。再选取一个适当大小的素数p，把这十个数字除以p的余数记作$d_1, \dots, d_{10}$.

注意：这十个数字看起来应该是完全随机的，关键是等式$d_j = x \bmod p$,成立。

最后，王对这一串数字作如下操作：前面i个数不动，后面的数字每个$+1$，然后发回给李。

这样一通复杂的操作之后，李检查第j个数字。如果等于$x \bmod p$的话，说明这个数字没有被$+1$，所以$i \geq j$. 反之，则 $i < j$。

这个过程的绝妙之处在于：在协议完成之后，王不知道j的具体值，而李也不知道i的值， 但是双方都知道谁的财富更多，这就是安全多方计算。

一般来说，在甲只知道x，乙只知道y的情况下，双方可以合作计算一个函数$f(x,y)$。协议完成时，只有函数值是公开的，而彼此都不知道对方的输入值。

Broadly speaking, an encryption scheme consists of:

- Unencrypted object, $m \in M$, referred to as a *message*.
  - M is the *message space*.
- Encrypted version, $c \in C$, referred to as a *cipher text*.
  - C is the *cipher text space*.
- Single $(k_s) \in K_s$, or pair $(k_s, k_p) \in K_s \times K_p$, of 'keys'.
  - Single key means secret key scheme;
  - Pair of keys means public key scheme.
- Injective map, $\mathsf{Enc} : K_p \times M \to C$.
  - not necessarily a function, message can encrypt to different cipher texts.
- Surjective function, $\mathsf{Dec} : K_s \times C \to M$.
- Enc and Dec satisfy:

$$m = \mathsf{Dec}(k_s, \mathsf{Enc}(k_p, m)) \quad \forall\, m \in M$$

Addition and multiplication seem pretty limiting, why all the excitement if this is all that is possible?

Note that if $M = \mathrm{GF}(2)$, then:

- $+ \equiv \veebar$, i.e. XOR, 'exclusive or'
- $\times \equiv \wedge$, i.e. AND, 'and'

Moreover, *any* electronic logic gate can be constructed using only XOR and AND gates. Therefore, theoretically any operation on a computer can be performed encrypted.

Most cryptography schemes are 'brittle' in that we can't manipulate the message contained in the mathematical vault: must decrypt to compute, then encrypt the result. i.e. seems only useful for shipping round static data!

In other words, if

$$c_1 := \mathsf{Enc}(k_p, m_1)$$
$$c_2 := \mathsf{Enc}(k_p, m_2)$$

then in general, for a given function $g(\cdot,\cdot)$, $\nexists\, f(\cdot,\cdot)$ (not requiring k_s) such that

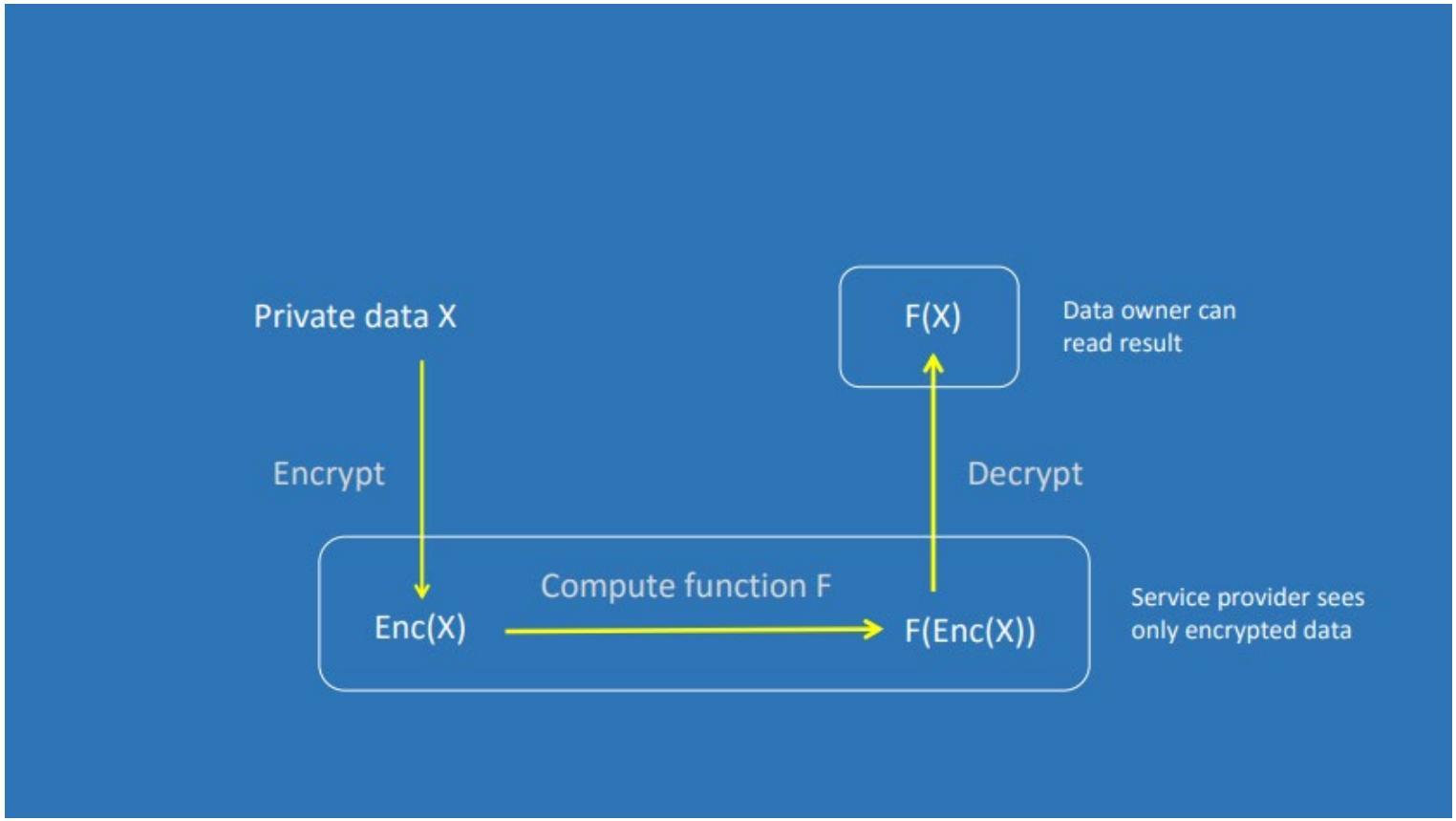
$$\mathsf{Dec}(k_s, f(c_1, c_2)) = g(m_1, m_2) \quad \forall\, m_1, m_2 \in M$$

Recall RSA from the introduction: it is in fact a homomorphic encryption scheme!

$$\mathcal{F}_M = \{\times\}, \mathcal{F}_C = \{\times\}$$

$$\begin{aligned}
\mathsf{Enc}(k_p, m_1) \times \mathsf{Enc}(k_p, m_2) &= (m_1^e \mod n) \times (m_2^e \mod n) \\
&= (m_1 m_2)^e \mod n \\
&= \mathsf{Enc}(k_p, m_1 m_2)
\end{aligned}$$

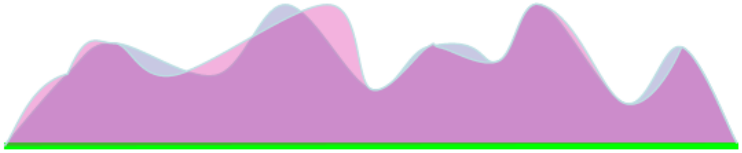
Final equality indicates a lack of semantic security, so actually RSA is not great when we want to encrypt plain old integer data as it will be very vulnerable to chosen plaintext attack.


Private data X
F(X)
Data owner can read result
Encrypt
Decrypt
Compute function F
Enc(X)
F(Enc(X))
Service provider sees only encrypted data

Differential Privacy

High level idea:

- What we want is a protocol that has a probability distribution over outputs:



such that if person i changed their input from x_i to any other allowed x_i', the relative probabilities of any output do not change by much.

- This would effectively allow that person to pretend their input was any other value they wanted.

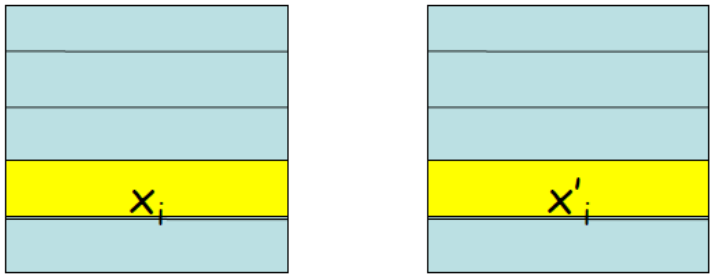
Bayes rule: $\frac{\Pr(x_i|output)}{\Pr(x_i'|output)} = \frac{\Pr(output|x_i)}{\Pr(output|x_i')} \cdot \frac{\Pr(x_i)}{\Pr(x_i')}$

(Posterior ≈ Prior)

Differential Privacy: Definition

It's a property of a protocol A which you run on some dataset X producing some output A(X).

- A is α-*differentially private* if for any two neighbor datasets S, S' (differ in just one element $x_i \rightarrow x_i'$),



for all outcomes v,

$$e^{-\alpha} \leq \Pr(A(S)=v)/\Pr(A(S')=v) \leq e^{\alpha}$$

$\approx 1-\alpha$

probability over randomness in A

$\approx 1+\alpha$

Differential Privacy: Definition

It's a property of a protocol A which you run on some dataset X producing some output A(X).

- A is ϵ-*differentially private* if for any two neighbor datasets S, S' (differ in just one element $x_i \rightarrow x_i'$),

View as model of plausible deniability

If your real input is x_i and you'd like to pretend was x_i', somebody looking at the output of A can't tell, since for any outcome v, it was nearly just as likely to come from S as it was to come from S'.

for all outcomes v,

$$e^{-\alpha} \leq \Pr(A(S)=v)/\Pr(A(S')=v) \leq e^{\alpha}$$

$\approx 1-\alpha$

probability over randomness in A

$\approx 1+\alpha$

Differential Privacy

- An individual is **in** or **out** of the database should make **little difference** of the analytical output. *Dwork, C. (2006). Differential Privacy*