CS545 – Machine Learning for Signal Processing

Privacy-Preserving Machine Learning

(+ more by request)

14 November 2023

University of Illinois at Urbana-Champaign

Today's lecture

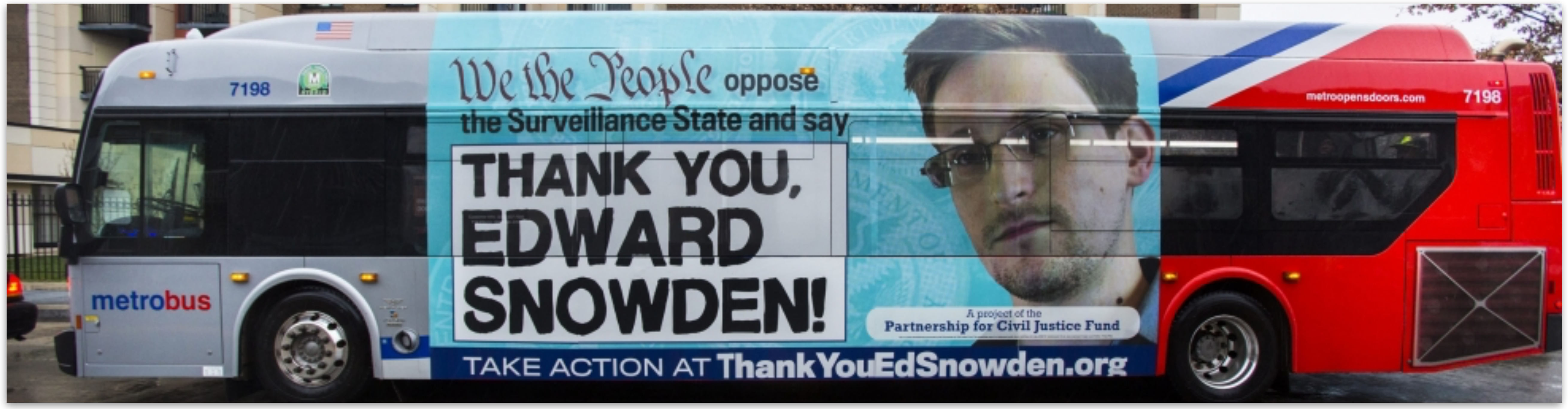
- Secure Multiparty Computations
  - Some cryptography basics
  - Applying SMC to MLSP
- Other pending tidbits

A problem

- Machine learning can be a great service
  - Give me your data, I'll give you some insight
- But, you can't ask everyone for their data
  - Sending your voice mail for transcription?
  - Camera feed from your house for security?
  - Sharing medical data with your phone?

What would be nice

- In a perfect world there would be absolute trust
  - But do you really trust anyone with your data?

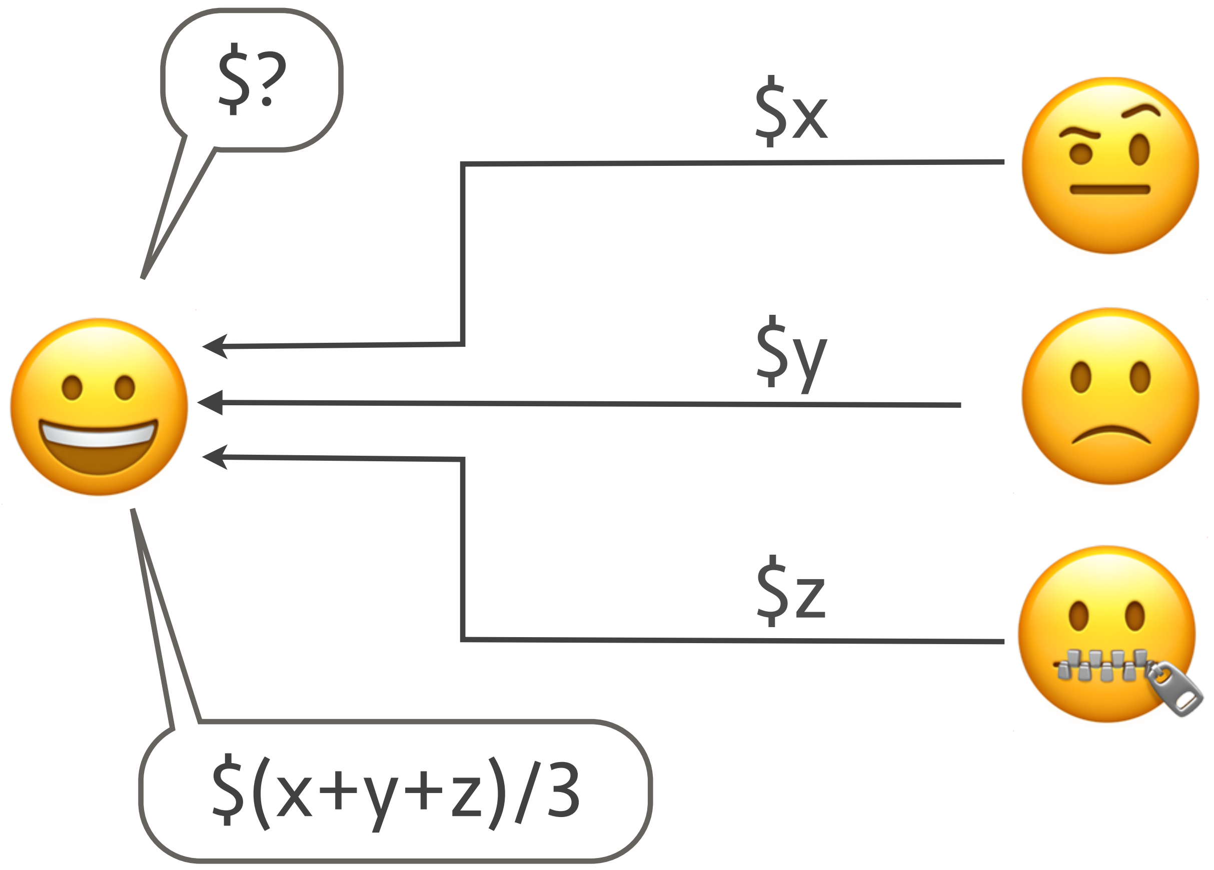


- In the real world
  - We want to ensure the privacy of our data
  - Others want to ensure privacy of their algorithms & data
    - Can these constraints co-exist?

Starting simple

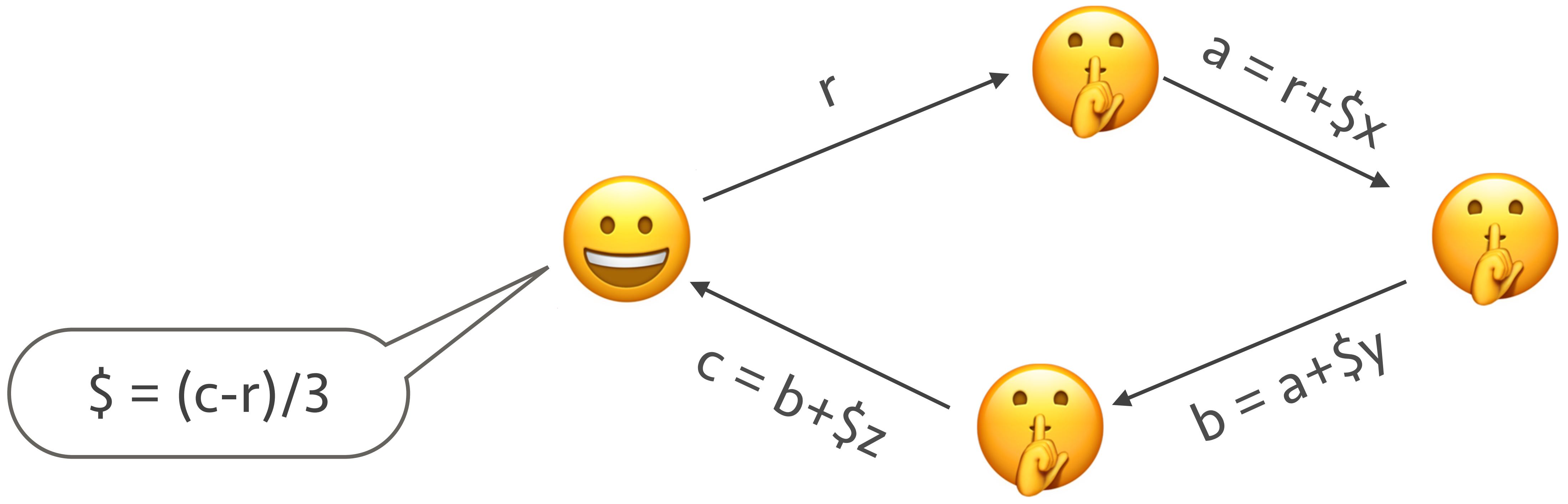
- Doing some statistics with private data

- E.g. finding the average person’s salary in the room?
  - The unacceptable way: Ask everyone for their income



A more socially acceptable approach

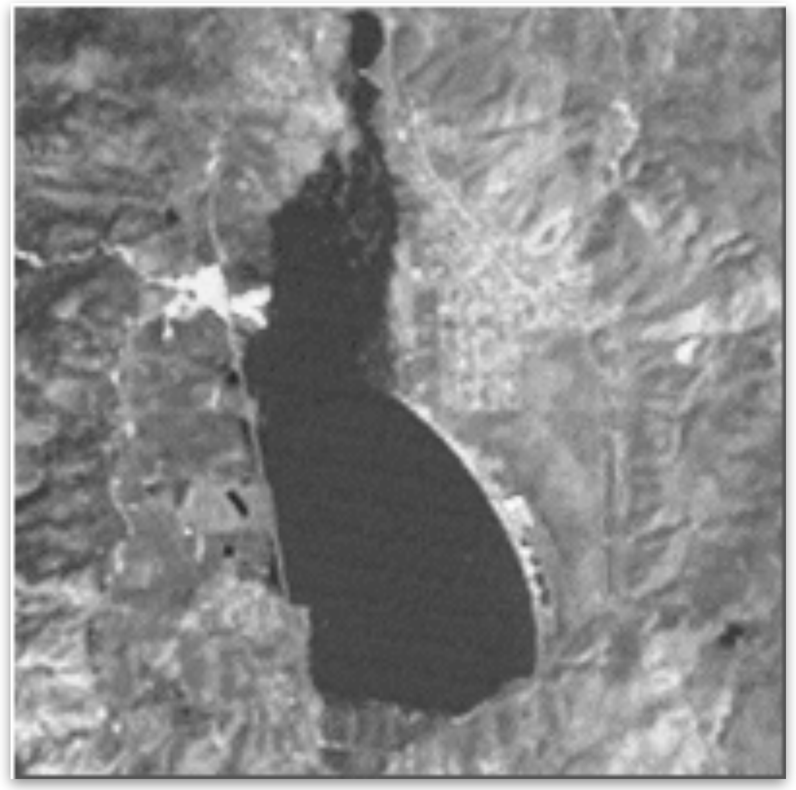
- Obfuscate dollar amounts with noise
  - Give a random number to next person, ask them to add their salary, pass the sum to the next person, and repeat
  - I get back sum of salaries plus my known random number
    - Remove random number and divide by number of people!
    - No private data has been shared!



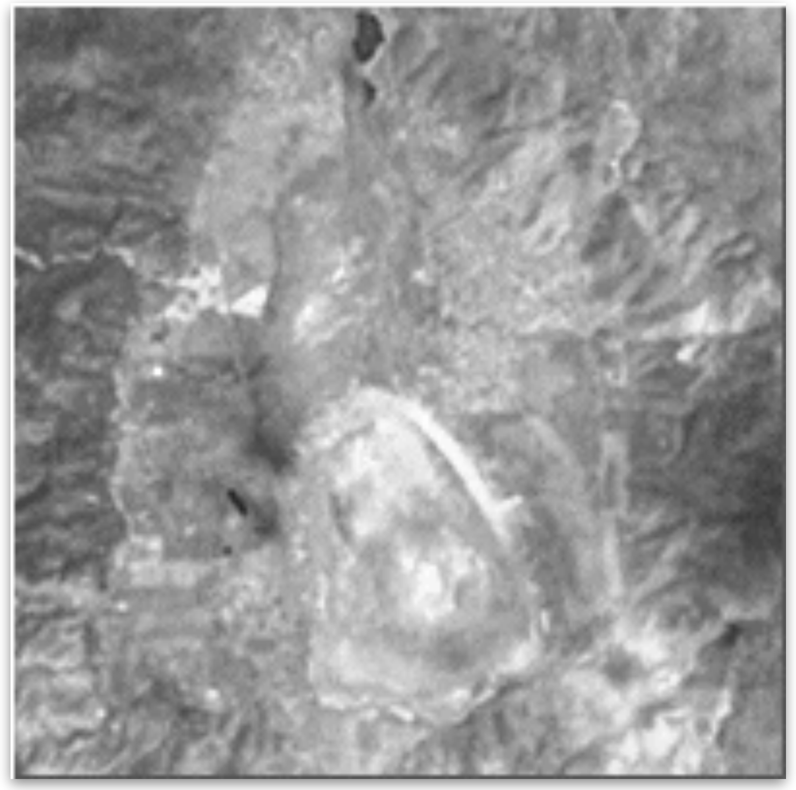
A signal-friendly example

- Your government wants you to examine the differences between satellite images over time
  - But the images are classified

Before



After

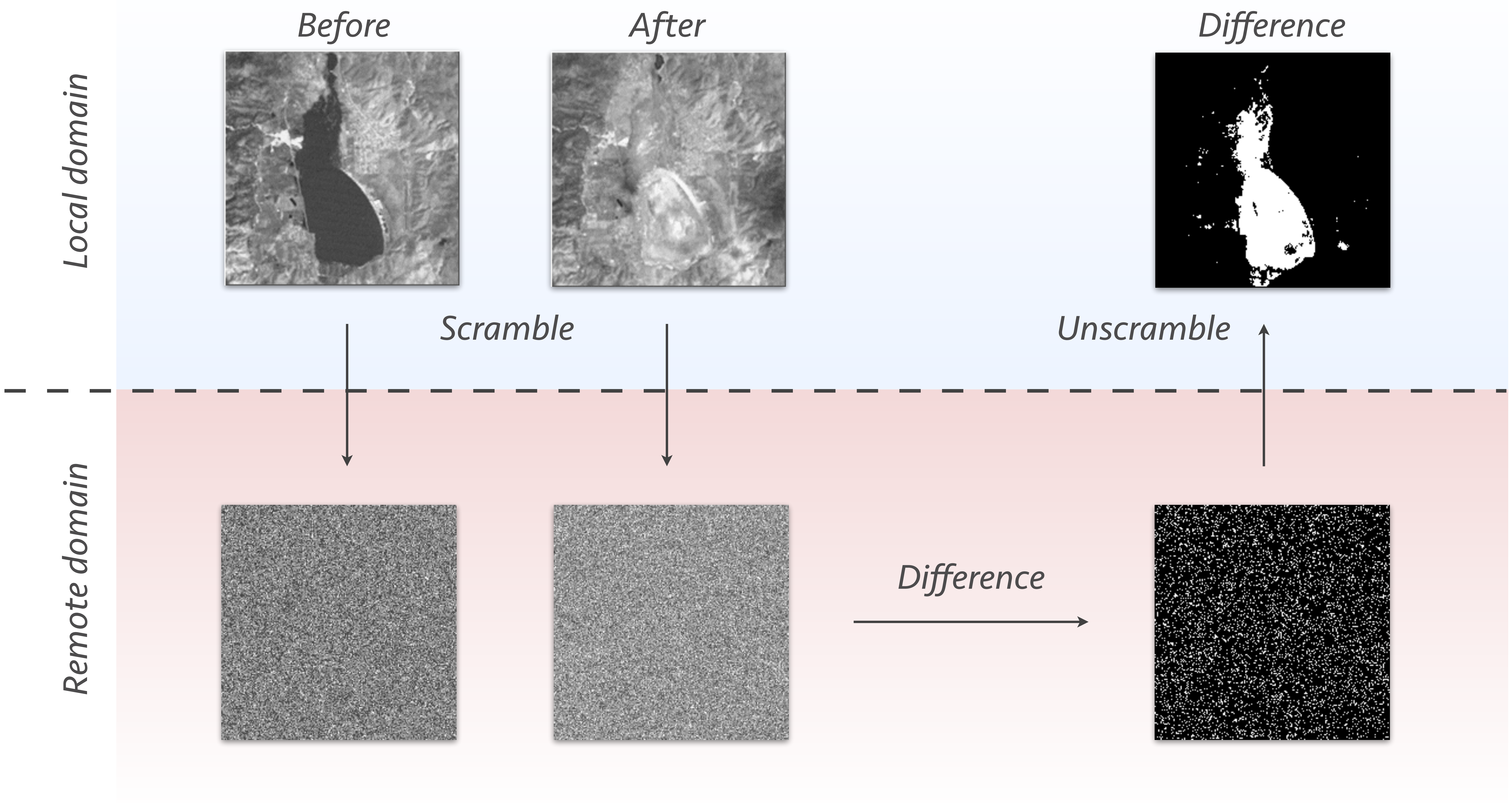


Difference

  - How do we do this one?

Using noise to obfuscate the data

- Permute the pixels before sending them for processing!



Basic processing primitives

- The vast majority of MLSP uses only a few operations
  - Dot product, greater than, maxarg, ...
- If we define secure versions of these, then we can construct more complex MLSP operations
  - While maintaining data privacy

The setup

- We will use two collaborators: Alice and Bob
  - They are not malicious, but curious
    - i.e. Won’t go out of their way to cheat, but are happy to inspect the data
  - There is no trusted 3rd party
    - i.e. they cannot trust a third person to mediate
- Both have private data that they do not want to share
  - But they need each other’s data to perform a computation
- They do not have infinite computing resources
  - Otherwise they could easily cheat

Multiple scenario cases

- Blind transaction
  - Alice wants to protect her data; Bob cannot see the data, but can see the process output
    - Even if Bob or a third party hacker is malicious they can't deduce the data
- Double-blind transaction
  - Alice wants to protect her data and the identity of the results; Bob cannot see the data nor the process output
    - Even if Bob or a third party hacker is malicious he can't deduce anything
- But there's more
  - Alice is malicious and wants to send specific data to reverse engineer Bob's algorithm
  - Alice has data which needs processing, and Bob's result can be sent over to Carl who analyses blindly to sent to Darren who then …

A simple exchange for now

- Alice has vector $\mathbf{x}$, Bob has vector $\mathbf{w}$ and threshold θ
  - They do not want to share their data with each other
  - They want to compute whether $\mathbf{w}^\top \mathbf{x} > \theta$
- Why this operation?
  - It is the core operation for most classifiers (elaborations later)

A cryptography diversion

- The Paillier cryptosystem
  - Provides an encoding: $y = \mathrm{E}[x]$ and it's inverse: $x = \mathrm{E}^{-1}[y]$
    - x is referred to as the *plaintext* and y as the *ciphertext*
- Has a hugely useful property
  - $\mathrm{E}[x]\mathrm{E}[y] = \mathrm{E}[x+y]$
  - This is known as *homomorphic encryption*
    - An operation on ciphertext corresponds to an operation in plaintext
    - Many other cryptosystems with similar properties

Some details

- Encryption: $c = \mathrm{E}[x] = g^x r^n \bmod n^2$, $x \in \mathbb{Z}_n, r \in \mathbb{Z}^*_n$
  - g is a random integer $\mathbb{Z}^*_{n^2}$, r is a random integer $\mathbb{Z}^*_n$
  - $n = p\, q$, where p, q are equal length primes
  - Public key: $\{n, g\}$
- Decryption: $x = \mathrm{E}[c] = \mathrm{L}(c^\lambda \bmod n^2) m \bmod n$
  - $\lambda = \mathrm{lcd}(p-1, q-1)$, $m = \mathrm{L}(g^\lambda \bmod n^2)-1 \bmod n$, $\mathrm{L}(x) = (x-1)/n$
  - Private key is: $\{\lambda, m\}$

Homomorphic properties

- Adding plaintexts via ciphertexts
  - $E^{-1}[\ E[x_1]\ E[x_2] \bmod n^2\ ] = x_1 + x_2 \bmod n$
  - $E^{-1}[\ E[x_1]\ g^{x_2} \bmod n^2\ ] = x_1 + x_2 \bmod n$
- Multiplying plaintexts via ciphertexts
  - $E^{-1}[\ E[x_1]^{x_2} \bmod n^2] = x_1 x_2 \bmod n$
  - $E^{-1}[\ E[x_2]^{x_1} \bmod n^2] = x_1 x_2 \bmod n$

Using Paillier for a secure inner product

- Desired operation:
  - Alice has vector $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$
  - Bob has vector $\mathbf{w} = \{w_1, w_2, \ldots, w_N\}$
  - Wanted outcome: Bob obtains $\mathrm{E}[\mathbf{w}^\top \mathbf{x}]$
- In the end, Bob should have result but cannot see it

The Secure Inner Product (SIP) protocol

- Alice:
  - Generates an $\mathrm{E}[]$ and $\mathrm{E}^{-1}[]$, sends $\mathrm{E}[]$ to Bob
  - Encrypts $\mathbf{x}$ and sends all $\mathrm{E}[x_i]$ to Bob
- Bob:
  - Computes homomorphic element-wise multiplication
    - $\mathrm{E}[x_i]^{w_i} = \mathrm{E}[x_i w_i]$
  - Bob computes homomorphic summation
    - $\prod \mathrm{E}[x_i w_i] = \mathrm{E}[\sum x_i w_i] = \mathrm{E}[\mathbf{w}^\top \mathbf{x}]$

But we need some more work

- A potential leak of Bob's data
  - Bob has $\mathrm{E}[\mathbf{w}^\top \mathbf{x}]$ and he can't see the result
    - That's a feature not a bug!
  - Alice can decrypt it, but then she will know $\mathbf{w}$
    - That would be a bug, not a feature!
- To protect Bob's data we need to "mask" the result

Masking

- In *masking/blinding* we obfuscate our data by adding random numbers (as we did with earlier examples)

- In our case we can perform *additive masking*
  - Bob can add a secret random number ρ to $\mathbf{w}^\top \mathbf{x}$
    - $\mathrm{E}[\mathbf{w}^\top \mathbf{x}]\mathrm{E}[\rho] = \mathrm{E}[\mathbf{w}^\top \mathbf{x} + \rho]$

Inching towards a classifier

- Use the SIP protocol:
  - Bob masks the obtained dot product: $\mathrm{E}[\mathbf{w}^\top \mathbf{x}]$
  - Bob sends the result $\mathrm{E}[\mathbf{w}^\top \mathbf{x} + \rho]$ to Alice
- Outcome:
  - Alice can obtain $\mathbf{w}^\top \mathbf{x} + \rho$, but has not clue what $\mathbf{w}$ is
  - Bob has no clue what $\mathbf{x}$ is
- To classify we need to compare $\mathbf{w}^\top \mathbf{x} + \rho$ with $\theta + \rho$

Performing a secure comparison

- Yao's millionaires' problem
  - Alice and Bob want to compare their assets, but not to reveal them
- More formally
  - Alice has x dollars
  - Bob has y dollars
  - Alice and Bob only need to find out if $x > y$
    - But Alice can't learn y and Bob can't learn x
- In our case instead of dollars we compare $\mathbf{w}^\top \mathbf{x} + \rho$ with $\theta + \rho$

Representing the values to compare

- Alice & Bob decide on a range of numbers to compare
  - e.g. \$5M, \$10M, …, \$50M, each value represented by $i = \{1, \dots, 10\}$
- Alice and Bob then have i_a and i_b dollars
  - They need to know if $i_a > i_b$
- A minor issue: This is a discrete and bound set
  - Bad for us since the dot product and threshold are real
    - But we can always quantize

Secure comparison algorithm

Alice's data is red
Bob's data is blue
Shared data is yellow

- Alice sends her public key to Bob
  - Bob can thus compute $\mathrm{E}[]$
- Bob computes $c = \mathrm{E}[x]$
  - x being a random integer of N bits
- Bob transmits to Alice $v = c + 1 - i_b$

  This ensures that Bob's number remains private
  - i_b is Bob's representation of assets

Secure comparison algorithm

Alice's data is red
Bob's data is blue
Shared data is yellow

- Alice generates a set of 10 numbers ← *Because we are considering 10 possible values*
  - $y_{i=1,\dots,10} = \mathrm{E}^{-1}[v + i - 1]$
  - The number corresponding to y_{i_b} will be equal to x
- Alice generates a random prime p ← *This ensures that we don't leak Alice's private key*
  - And computes $z_i = y_i \bmod p$
    - p is $N/2$-bits and must result in z_i's, that are apart by at least 2
- Alice sends p and $\mathbf{u} = \{z_1, \dots, z_{i_a-1}, z_{i_a}, 1+z_{i_a+1}, \dots, 1+z_{10}\}$

Secure comparison algorithm

Alice's data is red
Bob's data is blue
Shared data is yellow

- Bob computes $g = x \bmod p$
  - x was Bob's original random number, p is Alice's prime
- Bob compares g with the i_b'th element of $\mathbf{u}$
  - if $u_{i_b} = g$, then $i_a \geq i_b$
  - Otherwise $i_a < i_b$
- Neither party gets to share their original number!

Back to our original problem

- Alice has data $\mathbf{x}$, Bob has classifier $\{\mathbf{w}, \theta\}$
  - Step 1. Perform secure inner product
    - Bob obtains $\mathrm{E}[\mathbf{w}^\top \mathbf{x}]$
    - Bob sends to Alice $\mathrm{E}[\mathbf{w}^\top \mathbf{x}]\mathrm{E}[\rho] = \mathrm{E}[\mathbf{w}^\top \mathbf{x} + \rho]$
  - Step 2. Perform secure comparison
    - Alice compares $\mathbf{w}^\top \mathbf{x} + \rho$ with Bob's $\theta + \rho$, gets inequality result
- Alice keeps both data and classification outcome private
  - At the expense of extra overhead …

So what we do with this?

- Linear classifiers!
  - These are essentially the previous formula!
- What about more complex classifiers?
  - e.g. a Gaussian likelihood classifier
  - This isn't a linear operation

Making the Gaussian a dot product

- Gaussian log likelihood:

$$\log P(\mathbf{x};\mu,\Sigma) = -\frac{1}{2}\left(\mathbf{x}-\mu\right)^\top \Sigma^{-1}\left(\mathbf{x}-\mu\right) - \frac{d}{2}\log 2\pi - \frac{1}{2}\log\left|\Sigma_i\right|$$

- Equivalent to: $g(\mathbf{x}) = \mathbf{x}^\top \cdot \mathbf{W} \cdot \mathbf{x} + \mathbf{w}^\top \cdot \mathbf{x} + w$
  - Where: $\mathbf{W} = -\frac{1}{2}\Sigma^{-1},\;\; \mathbf{w} = \Sigma^{-1}\cdot\mu,\;\; w = -\frac{1}{2}\mu^\top\cdot\Sigma^{-1}\cdot\mu - \frac{1}{2}\log\left|\Sigma\right| - \frac{1}{2}\log 2\pi$
- Which can be simplified to: $g(\mathbf{x}) = \tilde{\mathbf{x}}^\top \cdot \tilde{\mathbf{W}} \cdot \tilde{\mathbf{x}}$
  - Concatenate 1 to $\mathbf{x}$ and include $\mathbf{w}$, w into $\mathbf{W}$

One more step

- But that's not a single product (we can do better)
- Instead we can do: $g(\mathbf{x}) = \tilde{\mathbf{W}} \cdot \tilde{\mathbf{x}}$
  - with:

$$\tilde{\mathbf{x}} = [1, x_1, x_2, \cdots, x_1 x_1, x_1 x_2, x_1 x_3, \cdots x_1 x_N, x_2 x_1, x_2 x_2, x_2 x_3, \cdots, x_N x_N]$$

$$\tilde{\mathbf{W}} = \begin{bmatrix} w, & \Sigma^{-1} \cdot \mu, & -\frac{1}{2} \operatorname{vec} \Sigma^{-1} \end{bmatrix}, \quad w = -\frac{1}{2} \mu^\top \Sigma^{-1} \mu - \frac{1}{2} \log |\Sigma| - \frac{1}{2} \log 2\pi$$

- Which only needs a simple secure product operation

Stringing protocols together

- It is often easier to modularize a process
  - Coming up with a secure HMM is hard
  - Coming up with a secure Gaussian, followed by a secure sum, followed by a secure transition regularizer, followed by …, is easier
- To do so we can use *additive shares*
  - At each step the output is additively distributed between parties
  - E.g. for SIP: $z+v=\mathrm{SIP}(x, y)$, Alice has x and z, Bob has y an v
  - We can keep processing keeping all intermediate results hidden

But there are some issues ...

- Data needs to be integer-valued
  - Not a huge problem, we can quantize data to desired accuracy
  - Can be a problem with, e.g. secure comparison though
- Encryption/decryption is computationally intensive
  - There is work on specialized hardware for this
- Are these worthwhile ideas?
  - Maybe later, email encryption was just as hopeless a while back

More reading material

- Yao's Millionaires' problem:
  - http://research.cs.wisc.edu/areas/sec/Yao1982.pdf
- Secure Multiparty Computations and Data Mining
  - https://eprint.iacr.org/2008/197.pdf
- Full-blown secure HMM implementation for speech:
  - http://paris.cs.illinois.edu/pubs/smaragdis-tasl07-3.pdf