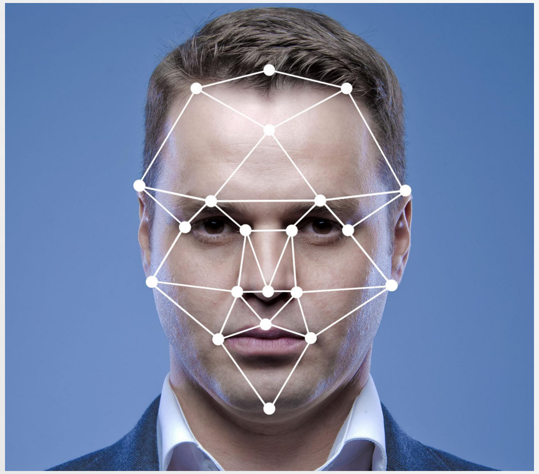
ABY^3: A MIXED PROTOCOL FRAMEWORK FOR MACHINE LEARNING

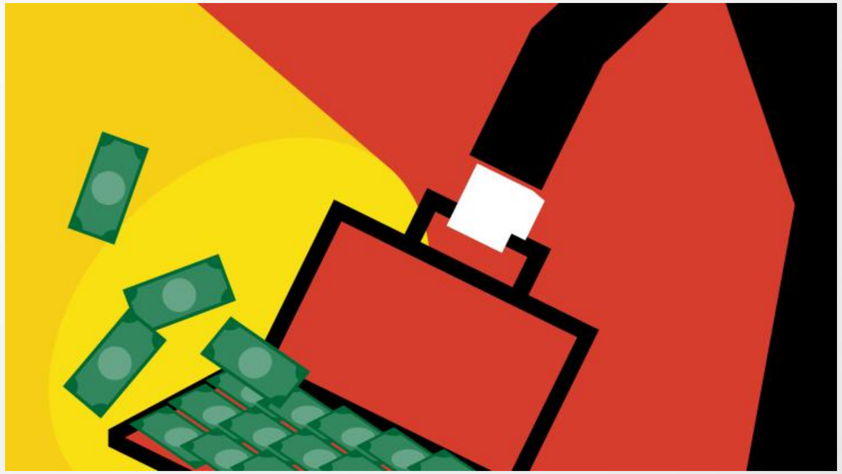
Peter Rindal
Payman Mohassel

VISA Research

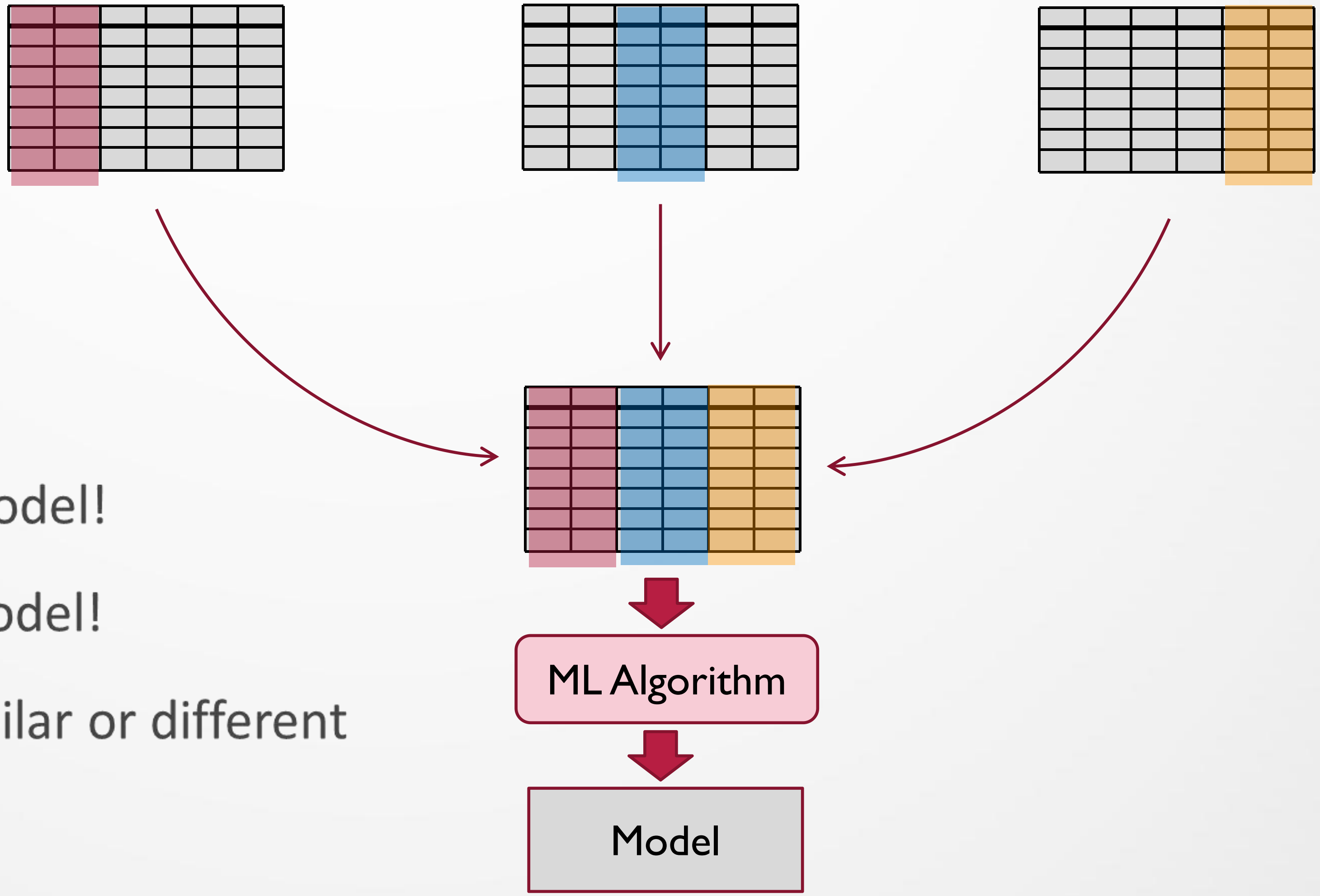
Machine Learning and Privacy

- **Cognitive tasks:** voice, facial recognition
- **Medical:** genetic testing, disease prediction
- **Financial:** fraud detection, credit rating
- Inference:
  - User does not want to share their data
  - Model owner does not want to share model
- Training:
  - Requires large data sets, often from different sources



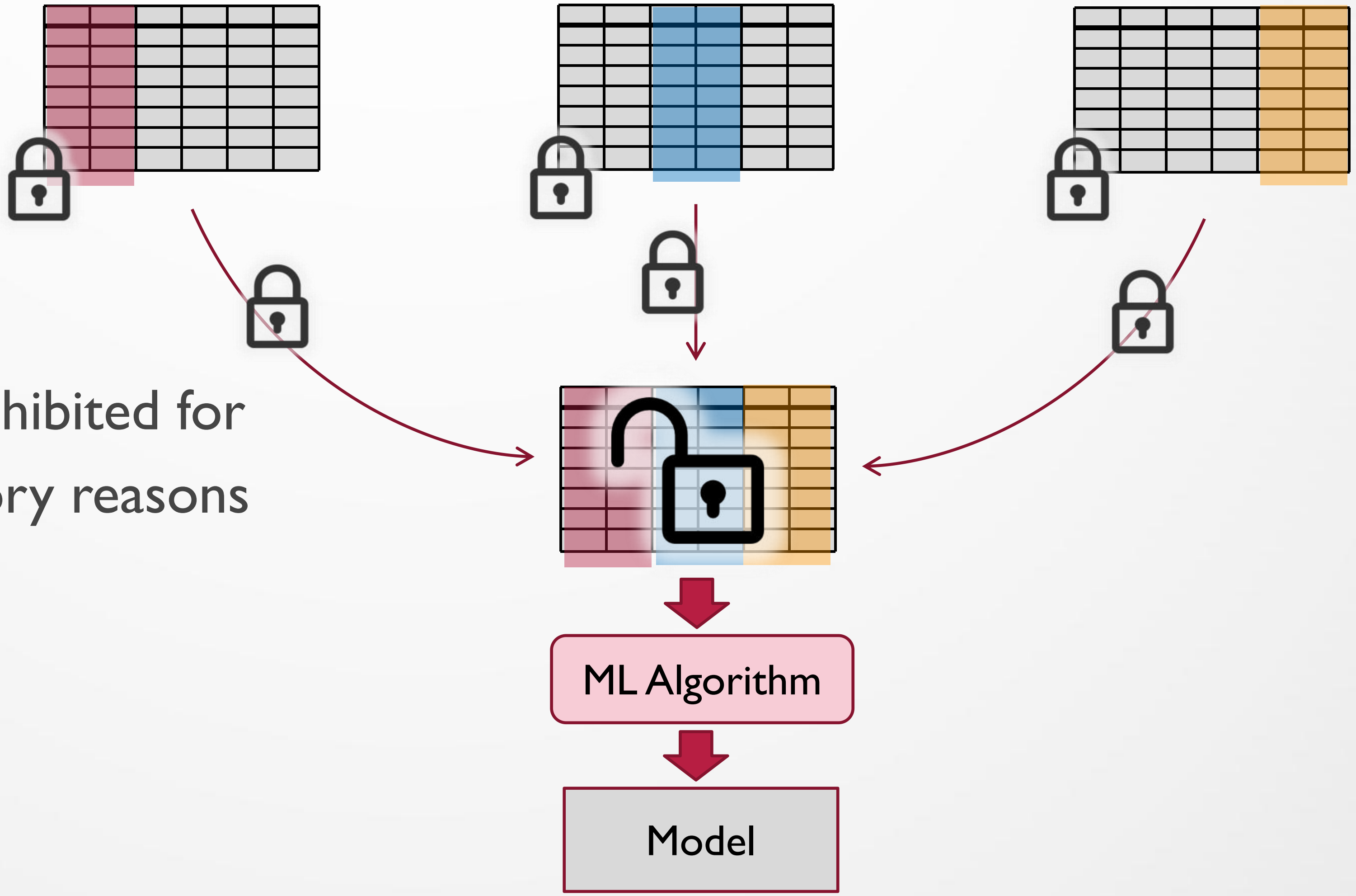


Training A better Model



- More data → better model!
- Richer data → better model!
- Each party can have similar or different types of data

Security Concerns



- Data sharing often prohibited for completive or regulatory reasons
- Use encryption?
  - Encrypt at rest
  - Encrypt in transit
  - **Decrypted in use!!**

Our Results

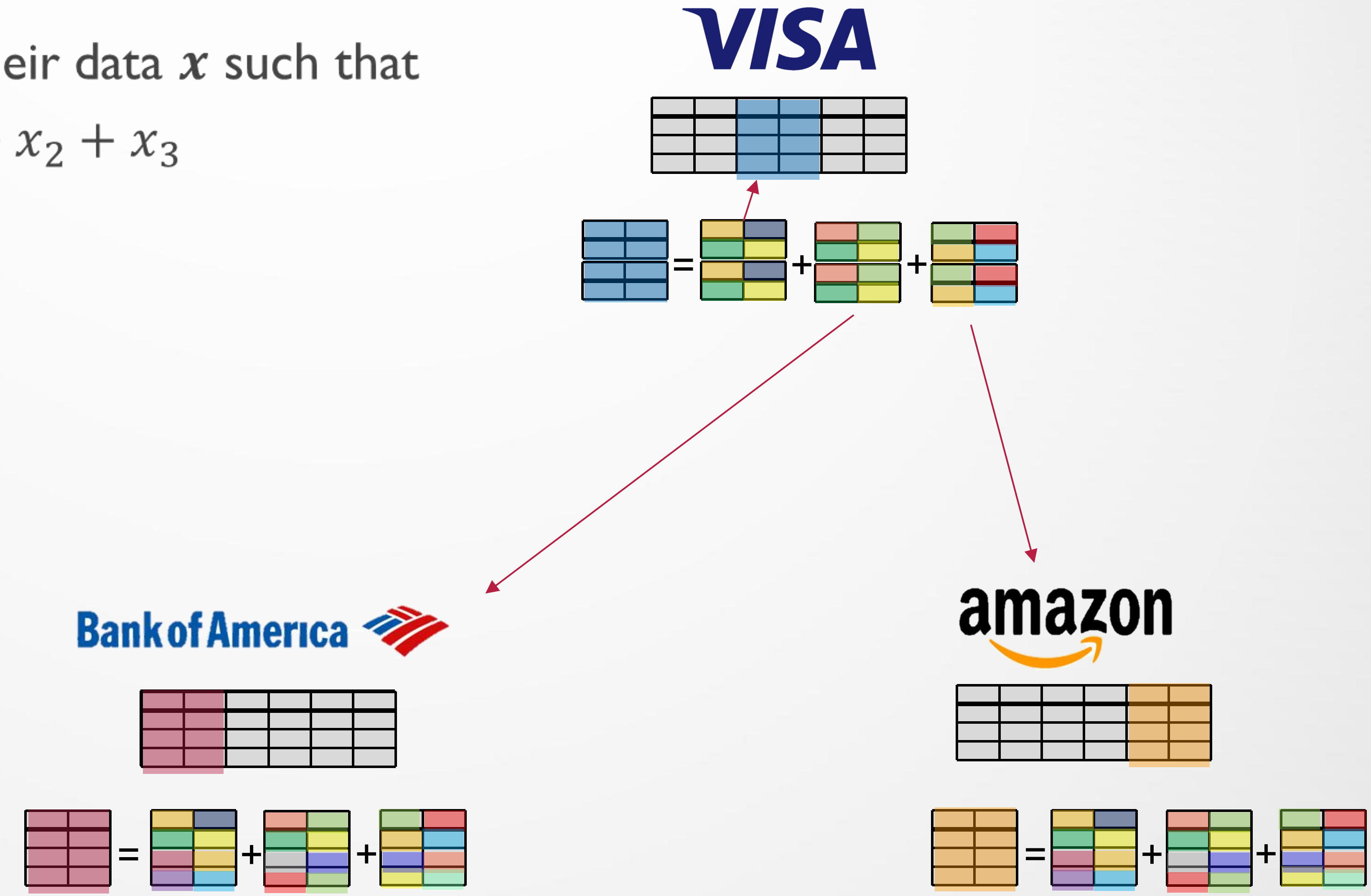
- New three party MPC Protocols:
  - Efficient support for fixed-point arithmetics
  - Improved matrix multiplication
  - Efficient piece-wise polynomial evaluation
  - Conversions between **A**rithmetic, **B**oolean, and **Y**ao secret shares
- Always encrypted machine learning **training and inference**:
  - Linear Regression
  - Logistic Regression
  - Neural Networks
  - Extendable to other models

Protocols and Building block

Always Encrypted

- Each party "Encrypts" their data x such that

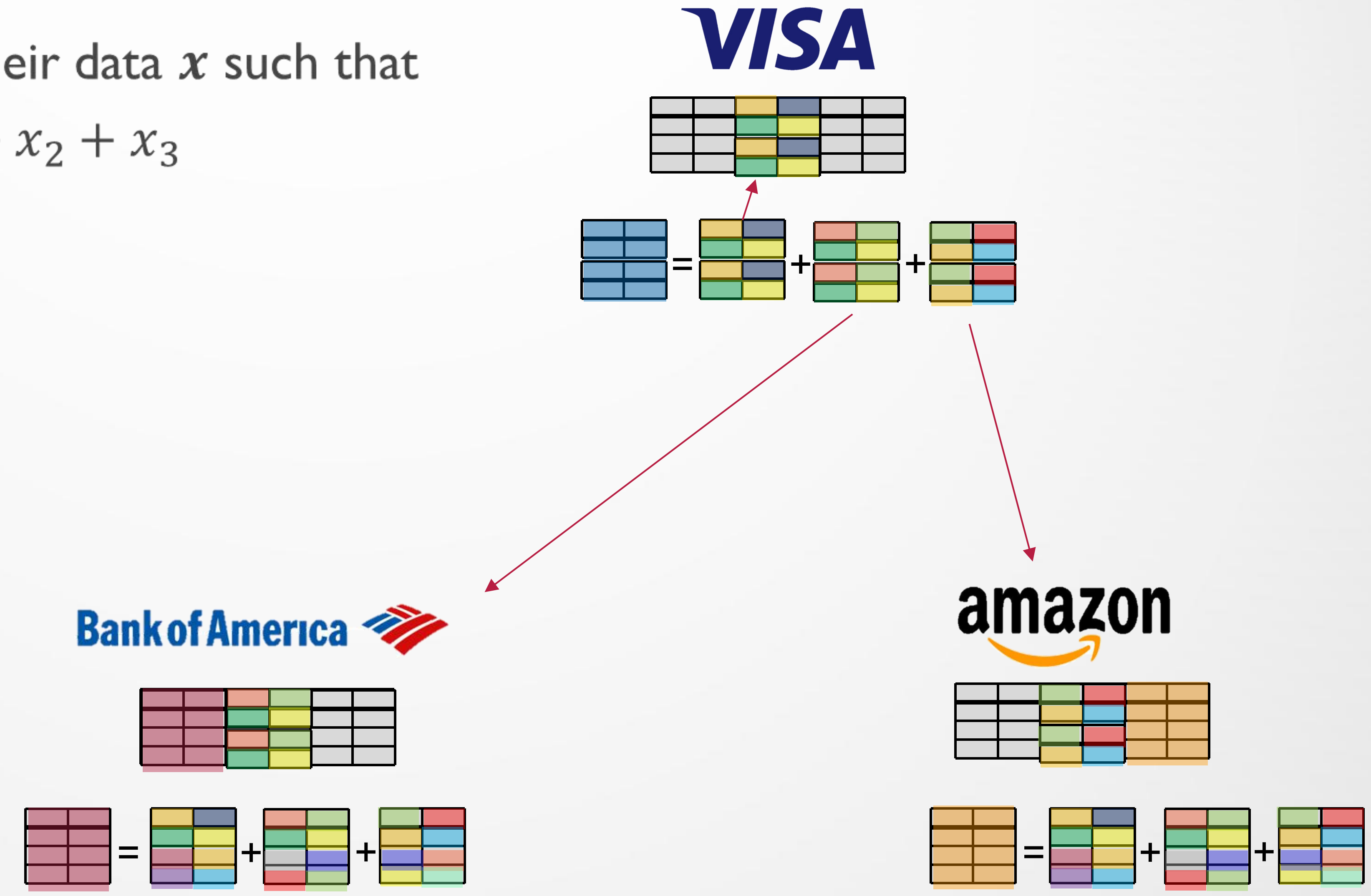
$$x = x_1 + x_2 + x_3$$



Always Encrypted

- Each party "Encrypts" their data x such that

$$x = x_1 + x_2 + x_3$$

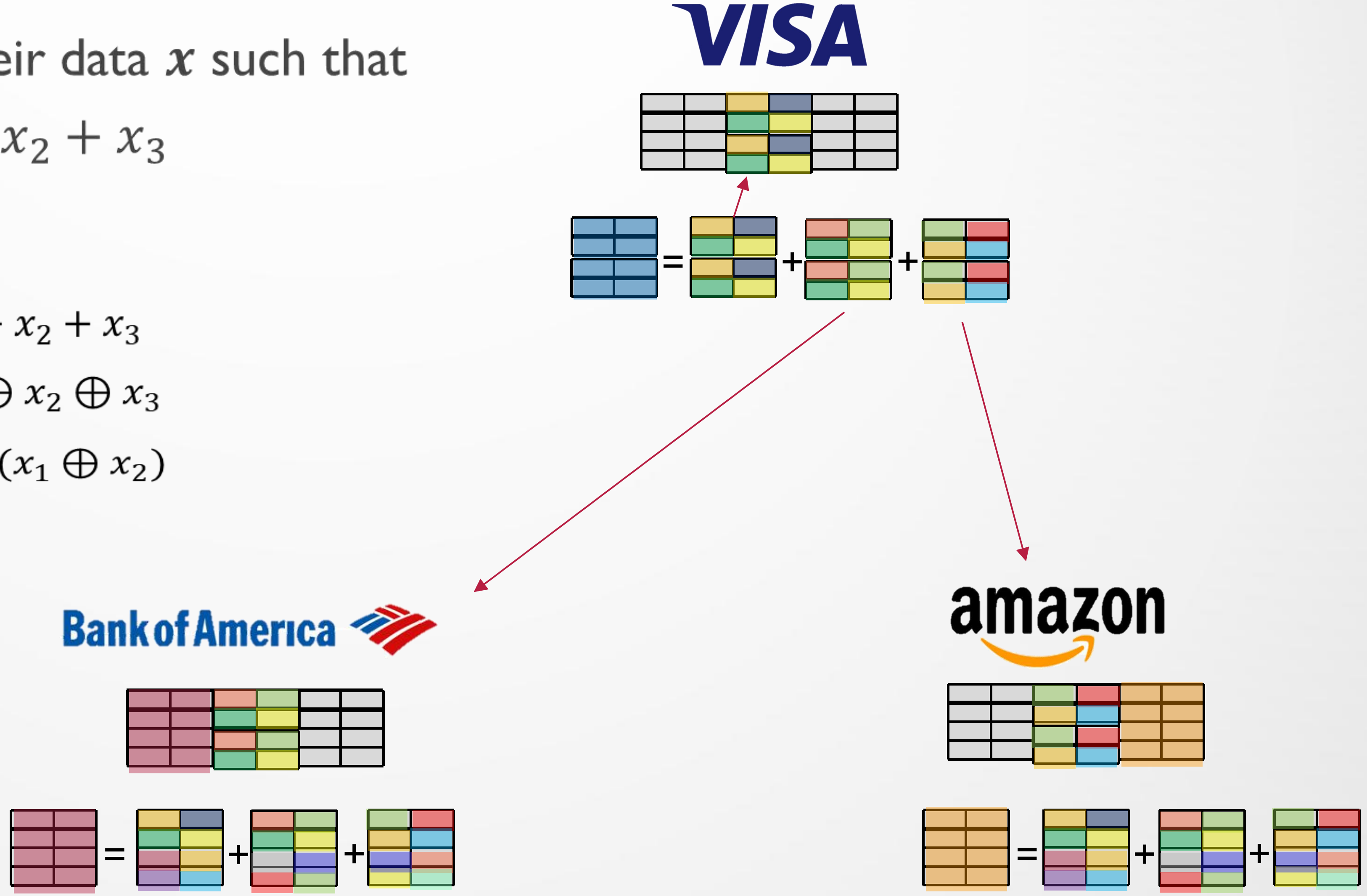


Always Encrypted

- Each party "Encrypts" their data x such that

$$x = x_1 + x_2 + x_3$$

- Share Types:
  - **A**rithmetic: $[\![x]\!]^{\mathrm{A}} \Rightarrow \quad x_1 + x_2 + x_3$
  - **B**oolean: $[\![x]\!]^{\mathrm{B}} \Rightarrow \quad x_1 \oplus x_2 \oplus x_3$
  - **Y**ao G.C.: $[\![x]\!]^{\mathrm{Y}} \Rightarrow \quad LSB(x_1 \oplus x_2)$

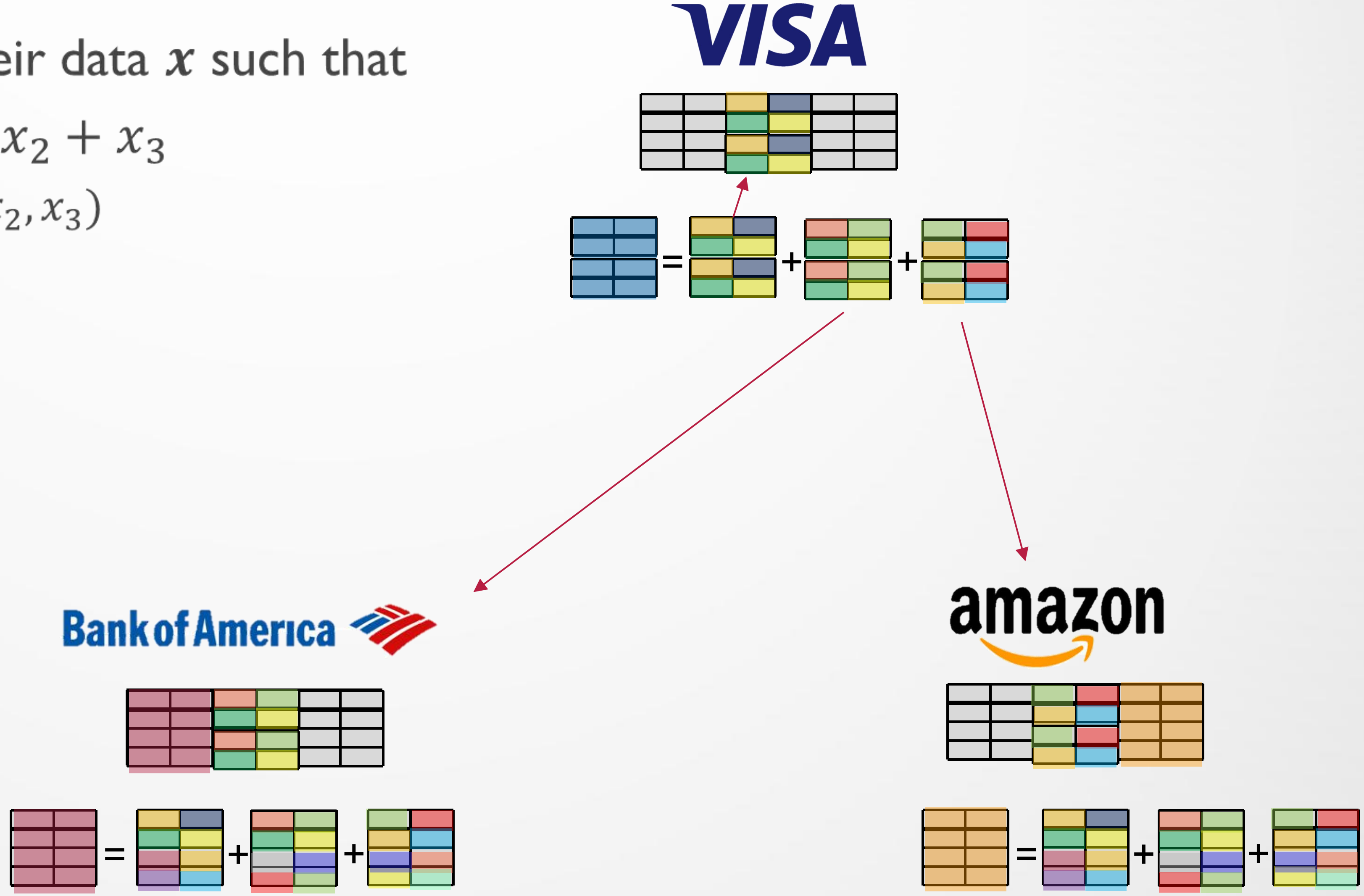


Always Encrypted

- Each party "Encrypts" their data x such that

$$x = x_1 + x_2 + x_3$$

$$[\![x]\!] := (x_1, x_2, x_3)$$

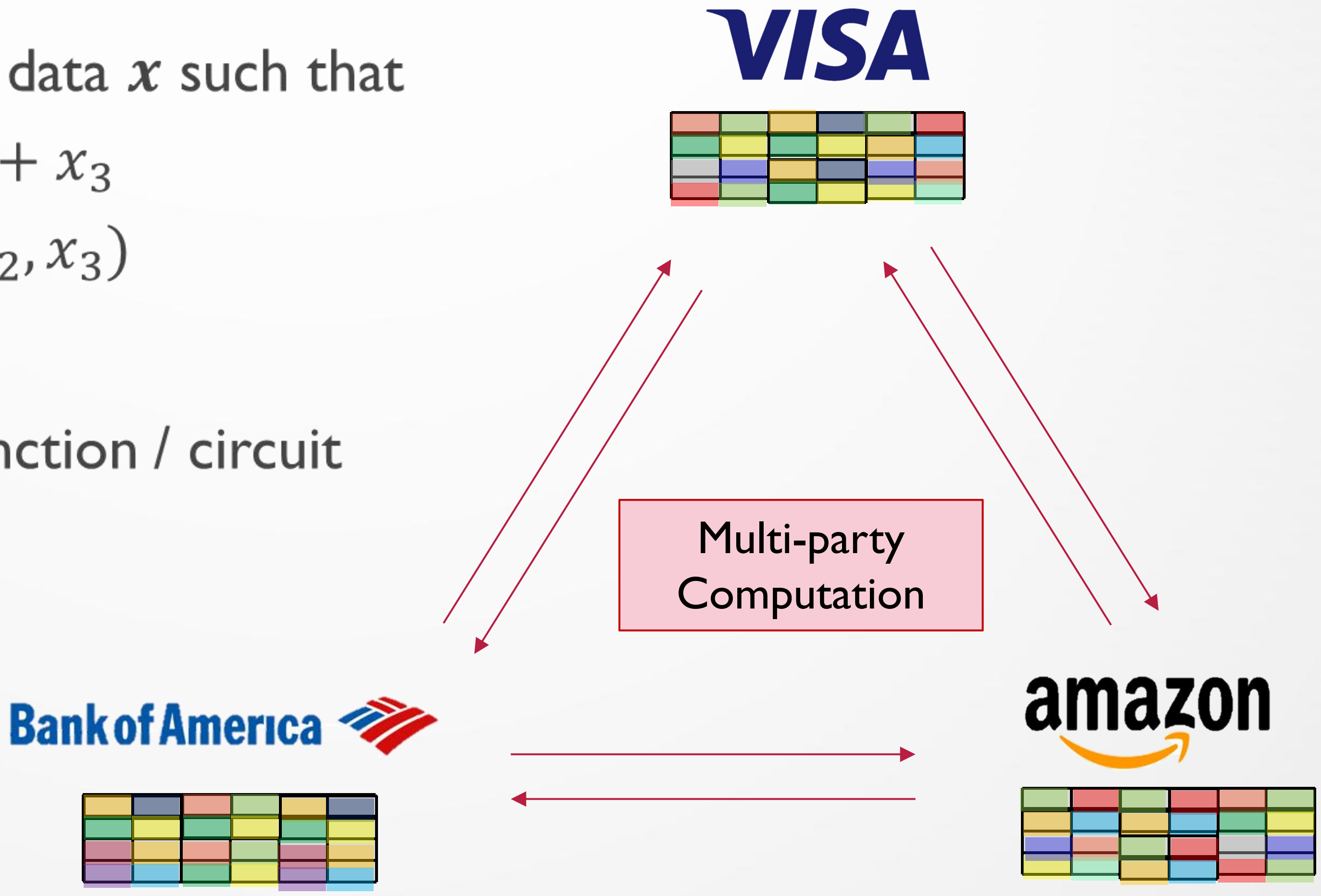


Always Encrypted

- Each party "Encrypts" their data x such that

$$x = x_1 + x_2 + x_3$$
$$[\![x]\!] := (x_1, x_2, x_3)$$

- Possible to compute any function / circuit in this format



How to Compute on Shared Data

- Say we have shared $[\![x]\!], [\![y]\!]$

- Addition $[\![z]\!] = [\![x]\!] + [\![y]\!]$:

$$z_i = x_i + y_i$$

VISA

x_1, y_1

x_2, y_2

x_3, y_3

Bank of America

amazon

How to Compute on Shared Data

- Multiplication $[\![z]\!] = [\![x]\!] * [\![y]\!]$:
  - $z_i \neq x_i * y_i$
  - Use "replicated secret sharing"

VISA

x_1, y_1

x_2, y_2

Bank of America

x_3, y_3

amazon

How to Compute on Shared Data

- Multiplication $[\![z]\!] = [\![x]\!] * [\![y]\!]$:
  - $z_i \neq x_i * y_i$
  - Use "replicated secret sharing"
  - Observe:

$$xy = (x_1 + x_2 + x_3)(y_1 + y_2 + y_3)$$
$$= x_1y_1 + x_1y_2 + x_1y_3$$
$$+ x_2y_1 + x_2y_2 + x_2y_3$$
$$+ x_3y_1 + x_3y_2 + x_3y_3$$

  - Let

$$z_1 = x_1y_1 + x_1y_3 + x_3y_1$$
$$z_2 = x_1y_2 + x_2y_2 + x_2y_2$$
$$z_3 = x_2y_3 + x_3y_2 + x_3y_3$$

VISA: x_1, y_1; x_3, y_3

Bank of America: x_1, y_1; x_2, y_2

amazon: x_2, y_2; x_3, y_3

How to Compute on Shared Data

- Multiplication $[\![z]\!] = [\![x]\!] * [\![y]\!]$:
  - $z_i \neq x_i * y_i$
  - Use "replicated secret sharing"
  - Observe:

$$xy = (x_1 + x_2 + x_3)(y_1 + y_2 + y_3)$$
$$= x_1y_1 + x_1y_2 + x_1y_3$$
$$+ x_2y_1 + x_2y_2 + x_2y_3$$
$$+ x_3y_1 + x_3y_2 + x_3y_3$$

  - Let

$$z_1 = x_1y_1 + x_1y_3 + x_3y_1$$
$$z_2 = x_1y_2 + x_2y_2 + x_2y_2$$
$$z_3 = x_2y_3 + x_3y_2 + x_3y_3$$

VISA

x_1, y_1, z_1
$x_3, y_3,$

Bank of America

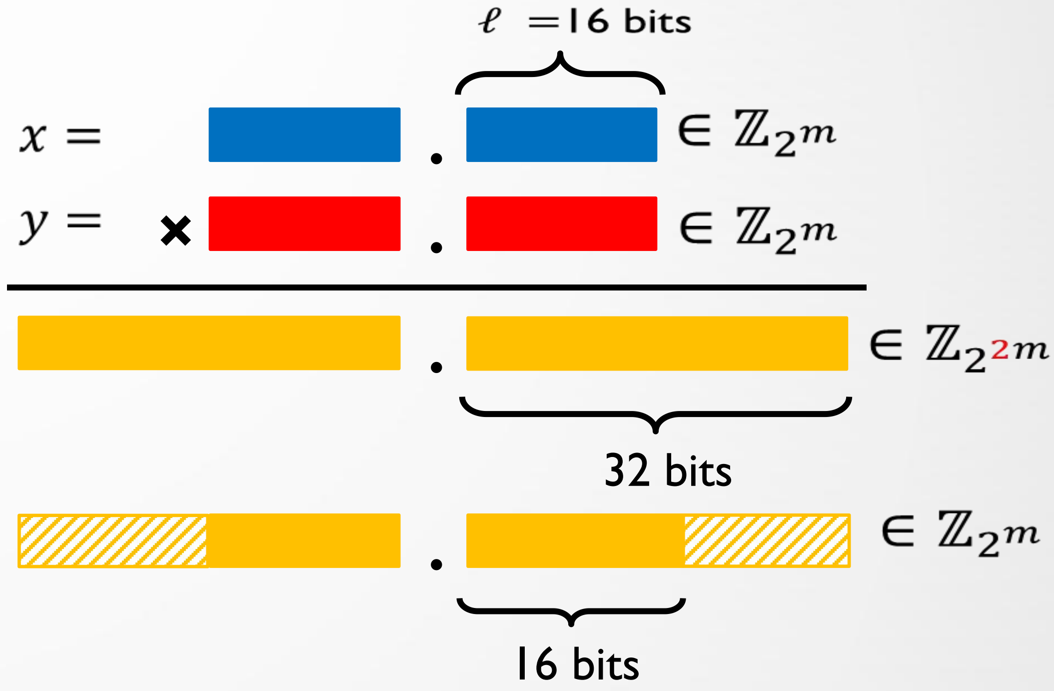
$x_1, y_1,$
x_2, y_2, z_2

amazon

$x_2, y_2,$
x_3, y_3, z_3

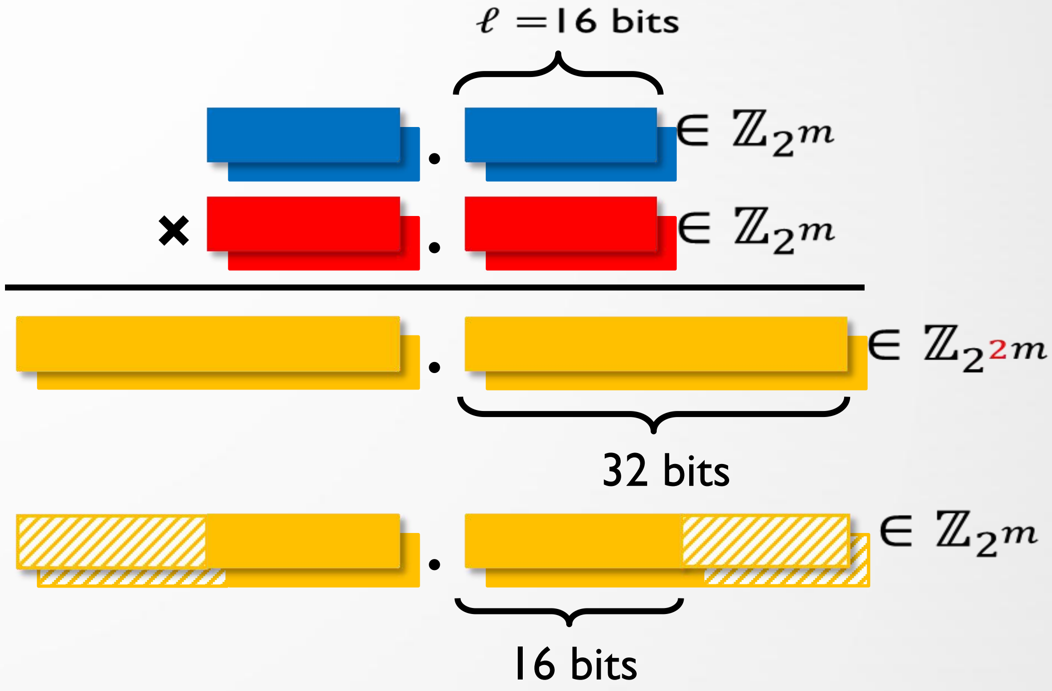
Decimal Multiplications in Integer Group

- Treat bottom bits as factional bits
- Multiplication drops top/bottom d bits



Decimal Multiplications in Integer Group

- Treat bottom bits as factional bits
- Multiplication drops top/bottom d bits
- Mohassel Zhang '17
  - Introduced secret shared version
  - Adds small rounding errors
  - Only works on 2-out-of-2 secret sharing
- We extend to **any honest majority**
  - Briefly switch to 2-out-of-2 sharing

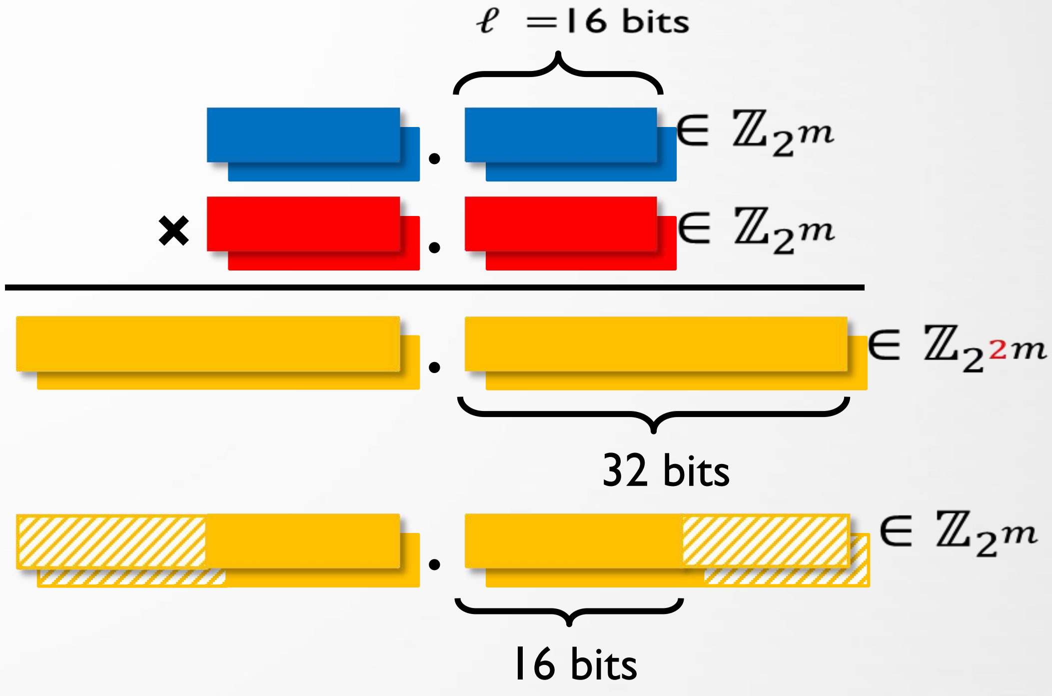


Preprocess: $[\![r]\!] \leftarrow \mathbb{Z}_{2^m}$, $\quad [\![r']\!] = [\![r]\!]/2^\ell$

1. $[\![z']\!] = [\![x]\!] * [\![y]\!]$
2. $t = \text{Reveal}([\![z']\!] - [\![r]\!])$
3. $[\![z]\!] = t/2^\ell + [\![r']\!]$

Decimal Multiplications in Integer Group

- Treat bottom bits as factional bits
- Multiplication drops top/bottom d bits
- Mohassel Zhang '17
  - Introduced secret shared version
  - Adds small rounding errors
  - Only works on 2-out-of-2 secret sharing
- We extend to **any honest majority**
  - Briefly switch to 2-out-of-2 sharing
  - Round preserving
  - Malicious secure



Preprocess: $[\![r]\!] \leftarrow \mathbb{Z}_{2^m}, \qquad [\![r']\!] = [\![r]\!]/2^{\ell}$

1. $[\![z']\!] = [\![x]\!] * [\![y]\!]$
2. $t = \text{Reveal}([\![z']\!] - [\![r]\!])$
3. $[\![z]\!] = t/2^{\ell} + [\![r']\!]$

Share #1 Share #2

Matrix Multiplication on Shared Data

- Given, $X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \end{bmatrix}$
- Compute $Z = X \times Y$
  - Classic "rows × columns" algorithm. E.g. $z_i = \sum_i x_{ij} y_j$
- Each multiplication requires communication
  - Communication = nd numbers!
- Recall "half" a multiplication is done locally.
  - Do half multiplies for $z_i = \sum_i x_{ij} y_j$
  - Only send final result z_i
  - **d times less communication**

$$z_1 = x_1 y_1 + x_1 y_3 + x_3 y_1$$
$$z_2 = x_1 y_2 + x_2 y_2 + x_2 y_2$$
$$z_3 = x_2 y_3 + x_3 y_2 + x_3 y_3$$

VISA

x_1, y_1, z_1
$x_3, y_3,$

$x_1, y_1,$
x_2, y_2, z_2

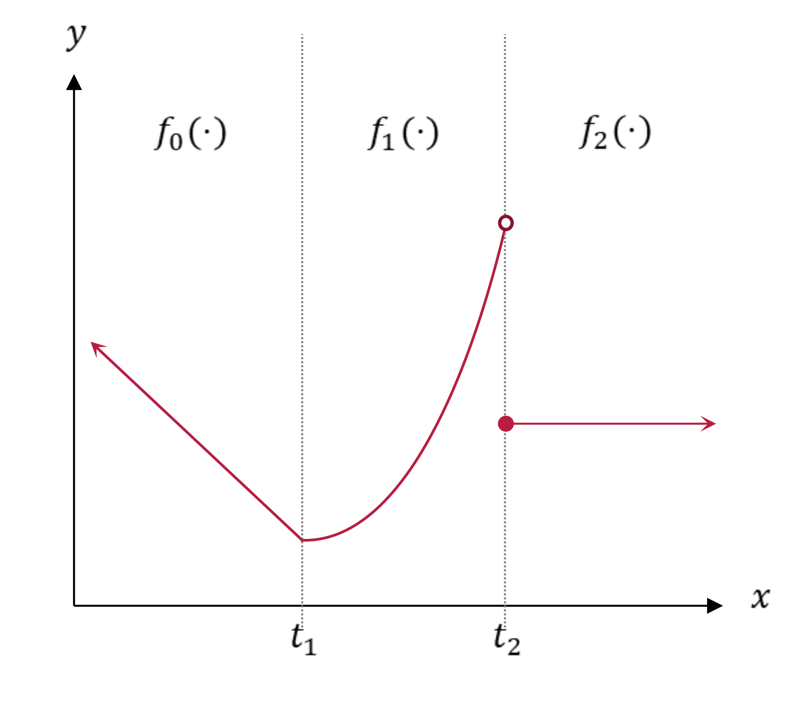
$x_2, y_2,$
x_3, y_3, z_3

Bank of America

amazon

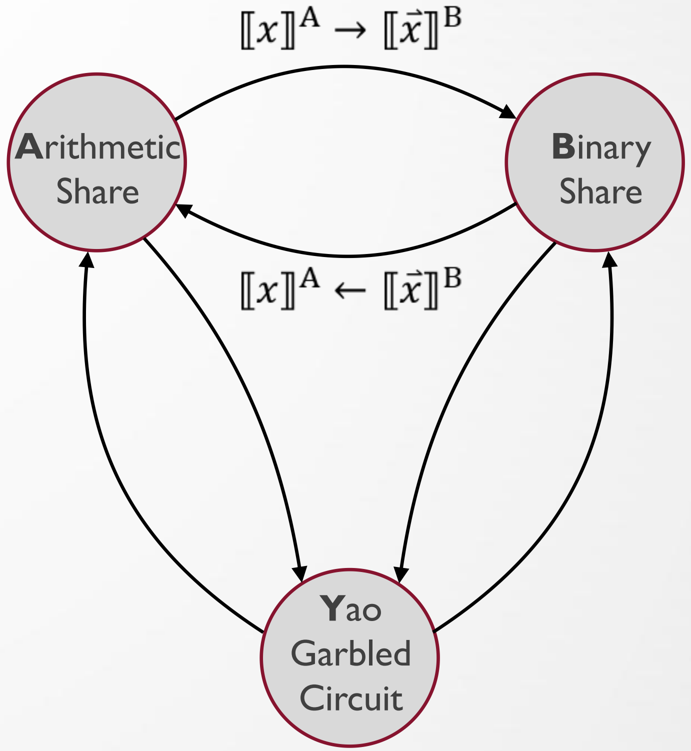
Piece-wise Polynomial

- Let
  - $t_0 = -\infty$
  - $t_1 =$ first threshold
  - $\dots$
  - $t_m = \infty$
- Strategy: $$f(\llbracket x \rrbracket) = \sum_{i=0}^{m-1} \overbrace{\left(\llbracket x \rrbracket > t_i \bigwedge \llbracket x \rrbracket \le t_{i+1} \right)}^{\text{range test}} * f_i(\llbracket x \rrbracket)$$
  - Polynomial $f_i(\cdot)$ easy to compute
  - How to compute $c_i = \llbracket x \rrbracket > t_i \wedge \llbracket x \rrbracket \le t_{i+1}$?
    - $\llbracket t_i - x \rrbracket^{\mathrm{B}}$ = *Convert-to-binary*$(\llbracket t_i - x \rrbracket)$
    - $\llbracket b_i \rrbracket^{\mathrm{B}}$ = *Most-significant-bit*$\left(\llbracket t_i - x \rrbracket^{\mathrm{B}}\right)$ // $b_i = (x > t_i)$
    - $\llbracket c_i \rrbracket$ = *Converty-to-arithmetic*$(\llbracket b_i \rrbracket \wedge \neg \llbracket b_{i+1} \rrbracket)$
- Total cost: $O(\log m)$ rounds and $O(m)$ bits of communication



Conversion

- Need to convert secret sharing of $x \in \mathbb{Z}_{2^k}$:
  - Arithmetic to binary: $[\![x]\!]^{\mathrm{A}} \rightarrow [\![\vec{x}]\!]^{\mathrm{B}} = [\![x_1]\!]^{\mathrm{B}}, \dots, [\![x_k]\!]^{\mathrm{B}}$
  - Binary to Arithmetic: $[\![\vec{x}]\!]^{\mathrm{B}} \rightarrow [\![x]\!]^{\mathrm{A}}$
- Strategy:
  - $[\![x]\!]^{\mathrm{A}} = (x_1, x_2), (x_2, x_3), (x_3, x_1)$
  - Parties provide x_1, x_2, x_3 as input to the binary MPC protocol
  - Evaluate the circuit: $[\![\vec{x}]\!]^{\mathrm{B}} = [\![\vec{x}_1]\!]^{\mathrm{B}} + [\![\vec{x}_2]\!]^{\mathrm{B}} + [\![\vec{x}_3]\!]^{\mathrm{B}}$
- Optimizations:
  - Implement $+$ using low depth circuit, $\mathit{depth} = \log k$
  - Have party 1 locally compute and input $x_1 + x_2$
- Also implement conversion to/from garbled circuits
  - Garbled circuits allow low latency for high depth circuit



Malicious Security

- We extend our semi-honest protocol to the malicious setting
- If any **one** party tries to cheat, the protocol can detect it.
  - Leverage replicated secret sharing
- Most operations roughly 4 times more expensive

x_1, y_1, z_1
x_3, y_3, z_3

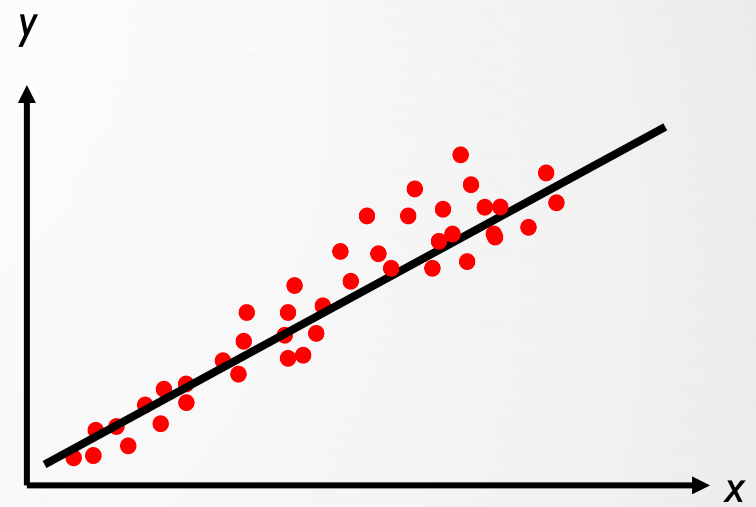
x_1, y_1, z_1
x_2, y_2, z_2

x_2, y_2, z_2
x_3, y_3, z_3

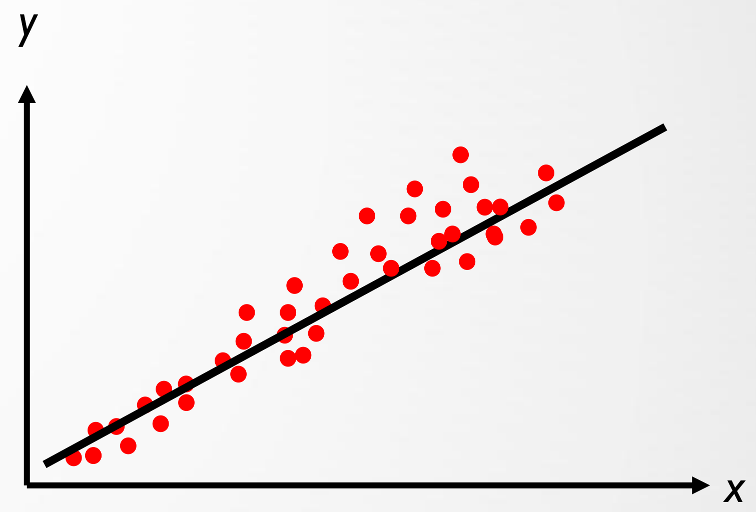
Application: Machine Learning

Linear Regression on Shared Data

- Given, $\mathrm{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$
- Find a linear function f s.t. $f(X_i) \approx y_i$



Linear Regression on Shared Data



- Given, $X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ \boldsymbol{x_{21}} & \boldsymbol{x_{22}} & \cdots & \boldsymbol{x_{2d}} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix}, y = \begin{bmatrix} y_1 \\ \boldsymbol{y_2} \\ \vdots \\ y_n \end{bmatrix}$
- Find a linear function f s.t. $f(X_i) \approx y_i$
- Model: $f_{\mathrm{w}}(X_i) = \sum_{j=1}^{d} \boldsymbol{x_{ij} w_j}$
- Update function:

$$\boldsymbol{w_j = w_j - \alpha(X_i \mathrm{w} - y_i) x_{ij}}$$

- Cost function: $C(\mathrm{w}) = \frac{1}{2}(f_{\mathrm{w}}(X) - y)^2$

(a.k.a. L2 norm)

Linear Regression on Shared Data

- Given, $\mathrm{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$
- Find a linear function f s.t. $f(X_i) \approx y_i$
- Model: f_{w} … x_{ij} …
- Update function:

$$w_j = w_j - \alpha(X_i\mathrm{w} - y_i)x_{ij}$$

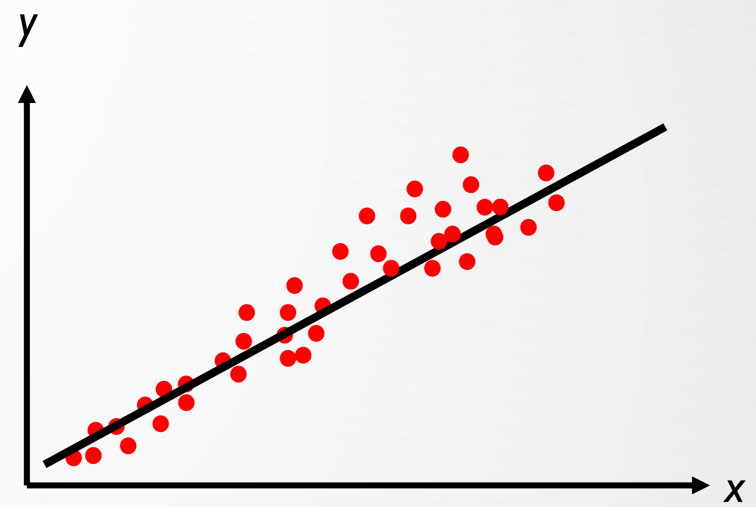
Learning Rate

Error Magnitude

Error Direction

…nction: $C(\mathrm{w}) = \frac{1}{2}(f_{\mathrm{w}}(X) - y)^2$

(a.k.a. L2 norm)



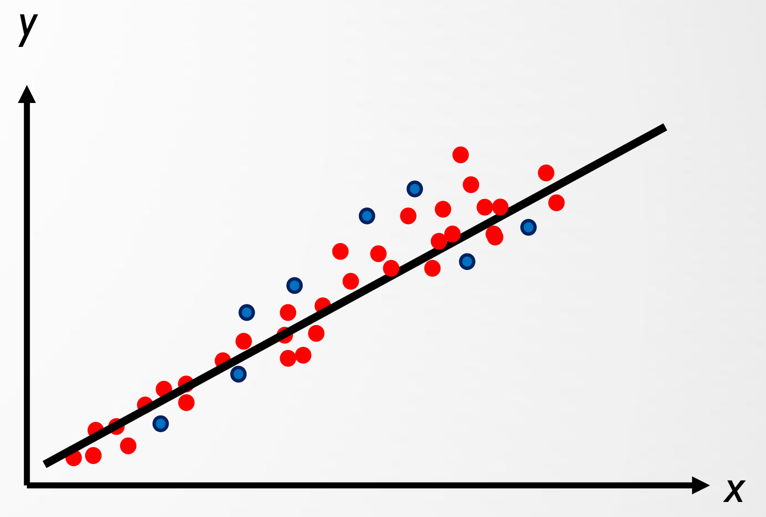
Batch Linear Regression on Shared Data

Current Batch

- Given, $X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$
- Find a linear function f s.t. $f(X_i) \approx y_i$
- Update in batches:
    - For $B \subset \{1,2,\ldots,n\}$

$$\mathbf{w} = \mathbf{w} - \frac{\mathbf{1}}{|\boldsymbol{B}|}\alpha \boldsymbol{X}_{\boldsymbol{B}}^{\boldsymbol{T}} \times (\boldsymbol{X}_{\boldsymbol{B}} \times \mathbf{w} - \boldsymbol{Y}_{\boldsymbol{B}})$$

- Must do many matrix multiplications



Logistic Regression

- Instead of computing

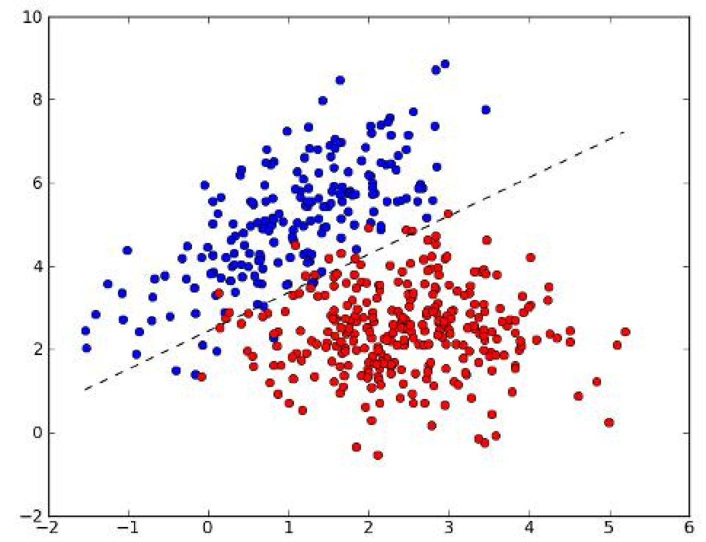
$$\mathbf{w} = \mathbf{w} - \frac{\mathbf{1}}{|\boldsymbol{B}|}\boldsymbol{\alpha X_B^T} \times (\boldsymbol{X_B} \times \mathbf{w} - \boldsymbol{Y_B})$$

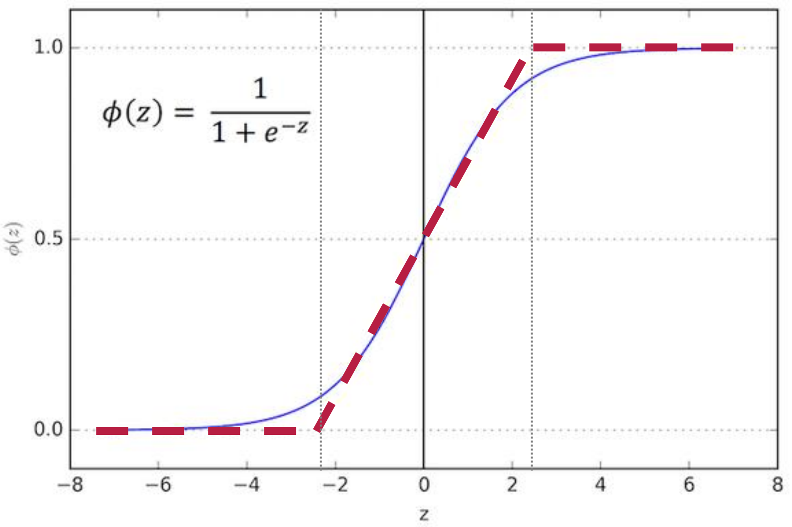
- We compute

$$\mathbf{w} = \mathbf{w} - \frac{\mathbf{1}}{|\boldsymbol{B}|}\boldsymbol{\alpha X_B^T} \times (\boldsymbol{\phi}(\boldsymbol{X_B} \times \mathbf{w}) - \boldsymbol{Y_B})$$

- ϕ is very expensive to compute with just $+, *$ operations
- We build a piece-wise polynomial protocol for $\phi'(\cdot) \approx \phi(\cdot)$

$$\phi'(z) = \max\left(0, \min\left(z + \frac{1}{2}, 1\right)\right)$$





Logistic Regression

- Instead of computing

$$\mathbf{w} = \mathbf{w} - \frac{\mathbf{1}}{|\boldsymbol{B}|}\boldsymbol{\alpha X_B^T} \times (\boldsymbol{X_B} \times \mathbf{w} - \boldsymbol{Y_B})$$

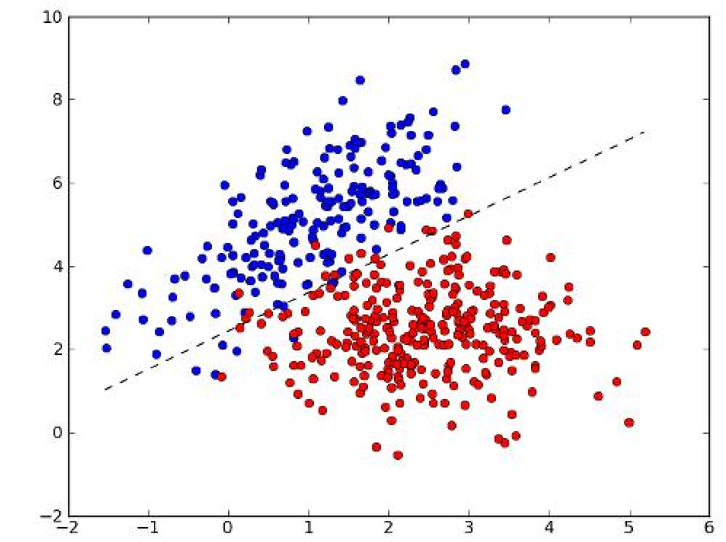
- We compute

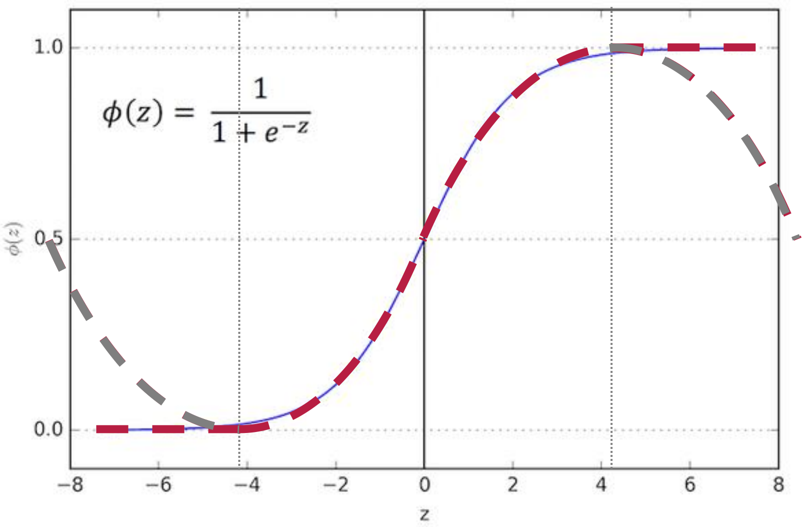
$$\mathbf{w} = \mathbf{w} - \frac{\mathbf{1}}{|\boldsymbol{B}|}\boldsymbol{\alpha X_B^T} \times (\boldsymbol{\phi}(\boldsymbol{X_B} \times \mathbf{w}) - \boldsymbol{Y_B})$$

- ϕ is very expensive to compute with just $+,*$ operations
- We build a piece-wise polynomial protocol for $\phi'(\cdot) \approx \phi(\cdot)$

$$\phi'(z) = \max\left(0, \min\left(z + \frac{1}{2}, 1\right)\right)$$

or higher degree approximation





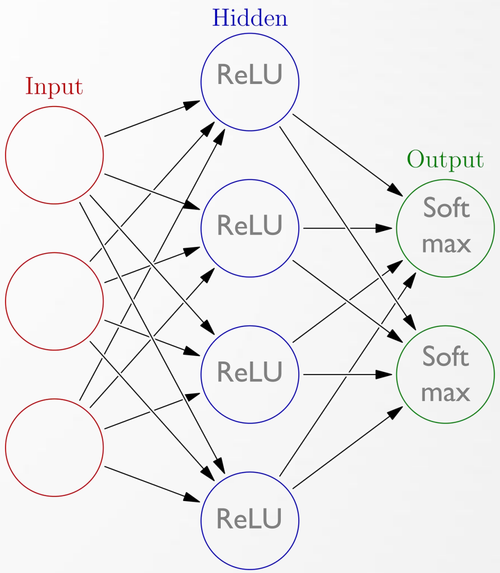
Neural Network

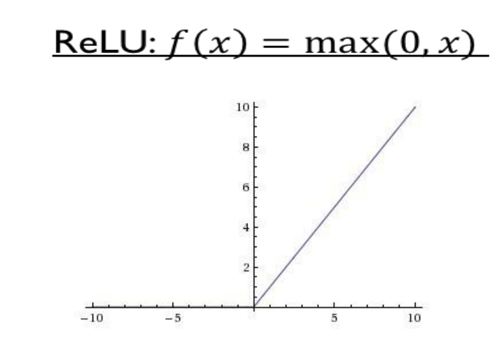
- Generalization of logistic regression
  - Each node is a regression problem
  - Replace logistic function with ReLU.
    - Easy to implement with piecewise polynomial

  - Output nodes use soft-max:

$$\sigma_i(\vec{x}) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

    - For inference/prediction, replace with arg-max.
    - For training,
      - Approximate e^x
      - Approximate $1/x$ or use one garbled circuit





Performance – Inference

The models are for the MNIST dataset with D = 784 features.

Model	Protocol	Batch Size	Running Time (ms) Online	Running Time (ms) Total	Comm. (MB)
Linear 93%	This	1	0.1	3.8	0.002
		100	0.3	4.1	0.008
	SecureML [47]	1	0.2	2.6	1.6
		100	0.3	54.2	160
Logistic* 98%	This	1	0.2	4.0	0.005
		100	6.0	9.1	0.26
	SecureML [47]	1	0.7	3.8	1.6
		100	4.0	56.2	161
NN 97%	This	1	3	8	0.5
	SecureML [47]	1	193	4823	120.5
CNN 99%	This*	1	6	10	5.2
	Chameleon [50]	1	1360	2700	12.9
	MiniONN [43]	1	3580	9329	657.5

Performance – Logistic Regression Training

- Measures iterations / second, **larger = better**
- Dimension = # of features
- Batch Size B = # examples used at each iteration

- Total running time: Up to **700 times faster**
- Communication: Up to **600 times less**

Dimension	Protocol	Batch Size B							
		Online				Online + Offline			
		128	256	512	1024	128	256	512	1024
10	This	2251	2053	1666	1245	2116	1892	1441	1031
	[47]	188	101	41	25	37	20	8.6	4.4
100	This	1867	1375	798	375	1744	1276	727	345
	[47]	183	93	46	24	3.6	1.9	1.1	0.6
1000	This	349	184	95	42	328	177	93	41
	[47]	105	51	24	13.5	0.43	0.24	0.12	0.06

Summary – ABY^3

- New Protocols:
  - Efficient support for fixed-point arithmetics
  - Improved matrix multiplication
  - Efficient piece-wise polynomial evaluation
  - Conversions between **A**rithmetic, **B**oolean, and **Y**ao secret shares
- Prototype machine learning implementation:
  - Linear Regression – 12000 iterations / second
  - Logistic Regression – 2000 iterations / second
  - Neural Networks – 10 millisecond inference

The End,

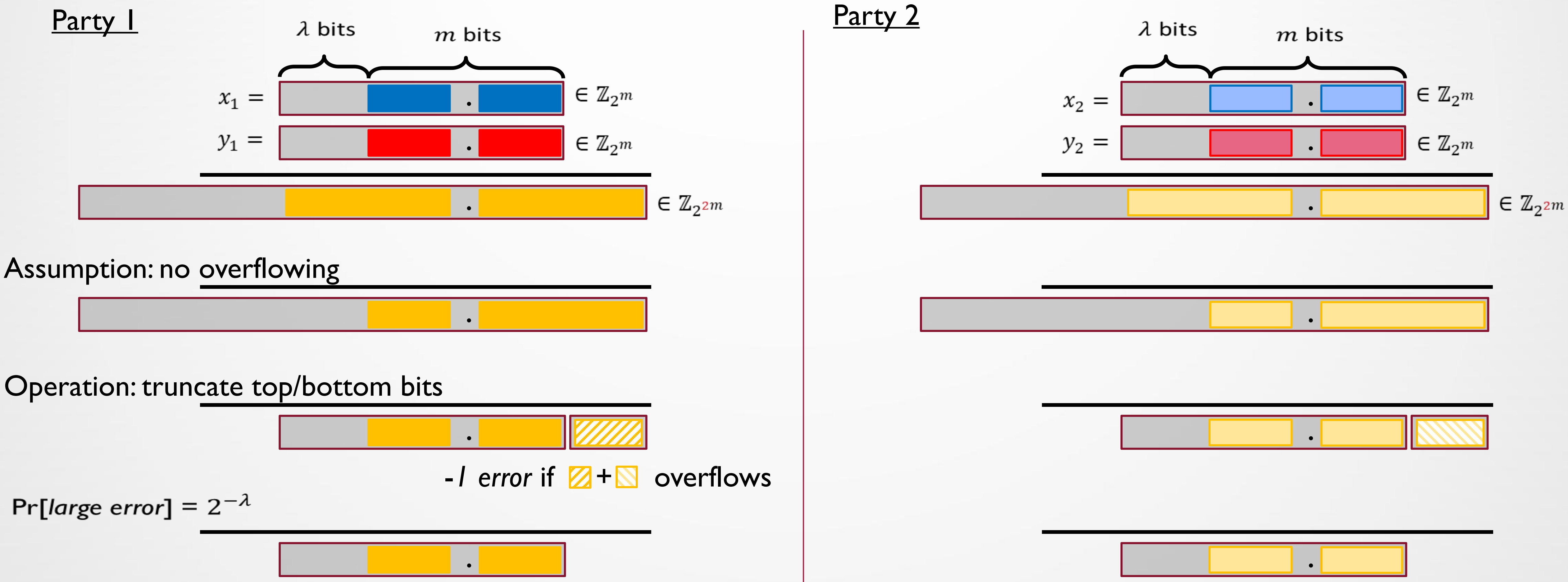
Questions?

Peter Rindal
Payman Mohassel

VISA Research

Truncation on 2-out-of-2 Secret Shares

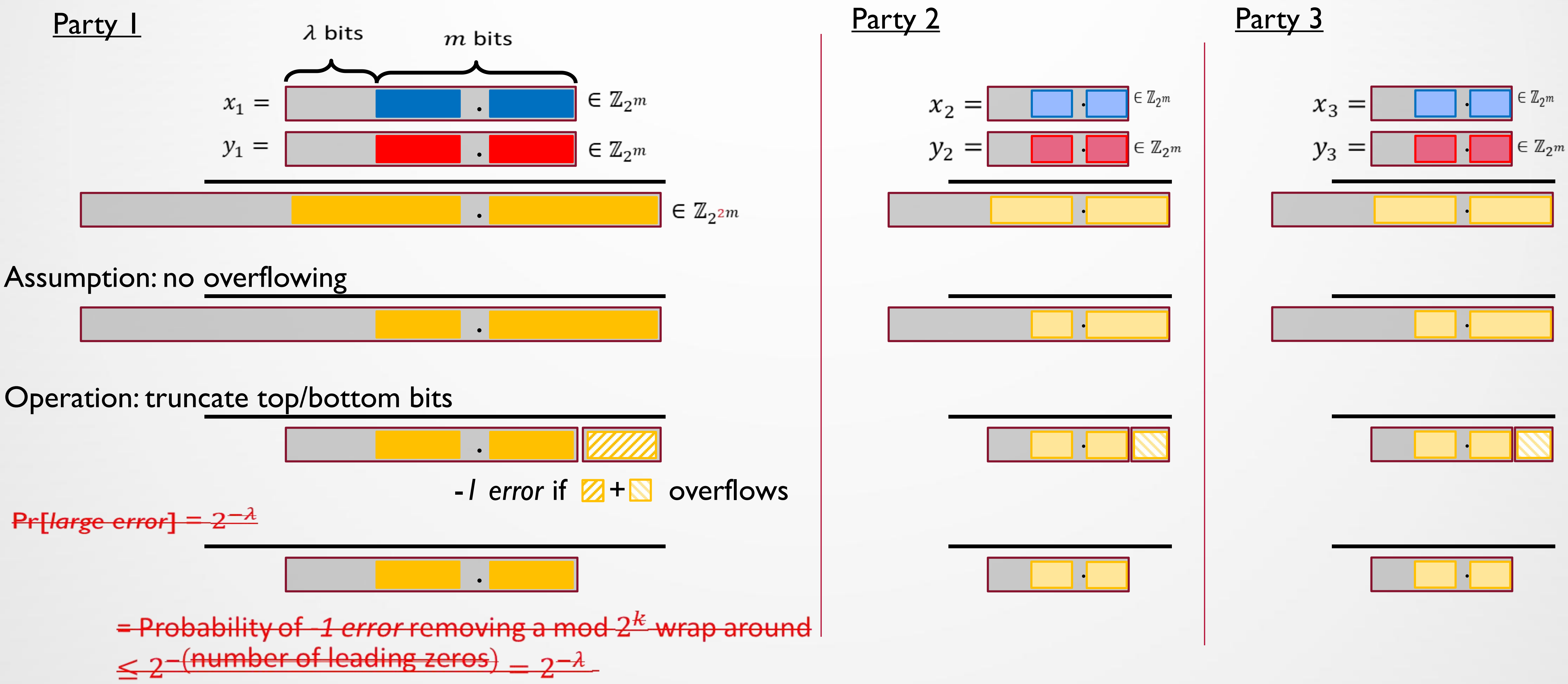
[MohasselZhang17]



= Probability of *-1 error* removing a mod 2^k wrap around

$\leq 2^{-(\text{number of leading zeros})} = 2^{-\lambda}$

Truncation on 2-out-of-3 Secret Shares



Truncation/Multiplication on 2-out-of-3 Secret Shares

- Multiplication $z = x * y$:
  - Preprocess
    - $[\![r']\!] = rand()$
    - $[\![r]\!] = [\![r']\!]/d$
  - Let $[\![z']\!] = [\![x]\!]\ [\![y]\!]$ (integer multiplication)

$$z_1' = x_1y_1 + x_1y_3 + x_3y_1$$
$$z_2' = x_1y_2 + x_2y_2 + x_2y_2$$
$$z_3' = x_2y_3 + x_3y_2 + x_3y_3$$

  - Compute

$$[\![z]\!] = \frac{reveal([\![z']\!] - [\![r']\!])}{d} + [\![r]\!]$$

VISA

$x_1, y_1, z_1' - r_1'$
$x_3, y_3,$

$z' - r'$

$x_1, y_1,$
$x_2, y_2, z_2' - r_2'$

Bank of America

$x_2, y_2,$
$x_3, y_3, z_3' - r$

amazon

Truncation/Multiplication on 2-out-of-3 Secret Shares

- Multiplication $z = x * y$:
  - Preprocess
    - $[\![r']\!] = rand()$
    - $[\![r]\!] = [\![r']\!]/d$
  - Let $[\![z']\!] = [\![x]\!]\ [\![y]\!]$ (integer multiplication)

$$z_1' = x_1y_1 + x_1y_3 + x_3y_1$$
$$z_2' = x_1y_2 + x_2y_2 + x_2y_2$$
$$z_3' = x_2y_3 + x_3y_2 + x_3y_3$$

  - Compute

$$[\![z]\!] = \frac{z' - r'}{d} + \frac{[\![r']\!]}{d}$$

VISA

x_1, y_1, z_1'
$x_3, y_3,$

$z' - r'$

$x_1, y_1,$
x_2, y_2, z_2'

Bank of America

$x_2, y_2,$
x_3, y_3, z_3'

amazon

Performance – Linear Regression Training

- Measures iterations / second, **larger = better**
- Dimension = # of features
- Batch Size B = # examples used at each iteration

Dimension	Protocol	Batch Size B							
		Online Throughput				Online + Offline Throughput			
		128	256	512	1024	128	256	512	1024
10	This	11764	10060	7153	5042	11574	9803	6896	4125
	[47]	7889	7206	4350	4263	47	25	11	5.4
100	This	5171	2738	993	447	5089	2744	1091	470
	[47]	2612	755	325	281	3.7	2.0	1.1	0.6
1000	This	406	208	104	46	377	200	100	46
	[47]	131	96	45	27	0.44	0.24	0.12	0.06