Privacy Preserving Machine Learning

Emil Hørning & Alexander Ramos

Supervisor: Bernardo Machado David

TABLE OF CONTENTS

01

Privacy Preserving Machine Learning

A brief introduction

Privacy Preserving Machine Learning

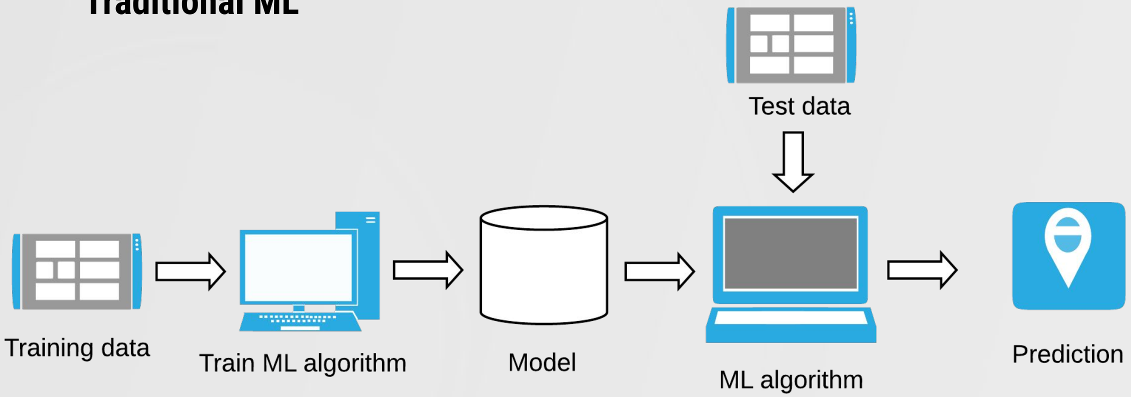
- Data with X number of rows (training examples), and Y number of columns (features)
- Label controls the training
- ML algorithm learns a hidden function which describes the data

Input	Blood pressure	Fat%	...	Label
Alex	132	20	...	Unhealthy
Emil	92	8	...	Healthy
...	...	...	...	...

Privacy Preserving Machine Learning

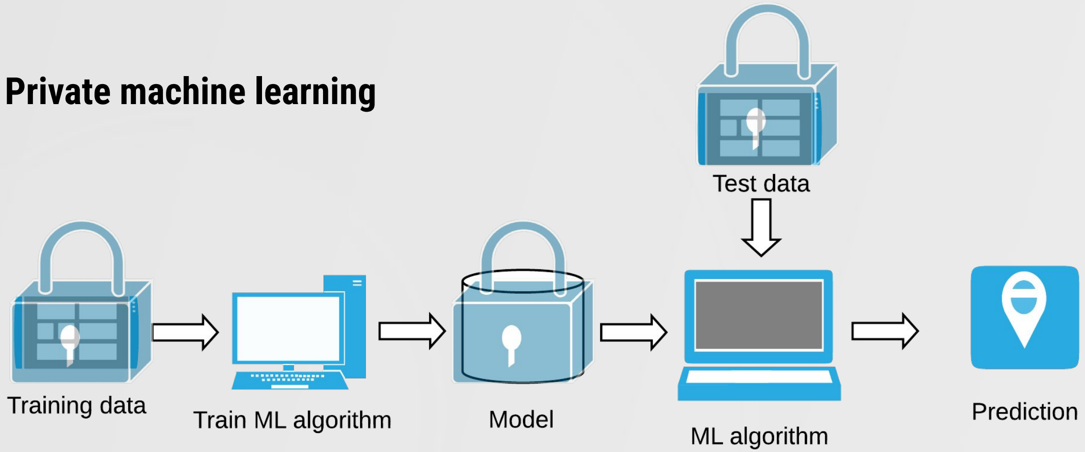
Traditional ML

- Data holders outsource ML
- Data may be sensitive



Privacy Preserving Machine Learning

- Data holders outsource ML
- Data may be sensitive



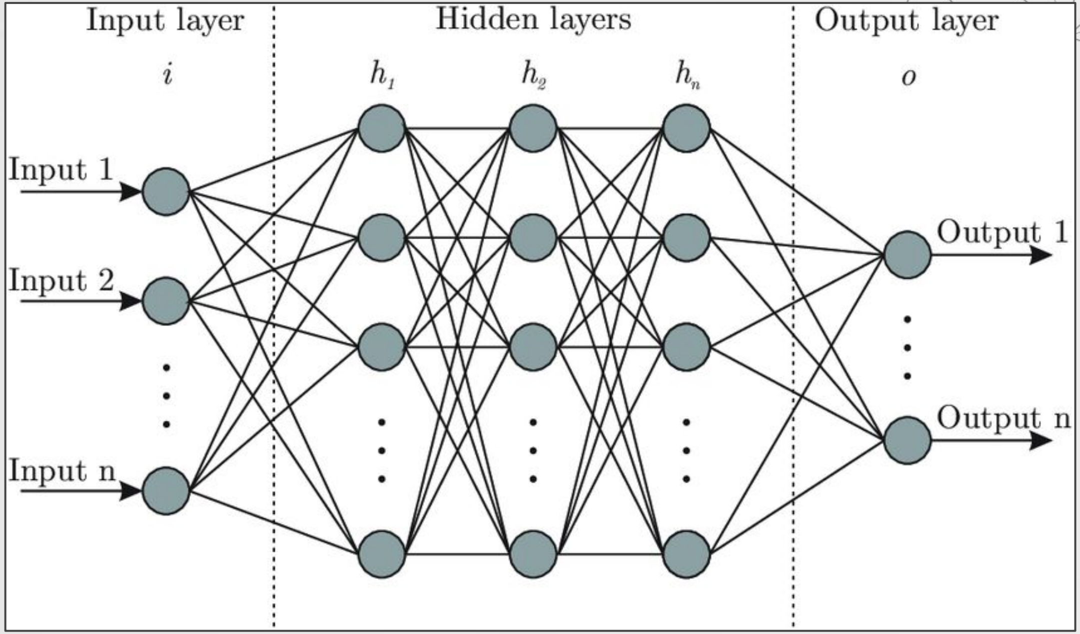
02

Background theory

(Deep) neural networks
Multi party computation/secret shares
Types of security/adversarial models

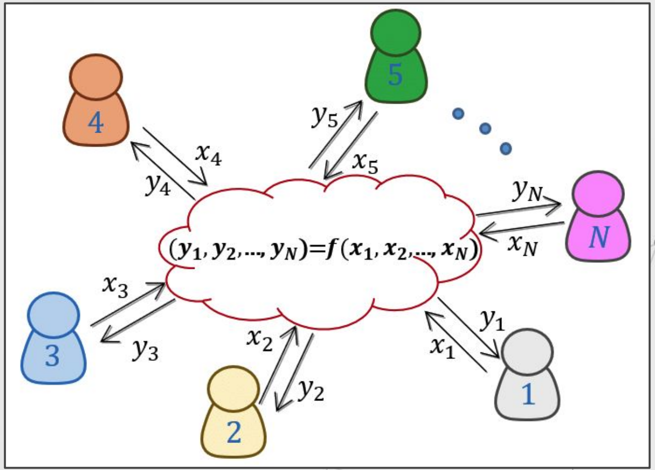
02: (Deep) Neural Networks

- A neural network estimates the parameters of “the hidden” function which describes a dataset
- Consists of **input, hidden,** and **output** layers
- Weights (parameters) are the edges
- **Forward propagation**
    - Linear combination of input values and weights $\Rightarrow$ activation function
    - Matrix multiplication
- **Backpropagation**
    - Adjust weights according to output and true label



02: Secure Multi-Party Computation

- N parties want to compute some function without revealing content of own input
- Functions are implemented as protocols that are private and secure
- Such MPC protocols can be used to do private machine learning



02: Types of security

Input privacy

The input remains private and no party learns the secret of other parties

Correctness

Any number of colluding dishonest parties participating in the protocol, should not be able to, by deviating from the protocol, force or trick an honest party to output something incorrect

Robust

Protocol achieves this in any scenario

Abort

Honest party detects deviation and aborts

02: Adversarial model

Passive adversary

- Adversary that corrupts 1 party and observes the protocol
- Tries to learn secrets
- Follows the protocol dutifully

Active adversary

- Adversary that corrupts 1 party and deviates from the protocol
- Tampers with inputs and outputs to alter result of protocol
- Tampers with inputs and outputs to learn secrets
- May halt the protocol to prevent completion

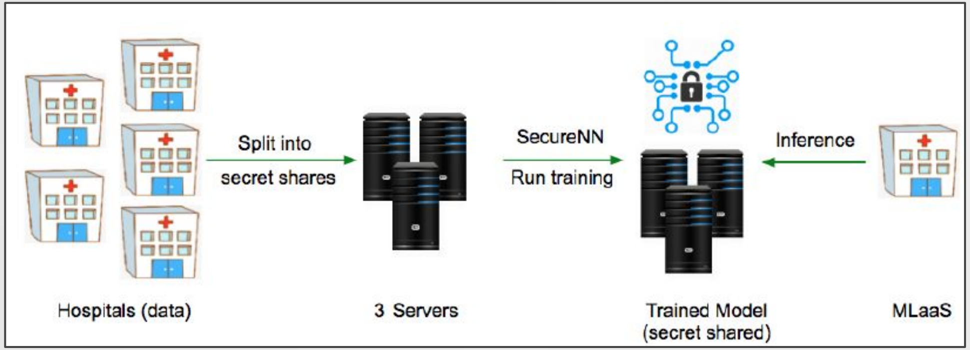
03

SecureNN

SecureNNs PPML approach
Security guarantees
Protocol structure
End-to-end protocol
Communication bottlenecks

03: SecureNNs PPML approach

- MPC protocols (3 parties)
- Input ⇨ secret shares ⇨ send to the 3 parties
- Run interactive protocol to train a neural network
- Trained model retained as secret shares
- Private prediction



03: Security guarantees

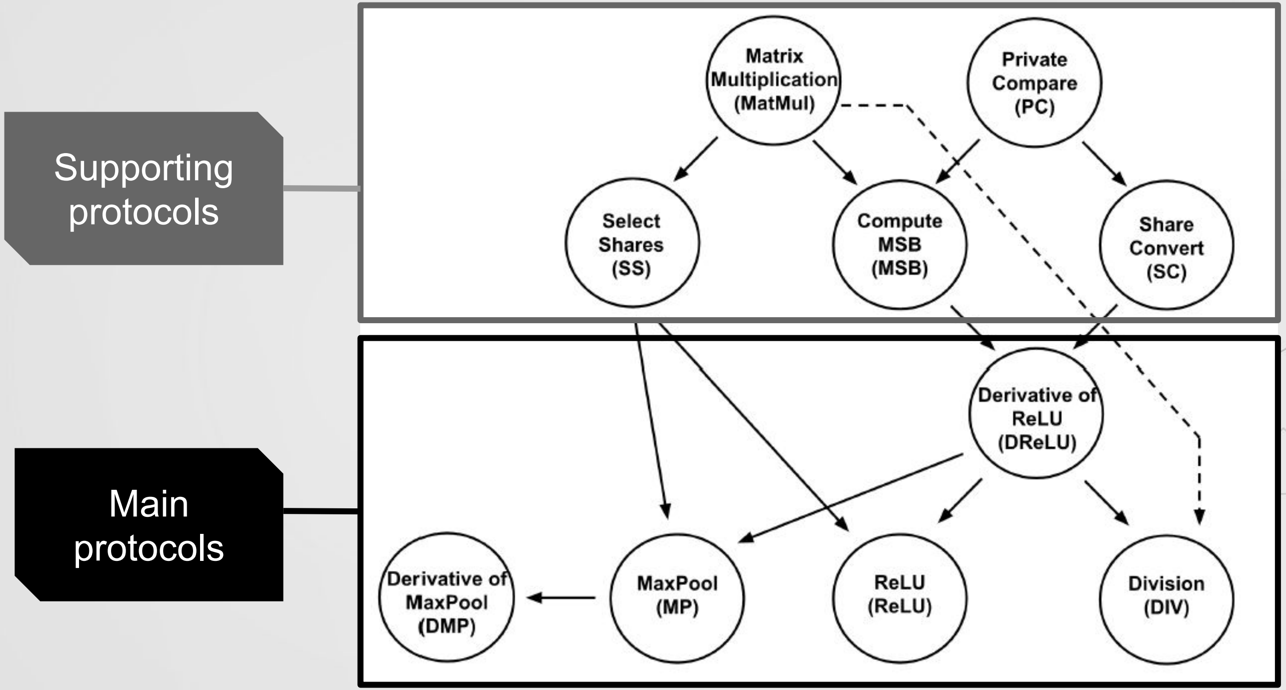
Passive adversary

- SecureNN provides full input privacy against a single passive corrupted party
- Proved with simulation proof
- Correctness of the protocol follows trivially

Active adversary

- SecureNN provides full input privacy against an active adversary corrupting 1 party
- SecureNN can't provide correctness for an active adversary

03: Protocol structure



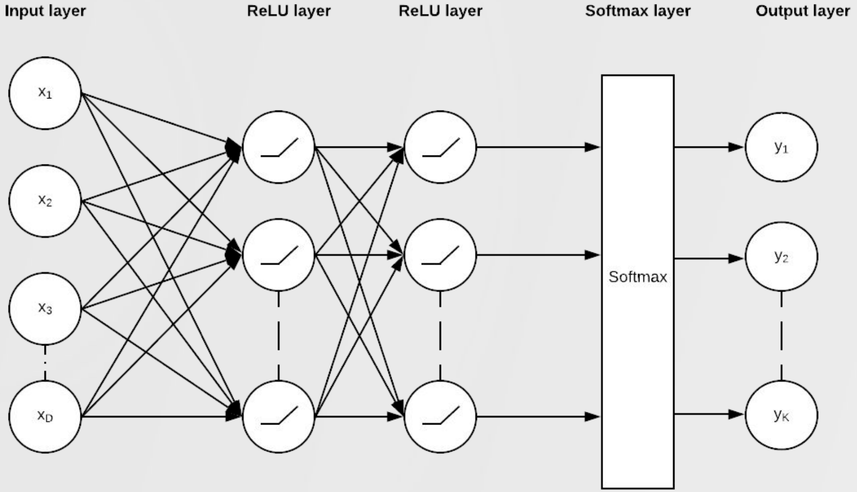
03: End-to-end protocol

- Construct a NN using a sequence of protocol calls

- E.g. a 3 layer network with a softmax output layer

$$\boxed{ASM(u_i) = \frac{ReLU(u_i)}{\sum ReLU(u_i)}}$$

1. Call MatMul
2. Call ReLU
3. Call MatMul
4. Call ReLu
5. Call DIV



03: Communication bottlenecks

- Function **DReLU** uses 9 rounds of communication
- Uses trick to obtain MSB by converting to odd ring

Algorithm 6 ReLU′, $\Pi_{\mathsf{DReLU}}(\{P_0, P_1\}, P_2)$:

Input: P_0, P_1 hold $\langle a\rangle_0^L$ and $\langle a\rangle_1^L$, respectively.

Output: P_0, P_1 get $\langle \mathsf{ReLU}'(a)\rangle_0^L$ and $\langle \mathsf{ReLU}'(a)\rangle_1^L$.

Common Randomness: P_0, P_1 hold random shares of 0 over $\mathbb{Z}_L$, denoted by u_0 and u_1 resp.

1: For $j \in \{0,1\}$, parties P_j computes $\langle c\rangle_j^L = 2\langle a\rangle_j^L$.

2: P_0, P_1, P_2 run $\boxed{\Pi_{\mathsf{SC}}(\{P_0, P_1\}, P_2)}$ with P_0, P_1 having inputs $\langle c\rangle_j^L$ & $\langle c\rangle_1^L$ & P_0, P_1 learn $\langle y\rangle_0^{L-1}$ & $\langle y\rangle_1^{L-1}$, resp.

3: P_0, P_1, P_2 run $\boxed{\Pi_{\mathsf{MSB}}(\{P_0, P_1\}, P_2)}$ with $P_j, j \in \{0,1\}$ having input $\langle y\rangle_j^{L-1}$ & P_0, P_1 learn $\langle \alpha\rangle_0^L$ & $\langle \alpha\rangle_1^L$, resp.

4: For $j \in \{0,1\}$, P_j outputs $\langle \gamma\rangle_j^L = j - \langle \alpha\rangle_j^L + u_j$.

Protocol	Rounds	Communication
$\mathsf{MatMul}_{m,n,v}$	2	$2(2mn + 2nv + mv)\ell$
$\mathsf{MatMul}_{m,n,v}$ (with PRF)	2	$(2mn + 2nv + mv)\ell$
SelectShare	2	5ℓ
PrivateCompare	1	$2\ell \log p$
ShareConvert	$\boxed{4}$	$4\ell \log p + 6\ell$
Compute MSB	$\boxed{5}$	$4\ell \log p + 13\ell$

04

Improving SecureNN

Initial attempt
Alternative approach (The Genome Paper)
Bit-decomposition
Optimised bit-decomposition (ComposeNet)

04: Initial attempt

- Remove ‘Share Convert’ and assume shares to be in Z_{L-1} already
- First month was spent trying to compile/run SecureNN code
- We were in contact with main author Sameer Wagh, but it wasn’t very helpful
- Later we realized our logic was wrong about removing ‘Share Convert’ and started looking for other ways to optimize ComputeMSB

04: Alternative PPML approach

High Performance Logistic Regression for Privacy-Preserving Genome Analysis

Martine De Cock, Rafael Dowsley, Anderson C. A. Nascimento, Davis Railsback, Jianwei Shen, Ariel Todoki

Abstract—In biomedical applications, valuable data is often split between owners who cannot openly share the data because of privacy regulations and concerns. Training Machine Learning models on the joint data without violating privacy is a major technology challenge that can be addressed by combining techniques from Machine Learning and cryptography. When collaboratively training Machine Learning models with the cryptographic technique named secure Multi-Party Computation, the price paid for keeping the data of the owners private is an increase in computational cost and runtime. A careful choice of Machine Learning techniques, algorithmic and implementation optimizations are a necessity to enable practical secure Machine Learning over distributed data sets. Such optimizations can be tailored to the kind of data and Machine Learning problem at hand.

Our setup involves secure Two-Party Computation protocols, along with a trusted initializer that distributes correlated randomness to the two computing parties. We use a gradient descent based algorithm for training a logistic regression model, and we break down the algorithm into corresponding cryptographic protocols. Our main contributions are a new protocol for computing the activation function that requires neither secure comparison protocols nor Yao's garbled circuits, and a series of cryptographic engineering optimizations to improve the performance. To the best of our knowledge, we present the fastest existing secure Multi-Party Computation implementation for training logistic regression models on high dimensional genome data distributed across a local area network.

For our largest gene expression data set, we train a model that requires over 7 billion secure multiplications; the training completes in about 26.90 seconds in a local area network. The implementation in this work is a further optimized version of the implementation with which we won first place in Track 4 of the iDASH 2019 secure genome analysis competition.

Index Terms—Logistic regression, Gradient descent, Machine Learning, Secure Multi-Party Computation, Gene expression data

I. BACKGROUND

A. Introduction

Machine Learning (ML) has many applications in the biomedical domain, such as medical diagnosis and personalized medicine. Biomedical data sets are typically characterized by high dimensionality, i.e. a high number of features such as lab test results or gene expression values, and low sample size, i.e. a small number of training examples corresponding to e.g. patients or tissue samples. Adding to these challenges, valuable training data is often split between parties (*data owners*) who cannot openly share the data because of privacy regulations and concerns. Due to these concerns, privacy-preserving solutions, using techniques such as secure Multi-Party Computation (MPC), become important so that this data can still be used to train ML models, perform a diagnosis, and in some cases even derive genomic diagnoses [25].

We tackle the problem of training a binary classifier on high dimensional gene expression data held by different data owners, while keeping the training data private. This work is directly inspired by Track 4 of the iDASH 2019 secure genome analysis competition[1]. The iDASH competition is a yearly international competition for participants to create and implement privacy-preserving protocols for applications with genomic data. The goal is in evaluating the best-known secure methods and advancing new techniques to solve real-world problems in handling genomic data. In the 2019 edition there were a total of four different tracks, where Track 4 invited participants to design MPC solutions for collaborative training of ML models originating from multiple data owners. One of the Track 4 competition data sets consists of 470 training examples (records) with 17,814 numeric features, while the other consists of 225 training examples with 12,634 numeric features. An initial 5-fold cross-validation analysis in the clear, i.e. without any encryption, indicated that in both cases logistic regression (LR) models are capable of yielding the level of prediction accuracy expected in the competition, prompting us to investigate MPC-based protocols for secure LR training.

The competition requirements implied the existence of multiple data owners who each send their training example(s) in an encrypted or secret shared form to *data processors* (computing nodes), as illustrated in Figure 1. The *honest-but-curious* data processors are not to learn anything about the data as they engage in computations and communications with each other. At the end, they disclose the trained classifier – in our case, the coefficients of the LR model – to the data owners. Since the data processors cannot learn anything about the values in the data set, this implies that our protocol is applicable in a wide range of scenarios, independently of how the original data is split by ownership. Our protocol works in scenarios where the data is horizontally partitioned, i.e. when each data owner

Martine De Cock, Anderson C. A. Nascimento, Davis Railsback, Jianwei Shen and Ariel Todoki are with the School of Engineering and Technology, University of Washington Tacoma. Emails: {mdecock, andclay, drail, sjw-james, atodoki}@uw.edu.
Martine De Cock is a Guest Professor at Ghent University.
Rafael Dowsley is with the Department of Computer Science, Bar-Ilan University, Israel. Email: rafael@dowsley.net. He is supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

[1] http://www.humangenomeprivacy.org/2019/competition-tasks.html, accessed on Jan 19, 2020

- The Genome Paper* released during our thesis
- Presents 2-PC method for performing logistic regression
- Same method can be used for private neural networks
- Won first prize in Track 4 of iDash 2019
- Secure bit-decomposition to obtain shares of MSB

*"*High performance logistic regression for privacy preserving genome analysis*" by De Cock et. al.

04: Bit-decomposition

- Protocols exists that produces XOR shares of all bits of a shared value.

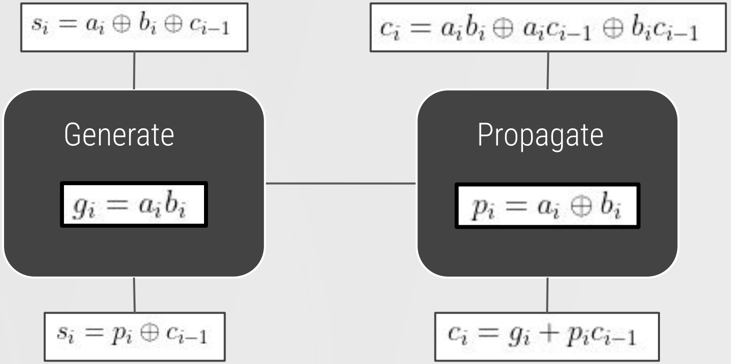
- Utilize this to get shares of MSB.

- Genome Paper presents 3 Bit Decomposition protocols for ƛ-bit integers
  - Linear Adder circuit (Linear, ƛ rounds)
  - Speculative optimized (2 + $\log_2$(ƛ) rounds)
  - Hyper Optimized ComposeNET (1 + $\log_2$(ƛ) rounds) (BitDecompOPT)

- We implemented ComputeMSB and BitDecompOPT to compare.

04: BitDecompOPT

- Based on the speculative method
    - Logarithmic rounds
- **Observation**: The carry bit depends on 2 signals
- Computing all matrix compositions M_1 to M_i yields the carry vector in upper right hand entry of resulting matrix

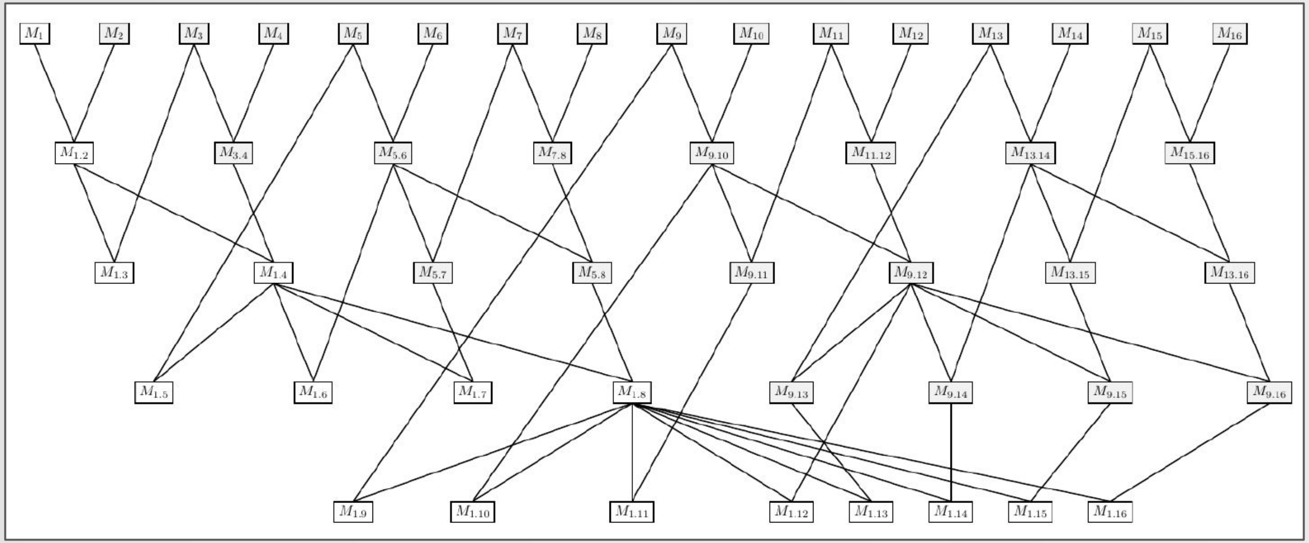


$$\begin{bmatrix} c_i \\ 1 \end{bmatrix} = \begin{bmatrix} p_i & g_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} c_{i-1} \\ 1 \end{bmatrix} = M_i \begin{bmatrix} c_{i-1} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} c_i \\ 1 \end{bmatrix} = \begin{bmatrix} p_i & g_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{i-1} & g_{i-1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} c_{i-2} \\ 1 \end{bmatrix} = M_i M_{i-1} \begin{bmatrix} c_{i-2} \\ 1 \end{bmatrix}$$

04: BitDecompOPT

- Matrix composition network: **ComposeNet**
- Logarithmic depth network



05

Experiments

Our implementation
Design of experiments

05: Our implementation

Architectural implementations

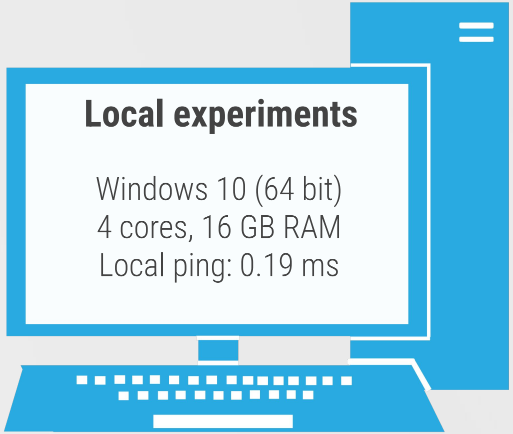
- Secret share logic
- Data structures for handling integers in Z_L
- Networking infrastructure

Protocol implementations

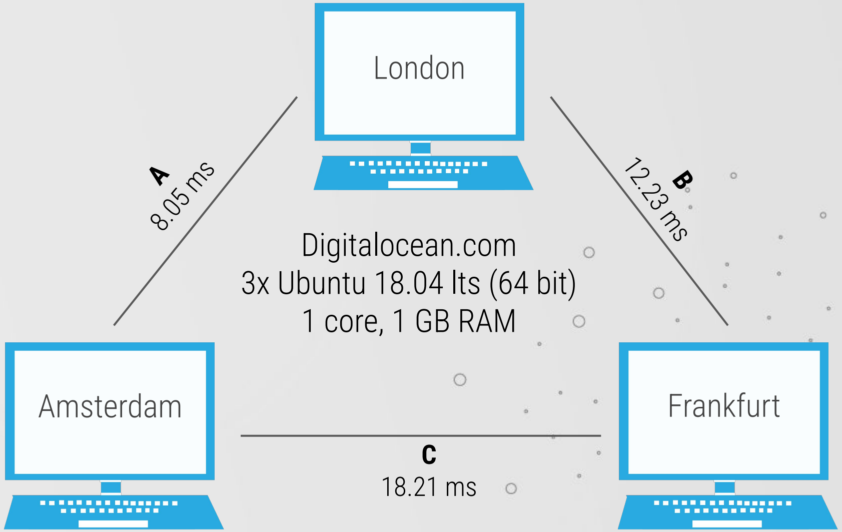
- Multiplication 2-party: Single-MatMul, Single-Mult, List-MatMul, List-Mult
- Multiplication 3-party: Single-Mult
- SC (Share Convert)
- PC (Private Compare)
- ComputeMSB
- ComposeNET
- BitDecomp
- BitDecompOPT

05: Design of experiments

- Bitlength L = 2^{64}, and prime p = 67
- Each protocol was run 1000 times



Distributed experiments



06
Results

06: Results

Local vs distributed

	Local time (s)		Dist. time (s)		Comm (bytes)	
Input size →	1	1000	1	1000	1	1000
BitDecompOPT A	-	-	0.0686	68.58	1674	1674000
BitDecompOPT B	-	-	0.0726	72.58	1674	1674000
BitDecompOPT C	-	-	0.0825	82.47	1674	1674000
BitDecompOPT Avg	0.0304	30.48	0.0745	74.54	1674	1674000
SC + ComputeMSB	0.0221	22.04	0.1015	101.52	1987.493	1987493

Raw local computation

	time (s)	
Input size →	1	1000
SC + ComputeMSB	0.000147	0.147
BitDecompOPT	0.0111	11.1

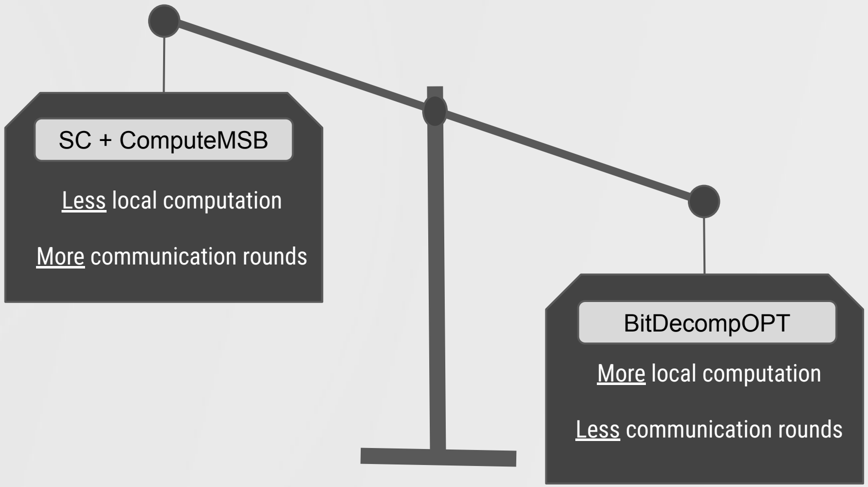
07

Discussion

ComputeMSB vs. BitDecompOPT - local setting
Theoretical vs. actual communication
Combining approaches
Utilizing full bit-decomposition

07: ComputeMSB vs BitDecompOPT - Local setting

- SecureNN's ComputeMSB runs faster than BitDecompOPT in a local setting
- **Tradeoff**: Local computation and communication rounds
- BitDecompOPT introduces significant computational overhead.



07: Theoretical vs actual Communication

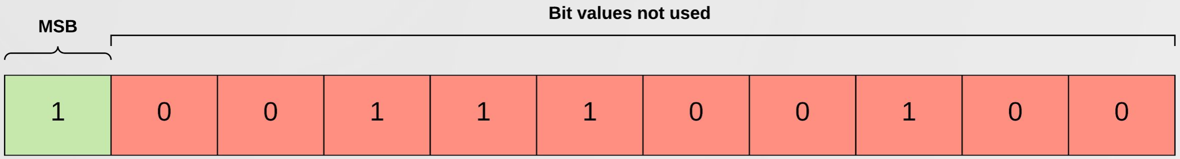
- Theoretical communication is based on the assumption that the implementation executes with no overhead.
- Requires intricate knowledge of low level programming and networking, which was out of scope for this project.
- Our results are achieved from an implementation that is far from the theoretical data transfer.

07: Combining approaches

- Implement BitDecomOPT in SecureNN to reduce DReLU rounds 9 $\Longrightarrow$ 7
- Both SecureNN and The Genome Paper claim same security.
- Problem with 3 party $\Longrightarrow$ 2 party
- SecureNN already uses a trusted initializer for common randomness, use it for beaver triplets to properly implement BitDecompOPT

07: Utilizing full bit-decomposition

- Bit-decomposition faster than computing MSB directly
- **But** needs to compute the full bit-decomposition of a value to obtain MSB
- Computes ƛ-1 bit values which are not used for anything else
- Further optimize by utilizing values computed during bit-decomposition
  - Other activation functions?



THANKS

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**.

Please keep this slide for attribution.