Secure Multi-party Computation

NISHANTH CHANDRAN
DIVYA GUPTA

The views expressed here are solely those of the author in his private capacity and do not in any way represent the views of Microsoft, or any other entity of Microsoft.
Can hospitals compute joint statistics on their databases without revealing patient information to one another?
Private Set Intersection

- Realty companies find list of customers who have double listed
- Can they do so without revealing individual customer names to each other?
A way to solve this problem

Can trust in other parties be completely removed?
Secure Two/Multi-party Computation (MPC) [Yao86, GMW87, BGW88, CCD88]

- $n$ parties, $t$ corruptions
- $P_i$ has input $x_i$
- Goal is to compute $f(x_1, x_2, \ldots, x_n)$
- Correctness: Execute protocol to compute $f(x_1, x_2, \ldots, x_n)$ correctly
- Security: Particles should not learn anything* about other parties’ inputs
• What is security in 2PC/MPC?

• Boolean Computation: Yao’s 2-party Garbling protocol

• Arithmetic Computation: Secret sharing and Beaver Triplets

• EzPC: Making MPC usable
Two-party Computation Security

Alice should not learn anything* about Bob’s input
Two-party Computation Security

What is our total net worth?

Net worth: X $

Net worth: Y $

Net worth:
Two-party Computation Security

Net worth: 
X $

f(x,y) = x + y

Net worth: 
Y $

f(x,y) = x + y

Alice should not learn anything* about Bob’s input; What does Alice learn?
Defining Security:
Alice *learns nothing more* than what can be learned from $x$ and $f(x,y)$

Secure Computation cannot prevent Alice from learning what she could have learned about Bob from the output (and her input)
Two-party Computation Security

Net worth: X $

Who is richer? (i.e., is X>Y ?)

Net worth: Y $

Alice and Bob learn if X>Y but nothing more
Two-party Computation Security

Alice and Bob execute a protocol to compute \( f(x,y) \)

Net worth: \( X \) $

Will Bob learn nothing about \( x \) (other than \( f(x,y) \)) even when he does not execute the protocol honestly?

Net worth: \( Y \) $
Two Kinds of Security – Semihonest vs Malicious

Net worth: $X$

Net worth: $Y$

Semihonest

- Security guaranteed when malicious party follows the protocol honestly

Malicious

- Security guaranteed even when malicious party does not follow the protocol honestly
Secure Multi-party Computation (MPC)

- Similar security notions
- Includes a corruption threshold $t < n$
- **Semihonest**: $t$ parties colluding do not learn any more information when they all follow the protocol honestly
- **Malicious**: $t$ parties colluding do not learn any more information even when they do not follow the protocol
Talk Outline

• What is security in 2PC/MPC?

• Boolean Computation: Yao’s 2-party Garbling protocol

• Arithmetic Computation: Secret sharing and Beaver Triplets

• EzPC: Making MPC usable
Boolean Computation

- All compute expressed as Boolean circuits (AND, XOR gates)

- Comparison, Bit-shifts etc. are most efficient when expressed as Boolean circuits

- Multiplication costs $O(l^2)$
Technique for 2 PC – Garbled Circuits [Yao86]

Garbler

\[ F(\mathbf{X}_1, \mathbf{X}_2) \]

\[ \mathbf{C} \]

Evaluator

\[ F(\mathbf{X}_1, \mathbf{X}_2) \]

\[ \mathbf{C} \]

\( (\text{Garbled circuit}) \)

\( (\text{Garbled inputs}) \)

\( (\text{PK operations}) \)

\( F(\mathbf{X}_1, \mathbf{X}_2) \)

\( F(\mathbf{X}_1, \mathbf{X}_2) \)

\( F(\mathbf{X}_1, \mathbf{X}_2) \)
How to Garble a gate? (E.g. NAND)

- Alice picks 2 random keys per wire (6 per gate).
- One key corresponds to 0 and the other to 1.
- If A = 0, then key = a₀.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

NAND Gate Truth

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₀</td>
<td>b₀</td>
<td>c₁</td>
</tr>
<tr>
<td>a₀</td>
<td>b₁</td>
<td>c₁</td>
</tr>
<tr>
<td>a₁</td>
<td>b₀</td>
<td>c₁</td>
</tr>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₀</td>
</tr>
</tbody>
</table>

Truth Table with Keys
How to Garble a gate? (E.g. NAND)

- Alice picks 2 random keys per wire (6 per gate).
- One key corresponds to 0 and the other to 1.
- If \( A = 0 \), then key = \( a_0 \).

How does Bob evaluate it?

**Truth Table with Keys**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Garbled NAND Gate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>c_1</td>
<td>( E_{a_0} (E_{b_0} (c_1)) )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>c_1</td>
<td>( E_{a_0} (E_{b_1} (c_1)) )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>c_1</td>
<td>( E_{a_1} (E_{b_0} (c_1)) )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>c_0</td>
<td>( E_{a_1} (E_{b_1} (c_0)) )</td>
</tr>
</tbody>
</table>

This ciphertext alone will decrypt correctly.
How does a Garbled Circuit look?

\[ GG_1 = \mathcal{E}_{a_0} (E_{b_1}(c_1) \mathcal{E}_{a_1} (E_{b_0}(c_1) \mathcal{E}_{a_0} (E_{b_0}(c_1) \mathcal{E}_{a_1} (E_{b_1}(c_0)))) \]

Output wire value: \( j_1 \) Decoded to 1 (by Alice)

\[ GG_2 = \mathcal{E}_{d_1} (E_{e_1}(f_0) \mathcal{E}_{d_0} (E_{e_0}(f_0) \mathcal{E}_{d_0} (E_{e_0}(f_0) \mathcal{E}_{d_1} (E_{e_0}(f_1)))) \]

\[ GG_3 = \mathcal{E}_{c_1} (E_{e_1}(h_1) \mathcal{E}_{c_0} (E_{c_1}(h_0) \mathcal{E}_{c_0} (E_{c_1}(h_1)))) \]

\[ GG_4 = \mathcal{E}_{h_0} (E_{i_0}(j_1) \mathcal{E}_{h_1} (E_{i_0}(j_0)) \mathcal{E}_{h_0} (E_{i_1}(j_0))) \]
Technique for 2 PC – Garbled Circuits [Yao86]

$$F$$

Garbler

$$c$$

Evaluator

$$(Garbled
circuit)$$

Oblivious Transfer

(PK operations)

$$(Garbled
circuit)$$

$$X_1, X_2$$

$$F(X_1, X_2)$$

$$F(X_1, X_2)$$

$$F(X_1, X_2)$$

$$F(X_1, X_2)$$
Oblivious Transfer \cite{Rabin81, EGL85}

$m_0, m_1$

\[ b \]

- Security 1: Alice does not learn $b$
- Security 2: Bob does not learn $m_{1-b}$
A protocol for Oblivious Transfer (OT) from (special) public-key encryption

\[ m_0, m_1 \]

\[ (pk_0, pk_1) \]

\[ b \]

\[ c_0 = \text{Enc}_{pk_0}(m_0), \quad c_1 = \text{Enc}_{pk_1}(m_1) \]

Security 1: Alice does not learn \( b \) because \( pk_0 \) and \( pk_1 \) are indistinguishable.

Security 2: Bob does not learn \( m_{1-b} \) because he does not know \( sk_{1-b} \).

- Pick \( (pk_b, sk_b) \) and \( pk_{1-b} \).
- Decrypt \( c_b \) to learn \( m_b \).
Where do OTs fit in Garbled Circuits?

\[ GG_1 = E_{c_0} (E_{b_1} (c_1) E_{b_0} (c_1) E_{b_0} (c_1) E_{b_1} (c_0)) \]

\[ GG_3 = E_{c_1} (E_{f_0} (h_1) E_{f_1} (h_0) E_{f_1} (h_0) E_{f_0} (h_1)) \]

\[ GG_2 = E_{d_1} (E_{e_1} (f_0) E_{d_0} (E_{e_0} (f_1) E_{d_0} (E_{e_0} (f_1) E_{d_1} (E_{e_0} (f_1)))) \]

\[ GG_4 = E_{h_0} (E_{i_0} (j_1) E_{h_1} (E_{i_0} (j_1)) E_{h_1} (E_{i_1} (j_0)) E_{h_0} (E_{i_1} (j_1)) \]

Output wire value: \( j_1 \)
Decoded to 1 (by Alice)

Evaluator Bob must learn keys corresponding to his input without Garbler Alice knowing Bob’s input.
Putting it all together

Garbler

Evaluator

\( F \)

(Garbled circuit)

\( c \)

Oblivious Transfer

(PK operations)

\( X_1, X_2 \)

\( X_1, X_2 \)

\( F(X_1, X_2) \)

\( F(X_1, X_2) \)

\( F(X_1, X_2) \)

\( F(X_1, X_2) \)
Why is the protocol secure (not easy)?

Garbler

Evaluator

\( F \)

\( C \)

(Garbled circuit)

\( F(X_1, X_2) \)

\( F(X_1, X_2) \)

Only depends on \( C \) - No information about Alice’s input

\( X_1, X_2 \)

(Oblivious Transfer (PK operations))

\( X_1, X_2 \)

\( F(X_1, X_2) \)
Why is the protocol not easy)

OT security says Alice does not learn Bob’s input and Bob learns only one key.

Garbler

\[ F \]

Evaluator

\[ X_1, X_2 \]

(Garbled circuit)

Oblivious Transfer

(PK operations)

\[ X_1, X_2 \]

(Garbled inputs)

\[ F(\mathcal{X}_1, \mathcal{X}_2) \]

\[ F(\mathcal{X}_1, \mathcal{X}_2) \]
Why is the protocol secure? (Not easy)

Garbler

Evaluator

\( F \)

\( C \)

(Garbled circuit)

Oblivious Transfer (PK operations)

\( X_1, X_2 \)

\( F(X_1, X_2) \)

\( F(X_1, X_2) \)

(Garbled inputs)

(Tricky) proof can show that Bob only learns one final key and no other information.
Why is the protocol secure? (Not easy)

Garbler

$F$

$C$

(Garbled circuit)

Evaluator

Oblivious Transfer
(PK operations)

$X_1, X_2$

(Garbled inputs)

Alice only sees one final key corresponding to $f(x_1, x_2)$
Why is the protocol secure? (Not easy)

- All (informal) security arguments made only against semihonest adversary.
- Malicious adversary protocols more complex.
Talk Outline

• What is security in 2PC/MPC?

• Boolean Computation: Yao’s 2-party Garbling protocol

• Arithmetic Computation: Secret sharing and Beaver Triplets

• EzPC: Making MPC usable
Secure Multi-party Computation & Applications to Private Machine Learning

DIVYA GUPTA

*THE VIEWS EXPRESSED HERE ARE SOLELY THOSE OF THE AUTHOR IN HIS PRIVATE CAPACITY AND DO NOT IN ANY WAY REPRESENT THE VIEWS OF MICROSOFT, OR ANY OTHER ENTITY OF MICROSOFT.*
Talk Outline

• Secure Computation for Arithmetic Circuits

• EzPC: Programmable, Efficient, and Scalable Secure Computation (applied to private machine learning)
Secure Computation of Arithmetic Circuits

- Arithmetic circuits have addition and multiplication gates

Protocol Summary:
- Alice and Bob start with 2-out-2 secret shares of input
- For a gate, given shares of input wires, run a protocol to compute shares of output wire
2-out-of-2 Secret sharing scheme

- Split secret into two parts $s_0, s_1$
- Single share reveals nothing about $s$
- Combine shares to get $s$

Example: Uniform shares s.t. $s_0 + s_1 = s$
Input sharing phase

• Each party shares its input with other party

Input $\mathbb{Z}_2^{32}$

Pick $\mathbf{x}_0$ s.t. $\mathbf{x}_0 + \mathbf{x}_1 = \mathbf{x}$

Output $\mathbf{y}_0$

Pick $\mathbf{y}_1$ s.t. $\mathbf{y}_0 + \mathbf{y}_1 = \mathbf{y}$
Addition gate

• Each locally adds the shares of input

Compute $c_0 = a_0 + b_0$

• Correctness:
• Security: trivial

Compute $c_1 = a_1 + b_1$

$\{a_0, b_0\}$

$\{a_1, b_1\}$

No Interaction
Multiplication gate

- Need setup such as "Beaver Triplet"
- Parties hold shares of random $x, y, z$ with $z = x \cdot y$

\[
\begin{align*}
  e_0 &= a_0 - x_0, \quad f_0 = b_0 - y_0 \\
  e_1 &= a_1 - x_1, \quad f_1 = b_1 - y_1 \\
  c_0 &= f \cdot a_0 + e \cdot b_0 + z_0 \\
  c_1 &= f \cdot a_1 + e \cdot b_1 + z_1 - e \cdot f
\end{align*}
\]

Reconstruct $e$ and $f$ independent of input and circuit!

Correctness? Security?
Secure Computation Protocols

• **Boolean circuits:** Garbled circuits [Yao], GMW, BMR, ..... (Good for expressing comparisons, bitwise operations, maximum, etc)

• **Arithmetic circuits:** Using beaver triplets [Beaver], BGW, CCD, SPDZ, ..... (Good for expressing multiplications and additions)

Very hard for non-crypto experts to select a good protocol for application!

Might need a mix of protocols!
Talk Outline

• Secure Computation for Arithmetic Circuits

• EzPC: Programmable, Efficient, and Scalable Secure Computation (applied to private machine learning)
  Joint work with Nishanth Chandran, Aseem Rastogi and Rahul Sharma
Many Challenges in using 2PC

• Very hard for developers to write secure 2PC applications

• Which protocol is best suited for my application?
  • GMW, Yao, BGW, BMR, ……

• How to express the function efficiently?
  • Circuits: Boolean vs Arithmetic

• Most protocols require low circuit level programming
  • Tedious and error-prone
Our Goal: Democratizing 2PC

- Programmer-friendly platform
  - Developer *only* specifies functionality
- Generality: Express arbitrary functionalities
- Performance: Automatically choose right circuit rep.
- Scale to practical tasks
- Formal guarantees of Correctness and Security

Function $F(x, y)$

Make 2PC accessible to developers
## Current state of affairs

**Alice:** \( w, b \)

**Bob:** \( x \)

**Function:** \( w^T x > b \)

<table>
<thead>
<tr>
<th>Option 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Program in one of the several DSLs such as Fairplay, Wysteria, ObliVM, CBMC-GC, SMCL, Sharemind, etc</td>
</tr>
<tr>
<td>• Pro: High-level programmer friendly framework</td>
</tr>
<tr>
<td>• Pro: Developer is oblivious of underlying crypto magic</td>
</tr>
<tr>
<td>• Cons: <strong>Poor performance</strong> (single circuit representation)</td>
</tr>
</tbody>
</table>

- Since complexity of 2PC protocol grows with circuit size, for performance, require Arithmetic circuit for comparison. Boolean circuit for comparison with
- **None** of the high-level frameworks support a mix of Arithmetic and Boolean circuit

---

43
Current state of affairs

Option 2

- Program in ABY framework (Demmler et al. NDSS-15)
- Pro: Uses a combination of Boolean and Arithmetic circuits
- Pro: Much better performance
- Cons: Not programmer friendly (low level)
  - Manually split compute into Boolean & Arithmetic
  - Write corresponding low-level circuits for each part
  - Insert inter-conversions between them
- Cons: Tedious and Error-prone + some
Current state of affairs

Function: $w^t x > b$

Option 3

- Design *specialized* protocols for functions of interest
- Pro: Good performance
- Cons: Requires a lot of *cryptographic expertise*
- Cons: **No generality**: Great effort for each function
## State of the art in 2PC

(for $F = (w^T x > b)$)

<table>
<thead>
<tr>
<th>Solution</th>
<th>Programmability</th>
<th>Generality</th>
<th>Performance</th>
<th>Scalability</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSLs like ObliVM, CBMC-GC, etc</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>ABY</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
</tr>
<tr>
<td>Specialized Protocols like MiniONN, etc</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>EzPC</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Our Approach</th>
</tr>
</thead>
</table>
Our tool: EzPC (Easy two-party computation)

- EzPC Source program
- EzPC Compiler
- ABY C++ Code
- 2PC protocol
- Scalability
- Formal Guarantees
- Evaluation
EzPC: Source Programs

Function: $w^T x > b$

- Base types and array types
- Mathematical operators (+, *, >, &, >>, ...)
- Statements for assignments, array read/write, bounded for loops and if conditions

```c
uint w[30] = input1();
uint x[30] = input2();
uint b = input1();

uint acc = 0;
for i in [0:30] {
    acc = acc + (w[i] * x[i]);
}

Output2(acc > b ? 1 : 0);
```

// circuit builders for arithmetic and boolean
Circuit* yc = s[S_YAO]->GetCircuitBuildRoutine();
Circuit* ac = s[S_ARITH]->GetCircuitBuildRoutine();
...

if(role == SERVER) {
    // Put gates to read w and b
} else { // role == CLIENT
    // Put gates to read x
}

for(uint32_t i = 0; i < 30; i++) {
    // acc = w^T x
    share * a_t_0 = ac->PutMULGate(a_w[i], a_x[i]);
    a_acc = ac->PutADDGate(a_acc, a_t_0);
}

// convert acc and b from arithmetic to boolean
share * y_acc = yc->PutA2YGate(a_acc);
share * y_b = yc->PutA2YGate(a_b);

share * y_pred = yc->PutGtGate(y_acc, y_b);
uint32_t one = 1;
share * y_1 = yc->PutCONSGate(one, bitlen);
uint32_t zero = 0;
share * y_0 = yc->PutCONSGate(zero, bitlen);
share * y_t = yc->PutMUXGate(y_pred, y_1, y_0);

share * y_out = yc->PutOUTGate(y_t, CLIENT);
party->ExecCircuit();

if(role == CLIENT) { // only to the client
    uint32_t _o = y_out->get_clear_value<uint32_t>();
}
```
EzPC: How the compiler works?

- EzPC source program → ABY code
- Problem: Automatically assigns variables and operators to Boolean or Arithmetic type
- Using cryptographic costs of primitive operators as well as inter-conversion costs

- Hard problem, can require exponential time
- Heuristics-based cryptographic cost-aware compiler

  - **Hard Constraints**: MULT in Arithmetic; GT/COND/BitwiseAND in Boolean
  - **Soft Constraints**: ADD can be either in Arithmetic or Boolean based on operands

  - Minimize inter-conversion cost
    - Maintain a map from variables to available types
EzPC: Cryptographic Cost-aware Compiler

- **Hard Constraints**: MULT in Arithmetic; GT/COND/BitwiseAND in Boolean
- **Soft Constraints**: ADD can be either in Arithmetic or Boolean based on operands

```c
uint w[30] = input1();
uint x[30] = input2();
uint b = input1();

uint acc = 0;
for i in [0:30] {
    uint temp = w[i] * x[i];
    acc = acc + temp;
}
Output2(acc > b ? 1 : 0);
```

Intermediate Program
(Annotate all variables & operators and insert inter-conversions)

```c
uint^A w[30] = input1();
uint^A x[30] = input2();
uint^B b = input1();

uint^A acc = 0;
for i in [0:30] {
    uint^A temp = w[i] * x[i];
    acc = acc + temp;
}

uint^B acc_B = A2B(acc);
Output2(acc_B > b ? 1 : 0);
```
EzPC: Cryptographic Cost-aware Compiler

- Minimize inter-conversions: Maintain a map from variables to available share type

```
uint acc = 0;
for i in [0:30] {
    uint temp = w[i] * x[i];
    acc = acc + temp;
}

uint o = (acc > b ? 1 : 0);
uint y = acc * w[0];
uint z = acc & b;
```

Intermediate Program

```
uint acc_A = 0;
for i in [0:30] {
    uint temp_A = w[i] * x[i];
    acc_A = acc_A + temp_A;
}

uint acc_B = A2B(acc);
uint o_B = (acc_B > b ? 1 : 0);
uint y_A = acc_A * w[0];
uint z_A = acc_A & b;
uint z_B = acc_B & b;
```
EzPC: Scalability

- Program needs to be written as circuit
  - Circuit needs to fit in memory

- Unroll the loops (Circuits don’t have loops)
  - Circuit size can be huge (>28 GB)

- Secure Code Partitioning
  - Partition into P1 and P2
  - Need to pass Acc to P2 securely
  - Secret-share Acc b/w Alice & Bob
  - P2 reconstructs Acc

- Very natural and crucial for benchmarks such as large DNNs, matrix factorization, etc.

Revealing Acc to Alice or Bob breaks security!
EzPC: Formal Guarantees

• **Correctness**
  • Formulate trusted party semantics and protocol semantics
  • For a well-typed $P$, both semantics
    • terminate without errors
    • produce same outputs
  • No array index out of bounds errors
EzPC: Formal Guarantees

• **Security**
  - Semi-honest security against corruption of one party
  - Honest-but-curious adversary that follows the protocol faithfully BUT is eager to learn more
  - Eavesdrop on communication
    • Learns nothing about Alice’s or Bob’s input
  - Corrupt Alice
    • Learn nothing about Bob’s input (beyond o/p)

- Formally reduce security of compiler to semi-honest security of 2PC back-end (ABY)
- Security of partitioning scheme
Applications of EzPC to Private Machine Learning
Secure Prediction using Secure 2PC

- Bob wants to learn output of classifier
- Solved by 2PC!
- Bob learns classifier output only
- Azure learns nothing about Bob’s input!

\[ F(\text{Model, Data}) \]

ML classifier for diabetes

Medical report

Data is private

Azure

2PC protocol
EzPC: Evaluation

• Demonstrate generality by evaluating EzPC on large variety of benchmarks

• In all cases, EzPC protocols BEAT/MATCH performance of state-of-the-art specialized protocols

• Writing benchmarks did not require any crypto know-how

• Lines of code (LOC) is proportional to C++ code for describing the ality

Generic 2PC protocols gives state-of-art performance (if done smartly)!
Deep Neural Networks

• Many layers; Each layer has
  • A linear operation that can be written as a matrix multiplication
  • A non-linear activation function such as Maxpool, ReLU, etc
• Matrix multiplication is suited for Arithmetic; non-linear function is suited to Boolean

• **Cryptonets** (ICML 16)
  • Based on Homomorphic Encryption (HE)
  • MNIST: 1 fully connected, 1 convolutional, square activation function

<table>
<thead>
<tr>
<th></th>
<th>DNN</th>
<th>Cryptonets Time (s)</th>
<th>EzPC Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptonets</td>
<td>297</td>
<td>0.6</td>
<td></td>
</tr>
</tbody>
</table>

2PC based approach much faster than HE!
## Our Evaluation

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Prev. Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve Bayes (Audiology)</td>
<td>3.9</td>
</tr>
<tr>
<td>Decision Trees (ECG)</td>
<td>0.4</td>
</tr>
<tr>
<td>SecureML (MNIST)</td>
<td>1.1</td>
</tr>
<tr>
<td>MiniONN (MNIST)</td>
<td>9.4</td>
</tr>
<tr>
<td>MiniONN CIFAR-10</td>
<td>544</td>
</tr>
<tr>
<td>Matrix Factorization</td>
<td>10440</td>
</tr>
<tr>
<td>SecureML (CIFAR-10)</td>
<td>546</td>
</tr>
<tr>
<td>MiniONN (CIFAR-10)</td>
<td>265</td>
</tr>
<tr>
<td>MiniONN CIFAR-10</td>
<td>2x</td>
</tr>
<tr>
<td>Matrix Factorization</td>
<td>19x</td>
</tr>
<tr>
<td>SecureML (CIFAR-10)</td>
<td>78</td>
</tr>
<tr>
<td>MiniONN (CIFAR-10)</td>
<td>154</td>
</tr>
<tr>
<td>MiniONN CIFAR-10</td>
<td>336</td>
</tr>
<tr>
<td>Matrix Factorization</td>
<td>1494</td>
</tr>
</tbody>
</table>
More ML classifiers in EzPC

- **TensorFlow tutorial benchmarks**
  - Softmax regression for MNIST: argmax
  - DNN for MNIST: 2 convolutional, 2 fully connected, ReLU activation, 99.2% accuracy

- **Bonsai (ICML 17):** Much smaller models for weak IoT devices, reasonable accuracy
  - Tree like structure, much smaller

### First 2PC implementations for these benchmarks

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Depth</th>
<th>LAN (1ms) Time (s)</th>
<th>WAN (40ms) Time (s)</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>USPS</td>
<td>2</td>
<td>0.2</td>
<td>0.9</td>
<td>156</td>
</tr>
<tr>
<td>WARD</td>
<td>3</td>
<td>0.3</td>
<td>1.1</td>
<td>283</td>
</tr>
</tbody>
</table>

### Demonstrates programmability and generality of EzPC
EzPC: In a nut-shell

• Developer friendly
  • Easy to get correct functionality

• Generality and Customizability
  • Small change in functionality requires small change in code

• State-of-the-art performance
  • Beats specialized protocols

• Scales to large programs

• Formal guarantees of correctness and security
Future Directions

- Generalize EzPC to more than 2 parties
  - Integrate existing MPC protocols to EzPC
  - Build new MPC protocols that combine Arithmetic and Boolean

- Malicious parties?

- Make language of EzPC more powerful
  - Enhance the expressiveness of the language with functions
  - Better support for floating point operations

- Find other exciting applications apart from private machine learning