Computing on Private Data: Private Genomics and More

David Wu
UT Austin
Computing on Private Data

Cryptographic tools: hide the input $x_i$ from the computing party [This talk]

Complementary goals as differential privacy (and oftentimes, can be combined)

Aggregate statistic $y \leftarrow f(x_1, \ldots, x_5)$

Differential privacy: ensure that output $y$ protects privacy of input $x_i$
Rare Disease Diagnosis

What gene causes a specific (rare) disease?

Patients with Kabuki Syndrome

Each patient has a list of 200-400 rare variants over \( \approx 20,000 \) genes
**Rare Disease Diagnosis**

Each patient has a list of 200-400 rare variants over ≈20,000 genes.

<table>
<thead>
<tr>
<th>Gene</th>
<th>A1BG</th>
<th>ZZ3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Each patient has a vector $\mathbf{v}$ where $v_i = 1$ if patient has a rare variant in gene $i$.

**Goal:** Identify gene with most variants across all patients.

Patients with Kabuki Syndrome

Each patient has a list of 200-400 rare variants over ≈20,000 genes.
Rare Disease Diagnosis

Each patient has a list of 200-400 rare variants over ≈20,000 genes.

A1BG

<table>
<thead>
<tr>
<th>Gene</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1BG</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

ZZZ3

<table>
<thead>
<tr>
<th>Gene</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZZZ3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Goal: Identify gene with most variants across all patients.

Each patient has a vector $\mathbf{v}$ where $v_i = 1$ if patient has a rare variant in gene $i$.

Patients with Kabuki Syndrome

Each patient has a list of 200-400 rare variants over ≈20,000 genes. Works well for Mendelian (monogenic) diseases (estimated to affect ≈10% of individuals).
### Rare Disease Diagnosis

#### Patients with Kabuki Syndrome

Each patient has a list of 200-400 rare variants over ≈20,000 genes.

#### Gene Representation

<table>
<thead>
<tr>
<th>Gene</th>
<th>A1BG</th>
<th>ZZZ3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Each patient has a vector $\nu$ where $\nu_i = 1$ if patient has a rare variant in gene $i$.

#### Goal

To identify causal rare variants, often need **exact** computation.

**Goal:** Identify gene with most variants across all patients.

Works well for Mendelian (monogenic) diseases (estimated to affect ≈10% of individuals).
Patients often in geographically-diverse locations

**Question:** Can we perform this computation without seeing complete patient genomes?
Rare Disease Diagnosis

[Image: Diagram showing two hospitals and patients with Kabuki Syndrome]

Patients with Kabuki Syndrome

Each patient has a list of 200-400 rare variants over \( \approx 20,000 \) genes

Patients “secret share” their data with two non-colluding hospitals
Rare Disease Diagnosis

Patients with Kabuki Syndrome

Each patient has a list of 200-400 rare variants over ≈20,000 genes

Hospitals run a multiparty computation (MPC) protocol on pooled inputs

Patients “secret share” their data with two non-colluding hospitals

[JWBBB17]
Rare Disease Diagnosis

Patients with Kabuki Syndrome

Each patient has a list of 200-400 rare variants over ≈20,000 genes

Top variants (sorted): KMT2D, COL6A1, FLNB

Known cause of disease
Rare Disease Diagnosis

Each patient has a list of 200-400 rare variants over ≈20,000 genes

MPC Protocol

Top variants (sorted): KMT2D, COL6A1, FLNB

Other variants that the patients possess are kept hidden
Rare Disease Diagnosis

General techniques apply to many different scenarios for diagnosing Mendelian diseases.

- Identify causal gene for a rare disease given a small patient cohort.
- Identify rare functional variants that are present in the child but in neither of the parents.
- Identify patients with the same rare functional mutation at two different hospitals.

Patients with Kabuki Syndrome

<table>
<thead>
<tr>
<th>Gene</th>
<th>A1BG</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>ZZZ3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Source: JWBBB17]
Rare Disease Diagnosis

Experimental benchmarks for identifying causal gene in small disease cohort
  • Simulated two non-colluding entities with 1 server on East Coast and 1 on West Coast

![Bar chart](chart.png)

- Number of Patients: 10, 50, 100
- End-to-End Time: 9.6 s, 13.7 s, 15.8 s
- Communication: 41.0 MB, 62.2 MB, 72.7 MB
Rare Disease Diagnosis

Experimental benchmarks for identifying causal gene in small disease cohort
  • Simulated two non-colluding entities with 1 server on East Coast and 1 on West Coast

For many rare disease diagnosis scenarios, disease cohort size can be very small (e.g., 5-10 patients)
Modern cryptographic tools enable useful computations while protecting the privacy of individual genomes.
Modern cryptographic tools enable useful computations while protecting the privacy of individual genomes.

Techniques apply to general computations over private data.
Yao’s Protocol for Two-Party Computation
Yao’s Protocol for Two-Party Computation

Private inputs

0
1
⋮
0

1
1
⋮
0

Security guarantee: everything the parties learn can be inferred from the output and their individual inputs

Classic protocol for two-party computation

[Yao82]
Yao’s Protocol for Two-Party Computation

Step 1: Model computation as a Boolean circuit

Private inputs

Party 1 “garbler”

AND

Party 1

AND

Party 2

NAND

Output

Party 2 “evaluator”

[Yao82]
Step 2: Garbler “encrypts” the circuit (i.e., “garbles” the circuit)

Garbler chooses two different encryption keys for every wire in the circuit

Each key is associated with a possible wire value

Yao’s Protocol for Two-Party Computation

[Yao82]
Step 2: Garbler “encrypts” the circuit (i.e., “garbles” the circuit)

Idea: Encrypt the output key (for the output wire) with the two input keys (for the input wires)

Garbler constructs a garbled truth table for each gate in the circuit
**Step 2:** Garbler “encrypts” the circuit (i.e., “garbles” the circuit)

\[
\text{Enc} \left( k_0^{(1)}, \text{Enc} \left( k_0^{(2)}, k_0^{(\text{out})} \right) \right)
\]

Garbler constructs a garbled truth table for each gate in the circuit.
Yao’s Protocol for Two-Party Computation

Step 2: Garbler “encrypts” the circuit (i.e., “garbles” the circuit)

Garbler constructs a garbled truth table for each gate in the circuit

Key for party 1’s input
Enc \( k_0^{(1)} \)

Key for party 2’s input
Enc \( k_0^{(2)}, k_0^{(\text{out})} \)

Key for output wire

Inputs
<table>
<thead>
<tr>
<th>Party 1</th>
<th>Party 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Output
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Garbler constructs a garbled truth table for each gate in the circuit

[Yao82]
Step 2: Garbler “encrypts” the circuit (i.e., “garbles” the circuit)

Garbler constructs a garbled truth table for each gate in the circuit:
Step 2: Garbler “encrypts” the circuit (i.e., “garbles” the circuit)

Garbler constructs a garbled truth table for each gate in the circuit.

Garbled truth table randomly permuted

\[
\begin{align*}
\text{Enc} \left( k_0^{(1)}, \text{Enc} \left( k_0^{(2)}, k_0^{(out)} \right) \right) \\
\text{Enc} \left( k_0^{(1)}, \text{Enc} \left( k_1^{(2)}, k_0^{(out)} \right) \right) \\
\text{Enc} \left( k_1^{(1)}, \text{Enc} \left( k_0^{(2)}, k_0^{(out)} \right) \right) \\
\text{Enc} \left( k_1^{(1)}, \text{Enc} \left( k_1^{(2)}, k_1^{(out)} \right) \right)
\end{align*}
\]
Yao’s Protocol for Two-Party Computation

**Step 2:** Garbler “encrypts” the circuit (i.e., “garbles” the circuit)

Garbler constructs a garbled truth table for each gate in the circuit

**Garbled truth table randomly permuted**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0</td>
<td>Enc (k_1^{(1)}, Enc \left(k_0^{(2)}, k_0^{(\text{out})}\right))</td>
</tr>
<tr>
<td>1 1 1</td>
<td>Enc (k_1^{(1)}, Enc \left(k_1^{(2)}, k_1^{(\text{out})}\right))</td>
</tr>
<tr>
<td>0 1 0</td>
<td>Enc (k_0^{(1)}, Enc \left(k_1^{(2)}, k_0^{(\text{out})}\right))</td>
</tr>
<tr>
<td>0 0 0</td>
<td>Enc (k_0^{(1)}, Enc \left(k_0^{(2)}, k_0^{(\text{out})}\right))</td>
</tr>
</tbody>
</table>

**Invariant:** Given just a single key for each input wire, evaluator can learn a single key for the output wire

Garbler constructs a garbled truth table for each gate in the circuit

[Yao82]
Step 2: Garbler “encrypts” the circuit (i.e., “garbles” the circuit)

Garbled truth table randomly permuted

Invariant: Given just a single key for each input wire, evaluator can learn a single key for the output wire
Step 2: Garbler “encrypts” the circuit (i.e., “garbles” the circuit)

\[
\text{Enc}(k_1^{(1)}, \text{Enc}(k_0^{(2)}, k_0^{(\text{out})}))
\]

Garbled truth table randomly permuted

**Invariant:** Given just a single key for each input wire, evaluator can learn a single key for the output wire

\[k_0^{(\text{out})}\] is just a symmetric key – does not reveal what the output bit is

\[k_1^{(1)} \quad k_0^{(2)}\]
Yao’s Protocol for Two-Party Computation

**Step 2:** Garbler “encrypts” the circuit (i.e., “garbles” the circuit)

Garbled truth table randomly permuted

Invariant: Given just a single key for each input wire, evaluator can learn a single key for the output wire

Cannot decrypt other output keys
**Yao’s Protocol for Two-Party Computation**

**Step 2:** Garbler “encrypts” the circuit (i.e., “garbles” the circuit)

**Invariant:** Given just a single key for each input wire and a garbled table, evaluator can learn a single key for the output wire.
Step 2: Garbler “encrypts” the circuit (i.e., “garbles” the circuit)

Invariant: Given just a single key for each input wire and a garbled table, evaluator can learn a single key for the output wire
Yao’s Protocol for Two-Party Computation

Step 2: Garbler “encrypts” the circuit (i.e., “garbles” the circuit)

Invariant: Given just a single key for each input wire and a garbled table, evaluator can learn a single key for the output wire.

Include decoding table to map output keys to output values.
Yao’s Protocol for Two-Party Computation

**Step 2:** Garbler “encrypts” the circuit (i.e., “garbles” the circuit)

Garbler can send garbled truth tables and keys for its inputs for Party 1’s inputs.

**Question:** how does evaluator obtain keys for its input?

Garbler can send garbled truth tables and keys for its inputs.
Step 3: Evaluator uses “oblivious transfer” to obtain keys for its input

For each wire corresponding to evaluator’s input, the garbler has two keys

For each input wire, evaluator wants to obtain key corresponding to its input value

At the end of the oblivious transfer protocol, garbler learns nothing about which key evaluator obtains, and evaluator learns exactly one of the two keys

Yao’s Protocol for Two-Party Computation

[Yao82]
Yao’s Protocol for Two-Party Computation

Two-round protocol for secure two-party communication

- OT message for keys corresponding to input wires
- Keys communicated using OT (garbler does not know which keys are transmitted)
- Garbler
- Evaluator
- Many improvements are possible to achieve better performance
- Evaluator uses keys to evaluate circuit gate-by-gate

[Yao82]
Yao’s Protocol for Two-Party Computation

Two-round protocol for secure two-party communication

Keys communicated using OT (garbler does not know which keys are transmitted)

Protocol is very efficient; communication is the bottleneck

Many improvements are possible to achieve better performance

OT message for keys corresponding to input wires

[Yoos82]
The Story So Far...

General techniques apply to many different scenarios for diagnosing Mendelian diseases

Simple frequency-based filters are useful for rare disease diagnosis and can be efficiently evaluated in a privacy-preserving manner
But What About More Complex Diseases?

Genome-wide association studies (GWAS):
- Identify genetic variants most correlated with a particular disease (or particular phenotype)
- Oftentimes, focused on identifying complex interactions between many variants

Control group (healthy)

Case group (affected)
But What About More Complex Diseases?

Each patient has a vector of SNPs (variations in specific locations in genome – 3 types)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>2</th>
<th>2</th>
<th>...</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Healthy</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>Patients</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>Patients</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>Patients</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td>1</td>
</tr>
</tbody>
</table>

Disease status

- Healthy: 0
- Patients with lung cancer: 1

Goal: identify SNPs that are most correlated with disease status
But What About More Complex Diseases?

Each patient has a vector of SNPs (variations in specific locations in genome – 3 types)

<table>
<thead>
<tr>
<th>Healthy individuals</th>
<th>Patients with lung cancer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 2 2 ... 1</td>
<td>0 1 1 0 2 2 ... 1</td>
</tr>
<tr>
<td>0 1 1 0 2 2 ... 1</td>
<td>0 1 1 0 2 2 ... 1</td>
</tr>
<tr>
<td>1 1 0 0 2 2 ... 0</td>
<td>0 1 1 0 2 2 ... 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Disease status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Goal: identify SNPs that are most correlated with disease status

Healthy individuals

Patients with lung cancer

Unlike Mendelian diseases, we are looking for many associations (e.g., several hundred)
But What About More Complex Diseases?

≈ 500,000 SNPs

Healthy individuals

Patients with lung cancer

≈ 25,000 individuals

Disease status

[CWB18]
But What About More Complex Diseases?

\[ \approx 500,000 \text{ SNPs} \]

\[ \approx 25,000 \text{ individuals} \]

**Challenge:** in real GWAS studies, we need to correct for population-level differences between groups.
GWAS computations most naturally expressed as arithmetic computations (e.g., matrix operations)

Recall: to apply Yao’s protocol, must first represent computation as a Boolean circuit

Can introduce significant overhead for arithmetic computations!
Arithmetic Computations on Shared Data

Patients “secret share” their data with two non-colluding hospitals

Approach: directly compute on secret-shared data
Arithmetic Computations on Shared Data

All operations done over a ring ($\mathbb{Z}_p$)

$[v_1]_1 + [v_1]_2 = v_1$
$[v_2]_1 + [v_2]_2 = v_2$
Arithmetic Computations on Shared Data

Observation: each party can locally compute on their shares to obtain a share of the sum.

\[
\begin{align*}
[r_1]_1 & = \begin{bmatrix} 3 \\ -1 \\ \vdots \\ 7 \end{bmatrix} \\
[r_2]_1 & = \begin{bmatrix} -2 \\ 2 \\ \vdots \\ 0 \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
[v_1]_1 & = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 7 \end{bmatrix} \\
[v_2]_1 & = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 0 \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
[v_1]_2 & = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \\
[v_2]_2 & = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
[v_1]_1 + [v_2]_1 & = [v_1 + v_2]_1 \\
[v_1]_2 + [v_2]_2 & = [v_1 + v_2]_2
\end{align*}
\]
Arithmetic Computations on Shared Data

Observation: each party can locally compute on their shares to obtain a share of the sum.

For computing products on shared values (e.g., matrix-vector products, inner products, etc.), we can use a single-round interactive protocol [Bea91].
What About More Complex Diseases?

This work: first end-to-end GWAS protocol (with population correction)

- Based on computing on secret-shared inputs
- For 25K individuals, computation completes in about 3 days: feasible for performing large-scale scientific studies

Approach: directly compute on secret-shared data

Can compose with differential privacy to ensure outputs preserve privacy of user data
Modern cryptographic tools enable useful computations while protecting the privacy of individual genomes.
Privacy-Preserving Machine Learning

**Private training:** Multiple parties train a joint model on their aggregate data while ensuring privacy of the input data.

**Private inference:** Client learns model’s output, server does not learn anything.

*Machine learning as a service*
Privacy-Preserving Machine Learning

Private training: Multiple parties train a joint model on their aggregate data while ensuring privacy of the input data.

Private inference: Client learns model's output, server does not learn anything.

Machine learning as a service: “Set an alarm for 3 PM”

Many constructions in recent years:
- MiniONN [MJLA17], EzPC [CGRST17], SecureML [MZ17], ABY³ [MR18], Chameleon [RWTS+S18], SecureNN [WGC19], XONN [RSCLLL+19], ASTRA [CCPS19], BLAZE [PS20], Delphi [MSZP20], FLASH [BCPS20], Trident [CRS20], CrypTFlow [KRCR+C20], Falcon [WTBK+21]
Privacy-Preserving Machine Learning

Simple models and datasets:

Feed-forward neural networks
10,000 – 100,000 parameters
2 – 3 layers

MNIST dataset
10 classes; 60,000 examples

Many constructions in recent years:
MiniONN [MJLA17], EzPC [CGRST17], SecureML [MZ17], ABY3 [MR18], Chameleon [RWTS*18], SecureNN [WGC19], XONN [RSCLL*19], ASTRA [CCPS19], BLAZE [PS20], Delphi [MLSZP20], FLASH [BCPS20], Trident [CRS20], CrypTFlow [KRCDR+20], Falcon [WTBKMK+21]
Privacy-Preserving Machine Learning

Larger models and datasets:

AlexNet [KSH12]
- 61,000,000 parameters
- 8 layers

Tiny ImageNet [LKJ17]
- Subset of ImageNet [RDSKS+15]
- 200 classes, 100,000 examples

Many constructions in recent years:

- MiniONN [MJLA17], EzPC [CGRST17], SecureML [MZ17], ABY³ [MR18], Chameleon [RWTS+18],
- SecureNN [WGC19], XONN [RSC+19], ASTRA [CCPS19], BLAZE [PS20], Delphi [MLSZP20],
- FLASH [BCPS20], Trident [CRS20], CrypTFlow [KCRD+20], Falcon [WTBKM+21]
The Scalability Challenge in Private ML

Tiny ImageNet [LKJ17]
Subset of ImageNet [RDSKS+15]
200 classes, 100,000 examples

Training the AlexNet model on Tiny ImageNet

Cost of privacy: 960×

[Assuming 90 epochs over dataset]
Modern Deep Learning (without Privacy)

Pre-2012: Training primarily on CPU

Neural machine translation
VGG
ResNets
AlphaZero

Deep belief networks
AlexNet

LeNet

RNN (for speech)

NETtalk

Perceptron

Moore’s Law Scaling

Figure adapted from OpenAI blog post
Modern Deep Learning (without Privacy)

- **Pre-2012:** Training primarily on CPU
- **2012-2014:** Small number (1-8) of GPUs
- **2014-2016:** Large number (10-100) of GPUs
- **2016-today:** Custom hardware (e.g., TPUs)

Figure adapted from OpenAI blog post.
CryptGPU: Private ML on the GPU

First cryptographic framework where all computations performed on the GPU

Supporting Cryptography on the GPU:
- Cryptographic protocols designed for general-purpose computing architectures
- CUDA kernels for linear algebra operate on floating-point types while cryptographic protocols operate on discrete data types (e.g., finite fields)

This Work:
- New abstractions and protocols to embed cryptographic operations onto the GPU
CryptGPU: Private ML on the GPU

First cryptographic framework where all computations performed on the GPU

Supporting Cryptography on the GPU:
- Cryptographic protocols designed for general-purpose computing architectures
- CUDA kernels for linear algebra operate on floating-point types while cryptographic protocols operate on discrete data types (e.g., finite fields)

This Work:
- New abstractions and protocols to embed cryptographic operations onto the GPU
- New protocols to take better advantage of GPU acceleration

Motivates the study of “GPU-friendly” cryptography
CryptGPU: Private ML on the GPU

For private training, largest dataset to date is Tiny ImageNet [LKJ17]

Subset of ImageNet [RDSKS+15] 200 classes, 100,000 examples

Training the AlexNet model on Tiny ImageNet

- **No Privacy**: 8.5 hours | 11 minutes
- **With Privacy**: 338 days | 9 days

**44× speedup**

**37× speedup**

Cost of privacy:
- 26× (CPU)
- 1100× (GPU)

[Assuming 90 epochs over dataset]
Towards an AlexNet Moment for Private ML

Feasibility result: Possible to run cryptographic protocols entirely on the GPU

More improvements possible if we tailor protocol design to GPU architecture
- Specialized CUDA kernels for cryptographic operations
- New embeddings for discrete cryptographic structures

Can we take advantage of even more specialized hardware (FPGAs, TPUs, etc.)?

Training the AlexNet model on Tiny ImageNet

No Privacy
- Training Time: 8.5 hours
- Cost of privacy: 26× (CPU)

With Privacy
- Training Time: 11 minutes
- Cost of privacy: 1100× (GPU)

37× speedup

338 days
9 days

44× speedup

10^6
10^5
10^4
10^3
10^2
10^1
10^0
Training Time (minutes)

CPU
GPU

Cost of privacy:
Computing on Private Data

Modern cryptographic tools enable computations on private data

Cryptographic tools: hide the input $x_i$ from the computing party

Complementary goals as differential privacy (and oftentimes, can be combined)

Aggregate statistic $y \leftarrow f(x_1, \ldots, x_5)$

Thank you!