Microsoft Research

Each year Microsoft Research hosts hundreds of influential speakers from around the world including leading scientists, renowned experts in technology, book authors, and leading academics, and makes videos of these lectures freely available.

2016 © Microsoft Corporation. All rights reserved.
Multiparty Computation Research
@ TU Eindhoven

Niek J. Bouman
(joint work with Frank Blom, Berry Schoenmakers
and Niels de Vreede)

Oct 1st 2018, Microsoft Research seminar
Plan

- MPyC – Python framework for MPC
- Secure Linear Algebra over $\mathbb{Q}$
- Secure Comparison of Medium-Range Integers
Secure Multi-Party Computation – Motivating Example

United States Census Bureau

HOSPITAL

HOSPITAL
Preliminaries: Secure Multi-Party Computation

- Suppose: 3 parties \((A, B, C)\), each holding private integer \(x_A, x_B, x_C\) (e.g., age, wealth)
- Parties want to compute \(f(x_A, x_B, x_C)\), such that:
  no party learns more than what can be deduced from her input and \(f(x_A, x_B, x_C)\)
Preliminaries: Secure Multi-Party Computation

- Suppose: 3 parties \((A, B, C)\), each holding private integer \(x_A, x_B, x_C\) (e.g., age, wealth)
- Parties want to compute \(f(x_A, x_B, x_C)\), such that:
  no party learns more than what can be deduced from her input and \(f(x_A, x_B, x_C)\)
- Example:

  \[
  f_1(x_A, x_B, x_C) := x_A + x_B + x_C
  \]

  \[
  f_2(x_A, x_B, x_C) := (x_A + x_B) \times (x_B + x_C)
  \]

  \[
  f_3(x_A, x_B, x_C) := [x_A < (x_B + x_C)] \in \{0, 1\}
  \]
Preliminaries: Secure Multi-Party Computation

- Suppose: 3 parties \((A, B, C)\), each holding private integer \(x_A, x_B, x_C\) (e.g., age, wealth)
- Parties want to compute \(f(x_A, x_B, x_C)\), such that:
  - no party learns more than what can be deduced from her input and \(f(x_A, x_B, x_C)\)
- Example:
  
  \[
  f_1(x_A, x_B, x_C) := x_A + x_B + x_C
  \]
  
  \[
  f_2(x_A, x_B, x_C) := (x_A + x_B) \times (x_B + x_C)
  \]
  
  \[
  f_3(x_A, x_B, x_C) := [x_A < (x_B + x_C)] \in \{0, 1\}
  \]

How to compute this?
MPC: Two flavours

- Garbled circuits:
  - Functionality represented as a Boolean circuit
  - Two-party (but extensions exist to multiple parties)
- Arithmetic Secret Sharing (*this talk*)
  - Functionality represented as an arithmetic circuit
  - Naturally extends to multiple parties
  - Operations over an arbitrary finite field (or finite ring)
Preliminaries – *How to Share a Secret: Shamir’s Secret Sharing Scheme* [Shamir, 1979]

- secret: $s \in \mathbb{F}_p$, with $p$ prime, and such that $p > n$
- $n$ players,
- Let $t \in \mathbb{N}$ s.t. $t < n/2$ (passive security)
- Let $p(x) \in \mathbb{F}_p[x]$ be random polynomial, $\deg p(x) \leq t$
  s.t. $f(0) = s$
Preliminaries – *How to Share a Secret: Shamir’s Secret Sharing Scheme* [Shamir, 1979]

- secret: $s \in \mathbb{F}_p$, with $p$ prime, and such that $p > n$
- $n$ players,
- Let $t \in \mathbb{N}$ s.t. $t < n/2$ (passive security)
- Let $p(x) \in \mathbb{F}_p[x]$ be random polynomial, $\deg p(x) \leq t$
  s.t. $f(0) = s$

![Diagram](image)
Preliminaries – *How to Share a Secret: Shamir’s Secret Sharing Scheme* [Shamir, 1979]

- secret: \( s \in \mathbb{F}_p \), with \( p \) prime, and such that \( p > n \)

- \( n \) players,

- Let \( t \in \mathbb{N} \) s.t. \( t < n/2 \) (passive security)

- Let \( p(x) \in \mathbb{F}_p[x] \) be random polynomial, \( \deg p(x) \leq t \) s.t. \( f(0) = s \)

- \( t + 1 \) evaluation points needed to reconstruct \( p(x) \) (and \( s \))

- \( t \) evaluation points give no information about \( s \)
Preliminaries: Secret Sharing

- Notation: \([a]\)
- Adding secret-shared values: \([a] + [b]\)
  Local operation
Preliminaries: Secret Sharing

- Notation: \([a]\)
- Adding secret-shared values: \([a] + [b]\)
  Local operation
- Multiplying secret-shared values: \([a] \cdot [b]\)
  More “expensive”, requires communication between players
History of MPC Frameworks

- Fairplay (Malkhi et al., 2004)
- VIFF: Virtual Ideal Functionality Framework (Geisler, et al., 2007)
- Sharemind (Bogdanov et al., 2008)
- Bristol-SPDZ (Keller et al., 2013)
- ... many more
History of MPC Frameworks

- Fairplay (Malkhi et al., 2004)
- VIFF: Virtual Ideal Functionality Framework (Geisler, et al., 2007)
- Sharemind (Bogdanov et al., 2008)
- Bristol-SPDZ (Keller et al., 2013)
- ...many more

This talk: **MPyC**, Python framework for MPC, inspired by VIFF
Classifying MPC Frameworks

- Library-based: MPC functions are in a library (VIFF, MPyC, ...)
  - Flexible to mix MPC computations and other computations (e.g. use functions from NTL)
- Domain specific language (Secre-C, Bristol SPDZ, ...)
  - Might be better suited for verification of correctness
MPyC and its Execution Model

- Computation with asynchronous tasks and futures
MPyC and its Execution Model

- Computation with *asynchronous tasks* and *futures*
- Given $a, b, d$, suppose you want to compute $e := abd$, i.e.,
  
  $$
  c = a \times b \\
  e = c \times d 
  $$

- multiplications involve interaction with other players (resharing)
MPyC and its Execution Model

- Computation with *asynchronous tasks and futures*
- Given $a, b, d$, suppose you want to compute $e := abd$, i.e.,

\[
\begin{align*}
  c &= a \times b \\
  e &= c \times d
\end{align*}
\]

- Multiplications involve interaction with other players (resharing)
- In a *synchronous* programming environment, you get:
  1. compute $c = a \times b$ (blocks until resharing is complete)
  2. compute $e = c \times d$ (blocks until resharing is complete)
MPyC and its Execution Model

- Computation with *asynchronous tasks and futures*
  - Given $a$, $b$, $d$, suppose you want to compute $e := abd$, i.e.,
    
    $$c = a \ast b$$
    $$e = c \ast d$$
  
- Multiplications involve interaction with other players (resharing)
  - In a *synchronous* programming environment, you get:
    1. compute $c = a \ast b$ (blocks until resharing is complete)
    2. compute $e = c \ast d$ (blocks until resharing is complete)
  
- With *asynchronous tasks*, you can do
  1. $\text{Future}(c) = \text{Task}(a \ast b)$ (returns immediately, no waiting)
MPyC and its Execution Model

- Computation with *asynchronous tasks* and *futures*

- Given $a, b, d$, suppose you want to compute $e := abd$, i.e.,
  
  $c = a \times b$
  
  $e = c \times d$

- Multiplications involve interaction with other players (resharing)

- In a *synchronous* programming environment, you get:
  
  1. compute $c = a \times b$ (blocks until resharing is complete)
  2. compute $e = c \times d$ (blocks until resharing is complete)

- With *asynchronous tasks*, you can do
  
  1. $\text{Future}(c) = \text{Task}(a \times b)$ (returns immediately, no waiting)
  2. $\text{Future}(e) = \text{Task}(\text{Future}(c) \times d)$
     
     (returns immediately, no waiting)
  3. $e = \text{await}(\text{Future}(e))$ (blocks until $e$ is “ready”)
MPyC and its Execution Model

- Computation with asynchronous tasks and futures
  - Given $a, b, d$, suppose you want to compute $e := abd$, i.e.,
    \[
    c = a \times b  \\
    e = c \times d
    \]

- Multiplications involve interaction with other players (resharing)
- In a synchronous programming environment, you get:
  1. compute $c = a \times b$ (blocks until resharing is complete)
  2. compute $e = c \times d$ (blocks until resharing is complete)

- With asynchronous tasks, you can do
  1. $\text{Future}(c) = \text{Task}(a \times b)$ (returns immediately, no waiting)
  2. $\text{Future}(e) = \text{Task}(\text{Future}(c) \times d)$ (returns immediately, no waiting)
  3. $e = \text{await}(\text{Future}(e))$ (blocks until $e$ is “ready”)

- Coroutines
  1. execution paradigm to multiplex several “blocking” tasks on a single thread
Computation with Async. Tasks + Futures

- Hybrid form of greedy and lazy execution
- Benefit: look ahead to see what kind of work is coming
Computation with Async. Tasks + Futures

- Hybrid form of greedy and lazy execution
- Benefit: look ahead to see what kind of work is coming
- If this is local work (no interaction), we can already execute it
Computation with Async. Tasks + Futures

- Hybrid form of greedy and lazy execution
- Benefit: look ahead to see what kind of work is coming
- If this is local work (no interaction), we can already execute it
- Hence, main purpose of MPyC’s execution model is to let the CPU(s) do useful work while waiting for data from other players
Computation with Async. Tasks + Futures

- Hybrid form of greedy and lazy execution
- Benefit: look ahead to see what kind of work is coming
- If this is local work (no interaction), we can already execute it
- Hence, main purpose of MPyC’s execution model is to let the CPU(s) do useful work while waiting for data from other players
- Requires some throttling mechanism to limit the amount of outstanding work
MPyC

- Open source, Python3 based
- Currently: passively secure MPC based on Shamir secret-sharing
- Suitable for rapid prototyping / teaching
- Jupyter notebook support, with single-party execution mode (simplifies development)
- ...Demo!
Secure Linear Algebra over the Rationals

Frank Blom, Niek J. Bouman, Berry Schoenmakers, Niels de Vreede

TU/e
Technische Universiteit Eindhoven
University of Technology

Monday, Oct 1, 2018

Funded by EU H2020 SODA
Setting

- Secret-sharing based MPC
- Multi-party ($N_{\text{players}} \geq 3$) scenario
Setting

- Secret-sharing based MPC
- Multi-party ($N_{\text{players}} \geq 3$) scenario
- Protocols on top of abstract MPC “arithmetic black box”
Problem Sketch

Consider a **full-rank** square matrix $A$ and vector $b$ with integral entries, **secret-shared** among the players.

**Task**

- Compute vector $x$ such that $Ax = b$,
- where $x$ is the rational solution (over $\mathbb{Q}$)
Solving Full-Rank System over $\mathbb{Q}$ (in MPC)

Motivation

Useful for privacy-preserving data processing / statistics / etc
Related Work: Secure Linear Algebra over $\mathbb{Q}$

Multi-party case
[Toft, 2009]

2-party case
Several results in the 2-party setting, like
[Nikolaenko et al., 2013, Gascón et al., 2017, Joye, 2017, Giacomelli et al., 2017]
Nonetheless, we do not target the 2-party scenario in this work.
Integer vs. Rational Arithmetic in MPC

Integer arithmetic

- One-to-one correspondence between field elements and integers in \([-\lfloor p/2 \rfloor, \lfloor p/2 \rfloor]\)
- Prevent "wrapping around the modulus"

Rational arithmetic
Integer vs. Rational Arithmetic in MPC

Integer arithmetic

- One-to-one correspondence between field elements and integers in $[-|p/2|, |p/2|]$.
- Prevent “wrapping around the modulus”.

Rational arithmetic

- Division $a/b$ gives field element $x = a \cdot b^{-1}$.
- As long as $|a|, |b| \leq \sqrt{p}/2$, we can uniquely reconstruct $a$ and $b$ from $x$ using **Rational Reconstruction** [Wang, 1981].
Integer vs. Rational Arithmetic in MPC

Integer arithmetic

- One-to-one correspondence between field elements and integers in $[-|p/2|, |p/2|]$.
- Prevent “wrapping around the modulus”

Rational arithmetic

- Division $a/b$ gives field element $x = a \cdot b^{-1}$.
- As long as $|a|, |b| \leq \sqrt{p}/2$, we can uniquely reconstruct $a$ and $b$ from $x$ using **Rational Reconstruction** [Wang, 1981].
- Reduce the lattice basis $\{(p, 0), (x, 1)\}$ (Lagrange–Gauss).
- Reduced basis will contain the vector $(a, b)$. 
Integer vs. Rational Arithmetic in MPC

Integer arithmetic

- One-to-one correspondence between field elements and integers in $[-|p/2|, |p/2|]$
- Prevent “wrapping around the modulus”

Rational arithmetic

- Division $a/b$ gives field element $x = a \cdot b^{-1}$
- As long as $|a|, |b| \leq \sqrt{p}/2$, we can uniquely reconstruct $a$ and $b$ from $x$ using **Rational Reconstruction** [Wang, 1981]
- Reduce the lattice basis $\{((p, 0), (x, 1))\}$ (Lagrange–Gauss),
- Reduced basis will contain the vector $(a, b)$.
- Problem: performing these steps obliviously would be **impractical**: $O(\log p)$ iterations [Vallée, 1991], with expensive secure integer division in each round
Solving $Ax = b$ over $\mathbb{Q}$ (A full rank)

- Let $A \in \mathbb{Z}^{n\times n}$
- Then, in general, $A^{-1} \in \mathbb{Q}^{n\times n}$. 

Solving $Ax = b$ over $\mathbb{Q}$ ($A$ full rank)

- Let $A \in \mathbb{Z}^{n \times n}$
- Then, in general, $A^{-1} \in \mathbb{Q}^{n \times n}$.
- Inverse of $A$ can be written as follows:

$$A^{-1} = \frac{\text{adj } A}{\det A}$$

where adj $A$ is the adjugate of $A$
Solving $Ax = b$ over $\mathbb{Q}$ ($A$ full rank)

- Let $A \in \mathbb{Z}^{n \times n}$
- Then, in general, $A^{-1} \in \mathbb{Q}^{n \times n}$.
- Inverse of $A$ can be written as follows:

$$A^{-1} = \frac{\text{adj } A}{\det A}$$

where adj $A$ is the adjugate of $A$

- adj $A$ has integral entries
Solving $Ax = b$ over $\mathbb{Q}$ ($A$ full rank)

- Let $A \in \mathbb{Z}^{n \times n}$
- Then, in general, $A^{-1} \in \mathbb{Q}^{n \times n}$.
- Inverse of $A$ can be written as follows:

$$A^{-1} = \frac{\text{adj } A}{\det A}$$

where $\text{adj } A$ is the adjugate of $A$

- $\text{adj } A$ has integral entries
- Solution $x$ of the system $Ax = b$ can be represented as

$$(\text{adj}(A)b, \det(A)) \in \mathbb{Z}^{n} \times \mathbb{Z}$$

- Representation avoids occurrence of rational entries
Our Solution ($Ax = b$ over $\mathbb{Q}$, $A$ full rank)

- We work over the finite field $\mathbb{F}_p := \mathbb{Z}/p\mathbb{Z}$, $p$ prime
- A modification of protocol of [Cramer and Damgård, 2001]
  (which is based on [Bar-Ilan and Beaver, 1989])
- Modification: keep adjugate and determinant separate
Our Solution ($Ax = b$ over $\mathbb{Q}$, $A$ full rank)

- We work over the finite field $\mathbb{F}_p := \mathbb{Z}/p\mathbb{Z}$, $p$ prime
- A modification of protocol of [Cramer and Damgård, 2001] (which is based on [Bar-Ilan and Beaver, 1989])
- Modification: keep adjugate and determinant separate
- $p$ must be large enough to represent $\det A$ and entries of $\text{adj}(A)b$
Our Solution ($Ax = b$ over $\mathbb{Q}$, $A$ full rank)

- We work over the finite field $\mathbb{F}_p := \mathbb{Z}/p\mathbb{Z}$, $p$ prime
- A modification of protocol of [Cramer and Damgård, 2001] (which is based on [Bar-Ilan and Beaver, 1989])
- Modification: keep adjugate and determinant separate
- $p$ must be large enough to represent $\det A$ and entries of $\text{adj}(A)b$
- Bound on $p$ follows essentially from Hadamard’s inequality:

**Lemma (Hadamard)**

*For any matrix $M \in [-B, B]^{n \times n}$

\[
|\det M| \leq B^n n^{n/2}.
\]
Computing \((\text{adj } A, \det A)\) via Random Self-Reduction

1. Let \([A]\) be Shamir-secret-shared over the field \(\mathbb{F}_p\).
Computing $\text{adj } A$, $\text{det } A$ via Random Self-Reduction

1. Let $[A]$ be Shamir-secret-shared over the field $\mathbb{F}_p$.

2. Sample lower triangular matrix $[L] \in \mathbb{F}_p^{n \times n}$ having ones on its diagonal uniformly at random.

3. Sample upper triangular matrix $[U] \in \mathbb{F}_p^{n \times n}$ uniformly at random such that diagonal does not contain zeros.
Computing \((\text{adj } A, \det A)\) via Random Self-Reduction

1. Let \([A]\) be Shamir-secret-shared over the field \(\mathbb{F}_p\).
2. Sample lower triangular matrix \([L] \in \mathbb{F}_p^{n \times n}\)
   having ones on its diagonal uniformly at random.
3. Sample upper triangular matrix \([U] \in \mathbb{F}_p^{n \times n}\) uniformly at random such that diagonal does not contain zeros.
4. Compute \([R] := [LU]\) and \([d] := [(\det R)] = (\prod_i \text{diag}(U)_i)\).
Computing \((\text{adj } A, \det A)\) via Random Self-Reduction

1. Let \([A]\) be Shamir-secret-shared over the field \(\mathbb{F}_p\).

2. Sample lower triangular matrix \([L] \in \mathbb{F}_p^{n \times n}\) having ones on its diagonal uniformly at random.

3. Sample upper triangular matrix \([U] \in \mathbb{F}_p^{n \times n}\) uniformly at random such that diagonal does not contain zeros.

4. Compute \([R] := [LU]\) and \([d] := [(\det R)] = (\prod_i \text{diag}(U)_i)\).

5. Compute \([RA]\) and reveal it.

6. In the clear, compute \(\text{adj } RA\) and \(\det RA\).
Computing $(\text{adj } A, \det A)$ via Random Self-Reduction

1. Let $[A]$ be Shamir-secret-shared over the field $\mathbb{F}_p$.

2. Sample lower triangular matrix $[L] \in \mathbb{F}_p^{n \times n}$ having ones on its diagonal uniformly at random.

3. Sample upper triangular matrix $[U] \in \mathbb{F}_p^{n \times n}$ uniformly at random such that diagonal does not contain zeros.

4. Compute $[R] := [LU]$ and $[d] := [(\det R)] = (\prod_i \text{diag}(U)_i)$

5. Compute $[RA]$ and reveal it.

6. In the clear, compute $\text{adj } RA$ and $\det RA$.

7. Compute $[\text{adj } A] := \text{adj}(RA)[R][d^{-1}]$, $[\det A] := \det(RA)[d^{-1}]$
Computing $(\text{adj } A, \det A)$ via Random Self-Reduction

1. Let $[A]$ be Shamir-secret-shared over the field $\mathbb{F}_p$.

2. Sample lower triangular matrix $[L] \in \mathbb{F}_p^{n \times n}$ having ones on its diagonal uniformly at random.

3. Sample upper triangular matrix $[U] \in \mathbb{F}_p^{n \times n}$ uniformly at random such that diagonal does not contain zeros.


5. Compute $[RA]$ and reveal it.

6. In the clear, compute $\text{adj } RA$ and $\det RA$.

7. Compute $[\text{adj } A] := \text{adj}(RA)[R][d^{-1}]$, $[\det A] := \det(RA)[d^{-1}]$.

$L$ is uni-triangular: simplifies proof in [Cramer and Damgård, 2001] (and slightly fewer multiplications & saves 1 communication round).
Solving $Ax = b$ securely over $\mathbb{Q}$, where $A$ is square ($n$ by $n$) and full rank.

<table>
<thead>
<tr>
<th>Our work</th>
<th># Rounds</th>
<th># Secure Mults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Self-Reducibility</td>
<td>$O(1)$</td>
<td>$O(n^2)$*</td>
</tr>
</tbody>
</table>

* Assuming “cheap” inner products (Shamir LSS)
Fast Secure Comparison for Medium-Sized Integers and Its Application in Binarized Neural Networks

Mark Abspoel, Niek J. Bouman, Berry Schoenmakers, Niels de Vreede

TU/e
ingenieurschool Eindhoven
University of Technology

Monday, Oct 1, 2018

Funded by EU H2020 SODA
Secure Comparison in MPC

\[ [x] := [a] - [b] \]

- Compute \( \text{sgn}(x) \) or \( \text{GEZ}(x) \)

\[
\text{sgn}(z) := \begin{cases} 
1 & \text{if } z > 0, \\
0 & \text{if } z = 0, \\
-1 & \text{if } z < 0.
\end{cases}
\]

\[
\text{GEZ}(z) := \begin{cases} 
1 & \text{if } z \geq 0, \\
-1 & \text{if } z < 0.
\end{cases}
\]
Computing the Sign via Quadratic Residuosity

For certain primes $p$ of bit-length $l$, the Legendre symbol $\left( \frac{\alpha}{p} \right)$ coincides with the sign of $x$ for $x \in [-d, d]$ where $d = O(l)$.

- Idea goes back to [Feige et al., 1994]
- also used by [Yu, 2011]
Computing the Sign via Quadratic Residuosity

For certain primes $p$ of bit-length $\ell$, the Legendre symbol $\left( \frac{x}{p} \right)$ coincides with the sign of $x$ for $x \in [-d, d]$ where $d = O(\ell)$.

- Idea goes back to [Feige et al., 1994]
- also used by [Yu, 2011]

Example: $p = 311$, $\ell = 9$, $d = 10$ \hspace{1cm} (\text{$p$ is a Blum prime})

\[
\begin{array}{cccccccccccc}
  x & -11 & -10 & -9 & -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 \\
  \left( \frac{x}{p} \right) & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
\end{array}
\]

\[
\begin{array}{cccccccccccccccc}
  x & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \ldots \\
  \left( \frac{x}{p} \right) & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 \\
\end{array}
\]
Securely Computing the Legendre symbol

Legendre symbol is completely multiplicative: \( \left( \frac{a}{p} \right) \left( \frac{b}{p} \right) = \left( \frac{ab}{p} \right) \), \( a, b \in \mathbb{Z} \).
Securely Computing the Legendre symbol

Legendre symbol is completely multiplicative: \( \left( \frac{a}{p} \right) \left( \frac{b}{p} \right) = \left( \frac{ab}{p} \right) \), \( a, b \in \mathbb{Z} \).

Protocol Legendre[\(x\)]

**Offline Phase**

1: \([a] \leftarrow \text{RandomElem}(\mathbb{F}_p^*)\)
2: \([b] \leftarrow \text{RandomBit}()\)
3: \([s] \leftarrow 2[b] - 1\)
4: \([r] \leftarrow [s] \cdot [a^2]\)
5: **return** \(([r], [s])\)

**Online Phase (only 1 communication round)**

6: \(c \leftarrow [x] \cdot [r]\)
7: \([z] \leftarrow \left( \frac{c}{p} \right) \cdot [s]\)
8: **return** \([z]\)
Securely Computing the Legendre symbol

Legendre symbol is completely multiplicative: \( \left( \frac{a}{p} \right) \left( \frac{b}{p} \right) = \left( \frac{ab}{p} \right) \), \( a, b \in \mathbb{Z} \).

Protocol Legendre\([x]\)

**Offline Phase**

1: \([a] \leftarrow \text{RandomElem}(\mathbb{F}_p^*)\)
2: \([b] \leftarrow \text{RandomBit}()\)
3: \([s] \leftarrow 2[b] - 1\)
4: \([r] \leftarrow [s] \cdot [a^2]\)
5: \text{return} \( ([r], [s]) \)

**Online Phase** (only 1 communication round)

6: \( c \leftarrow [x] \cdot [r] \)
7: \([z] \leftarrow \left( \frac{c}{p} \right) \cdot [s]\)
8: \text{return} \([z]\)

Cannot securely evaluate \( \left( \frac{0}{p} \right) \)!! (But we ignore this in this talk)
How to Increase the Range?

- We want to find sign of \( x \in [-d, d] \); we call \( d \) the range.
- How to increase the range?
How to Increase the Range?

- We want to find sign of $x \in [-d, d]$; we call $d$ the range.
- How to increase the range?
  - Increase $\ell$ and search for an $\ell$-bit prime with “good” range.
How to Increase the Range?

- We want to find sign of $x \in [-d, d]$; we call $d$ the range.
- How to increase the range?
  - increase $\ell$ and search for an $\ell$-bit prime with "good" range
  - can we increase the range while keeping $\ell$ fixed?
How to Increase the Range?

- We want to find sign of $x \in [-d, d]$; we call $d$ the range.
- How to increase the range?
  - increase $\ell$ and search for an $\ell$-bit prime with “good” range
  - can we increase the range while keeping $\ell$ fixed?

Why keeping $\ell$ fixed?
Cost of local computations grows with $\ell$. 
New Idea: Correcting “Errors” via Majority Vote

Let’s take again $p = 311$

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | … |
|-----|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|-----|
| ($x_p$) | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | -1 | 1 | 1 | 1 | 1 | -1 | -1 | 1 | … |
New Idea: Correcting “Errors” via Majority Vote

- Let’s take again \( p = 311 \)

\[
\begin{array}{cccccccccccccccc}
 x & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & \ldots \\
\hline
(\frac{x}{p}) & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \ldots \\
\end{array}
\]

- Inspect neighboring Legendre symbols, compute sign of

\[
y(x) = \left(\frac{x - 1}{p}\right) + \left(\frac{x}{p}\right) + \left(\frac{x + 1}{p}\right) \in [-3, +3] \subset \mathbb{Z}
\]

- We can compute sign of \( y(x) \) using a polynomial
Definition

Let $k$ be a non-negative integer, and let $p > 2k + 1$ be a Blum prime. We define the *$k$-range of $p$*, denoted $d_k(p)$, to be the largest integer $d$ such that for all $x$ with $1 \leq x \leq d$ it holds that

$$\sum_{i=-k}^{k} \left( \frac{x+i}{p} \right) > 0,$$

and we set $d_k(p) := 0$ if no such $d$ exists.
## Concrete Examples of Triples \((\ell, p, d_k(p))\)

<table>
<thead>
<tr>
<th>(\ell)</th>
<th>(p)</th>
<th>(d_0(p))</th>
<th>(\ell)</th>
<th>(p)</th>
<th>(d_1(p))</th>
<th>(\ell)</th>
<th>(p)</th>
<th>(d_2(p))</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>11</td>
<td>1</td>
<td>5</td>
<td>23</td>
<td>5</td>
<td>5</td>
<td>23</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>23</td>
<td>4</td>
<td>5</td>
<td>31</td>
<td>10</td>
<td>7</td>
<td>71</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>71</td>
<td>6</td>
<td>8</td>
<td>167</td>
<td>13</td>
<td>8</td>
<td>167</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>311</td>
<td>10</td>
<td>8</td>
<td>191</td>
<td>19</td>
<td>8</td>
<td>191</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>479</td>
<td>12</td>
<td>8</td>
<td>599</td>
<td>20</td>
<td>8</td>
<td>239</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>1559</td>
<td>16</td>
<td>10</td>
<td>1319</td>
<td>37</td>
<td>9</td>
<td>359</td>
<td>26</td>
</tr>
<tr>
<td>13</td>
<td>5711</td>
<td>18</td>
<td>11</td>
<td>3119</td>
<td>40</td>
<td>9</td>
<td>479</td>
<td>35</td>
</tr>
<tr>
<td>14</td>
<td>10559</td>
<td>22</td>
<td>12</td>
<td>9719</td>
<td>45</td>
<td>9</td>
<td>1151</td>
<td>38</td>
</tr>
<tr>
<td>15</td>
<td>18191</td>
<td>28</td>
<td>14</td>
<td>14951</td>
<td>60</td>
<td>11</td>
<td>1511</td>
<td>41</td>
</tr>
<tr>
<td>15</td>
<td>31391</td>
<td>30</td>
<td>14</td>
<td>110039</td>
<td>65</td>
<td>12</td>
<td>3527</td>
<td>43</td>
</tr>
<tr>
<td>19</td>
<td>36679</td>
<td>42</td>
<td>17</td>
<td>21155</td>
<td>66</td>
<td>12</td>
<td>3911</td>
<td>58</td>
</tr>
<tr>
<td>22</td>
<td>408039</td>
<td>46</td>
<td>18</td>
<td>28363</td>
<td>67</td>
<td>13</td>
<td>6551</td>
<td>59</td>
</tr>
<tr>
<td>24</td>
<td>12537719</td>
<td>52</td>
<td>19</td>
<td>28951</td>
<td>72</td>
<td>14</td>
<td>8951</td>
<td>66</td>
</tr>
<tr>
<td>25</td>
<td>30760679</td>
<td>58</td>
<td>19</td>
<td>33379</td>
<td>109</td>
<td>14</td>
<td>12239</td>
<td>89</td>
</tr>
<tr>
<td>26</td>
<td>36415991</td>
<td>60</td>
<td>19</td>
<td>18847</td>
<td>129</td>
<td>15</td>
<td>25679</td>
<td>140</td>
</tr>
<tr>
<td>27</td>
<td>82636319</td>
<td>66</td>
<td>21</td>
<td>28172</td>
<td>140</td>
<td>19</td>
<td>28951</td>
<td>176</td>
</tr>
<tr>
<td>27</td>
<td>120293879</td>
<td>72</td>
<td>22</td>
<td>105225</td>
<td>149</td>
<td>20</td>
<td>66263</td>
<td>182</td>
</tr>
<tr>
<td>27</td>
<td>131486759</td>
<td>82</td>
<td>24</td>
<td>79107</td>
<td>179</td>
<td>24</td>
<td>161783</td>
<td>226</td>
</tr>
<tr>
<td>32</td>
<td>2929911599</td>
<td>96</td>
<td>25</td>
<td>251551</td>
<td>156</td>
<td>22</td>
<td>279835</td>
<td>212</td>
</tr>
<tr>
<td>33</td>
<td>7979490791</td>
<td>100</td>
<td>25</td>
<td>290369</td>
<td>157</td>
<td>24</td>
<td>1032811</td>
<td>223</td>
</tr>
<tr>
<td>35</td>
<td>33857579279</td>
<td>106</td>
<td>27</td>
<td>791073</td>
<td>179</td>
<td>24</td>
<td>1617839</td>
<td>226</td>
</tr>
<tr>
<td>37</td>
<td>89206899239</td>
<td>108</td>
<td>27</td>
<td>896587</td>
<td>217</td>
<td>25</td>
<td>174319</td>
<td>250</td>
</tr>
<tr>
<td>37</td>
<td>121560956039</td>
<td>112</td>
<td>30</td>
<td>927636</td>
<td>227</td>
<td>25</td>
<td>196327</td>
<td>255</td>
</tr>
<tr>
<td>39</td>
<td>328878629999</td>
<td>130</td>
<td>31</td>
<td>15149705</td>
<td>276</td>
<td>25</td>
<td>2538091</td>
<td>276</td>
</tr>
<tr>
<td>39</td>
<td>513928651919</td>
<td>136</td>
<td>36</td>
<td>5670962375</td>
<td>277</td>
<td>25</td>
<td>3080915</td>
<td>280</td>
</tr>
<tr>
<td>42</td>
<td>4306732833311</td>
<td>150</td>
<td>36</td>
<td>602211963</td>
<td>281</td>
<td>26</td>
<td>5342215</td>
<td>290</td>
</tr>
<tr>
<td>43</td>
<td>8402847753431</td>
<td>156</td>
<td>37</td>
<td>817202289</td>
<td>291</td>
<td>27</td>
<td>9298951</td>
<td>308</td>
</tr>
<tr>
<td>47</td>
<td>70864718555231</td>
<td>162</td>
<td>37</td>
<td>8634528671</td>
<td>339</td>
<td>28</td>
<td>2462415</td>
<td>318</td>
</tr>
<tr>
<td>49</td>
<td>317388900373231</td>
<td>166</td>
<td>38</td>
<td>18780947879</td>
<td>396</td>
<td>28</td>
<td>4426627</td>
<td>329</td>
</tr>
<tr>
<td>49</td>
<td>501108392233879</td>
<td>190</td>
<td>43</td>
<td>8714428081631</td>
<td>431</td>
<td>30</td>
<td>72125035</td>
<td>379</td>
</tr>
<tr>
<td>53</td>
<td>555118579073591</td>
<td>198</td>
<td>44</td>
<td>1042210355155</td>
<td>437</td>
<td>30</td>
<td>98409343</td>
<td>458</td>
</tr>
<tr>
<td>53</td>
<td>7832488789769159</td>
<td>222</td>
<td>44</td>
<td>1372979754247</td>
<td>443</td>
<td>35</td>
<td>18233703479</td>
<td>488</td>
</tr>
</tbody>
</table>
Application: Secure Neural Network Evaluation

Binarized multi-layer perceptron for MNIST
[Courbariaux et al., 2016, Hubara et al., 2017]
Questions