Donghang Lu*, Albert Yu, Aniket Kate, and Hemanta Maji

Polymath: Low-Latency MPC via Secure Polynomial Evaluations and its Applications

Abstract: While the practicality of secure multi-party computation (MPC) has been extensively analyzed and improved over the past decade, we are hitting the limits of efficiency with the traditional approaches of representing the computed functionalities as generic arithmetic or Boolean circuits. This work follows the design principle of identifying and constructing fast and provably-secure MPC protocols to evaluate useful high-level algebraic abstractions; thus, improving the efficiency of all applications relying on them. We present Polymath, a constant-round secure computation protocol suite for the secure evaluation of (multi-variate) polynomials of scalars and matrices, functionalities essential to numerous data-processing applications. Using precise natural precomputation and high-degree of parallelism prevalent in the modern computing environments, Polymath can make latency of secure polynomial evaluations of scalars and matrices independent of polynomial degree and matrix dimensions.

We implement our protocols over the HoneyBadgerMPC library and apply it to two prominent secure computation tasks: privacy-preserving evaluation of decision trees and privacy-preserving evaluation of Markov processes. For the decision tree evaluation problem, we demonstrate the feasibility of evaluating high-depth decision tree models in a general n-party setting. For the Markov process application, we demonstrate that Polymath can compute large powers of transition matrices with better online time and less communication.

Keywords: Secure multi-party computation; polynomial evaluation; privacy-preserving decision tree evaluation

DOI Editor to enter DOI

***Corresponding Author: Donghang Lu:** Purdue University, E-mail: lu562@purdue.edu
Albert Yu: Purdue University, E-mail: yu646@purdue.edu
Aniket Kate: Purdue University, E-mail: aniket@purdue.edu
Hemanta Maji: Purdue University, E-mail: hemanta.maji@gmail.com

1 Introduction

Secure multi-party computation (MPC) [3, 7, 30] enables mutually distrusting parties to compute securely over their private data. Informally, in a system of $n > 1$ mutually distrusting parties, an MPC protocol allows them to "securely" evaluate any agreed-on function f of their private inputs, in the presence of a centralized adversary controlling at most any $t < n$ parties[1]. After over four decades of research in MPC, only recently, the number of MPC instances for real-world use cases has increased significantly. Indeed, thanks to recent researches that significantly improved the performance of MPC, several MPC frameworks are now available to help with various use cases like privacy-preserving machine learning [33, 35, 38, 41], privacy-preserving financial solutions [11, 32], and secure information escrows [9, 29].

Typically, in MPC protocols, general-purpose compilers represent any computation/functionality (over private inputs) as a Boolean or an arithmetic circuit. The secure computation proceeds by inductively composing the secure protocols for the atomic elementary Boolean/arithmetic gates [11, 29, 33, 42]. Although this natural approach is complete, it tends to introduce significant overheads. The privacy-preserving computations are at least a few orders of magnitude slower than their non-private counterparts. Moreover, the overhead increases as we add more parties or consider a more powerful adversary. While the growing demand for privacy, in the form of privacy regulations (for example, GDPR [14] and CPRA/CCPA [24]) and user expectations, is here to drive the use of MPC in a broad spectrum of online applications, the slowdowns among most of the current privacy solutions are far from being acceptable.

There are opportunities to significantly improve the efficiency of these secure computation protocols by identifying optimizations within specific computation tasks. However, carefully handcrafting provably-secure MPC solutions for every interesting application is not scal-

[1] For different system adversary settings, MPC may have stricter requirements such as $t < \frac{n}{2}$ or $t < \frac{n}{3}$.

able. Ad hoc protocol design, on the other hand, has historically been riddled with latent security vulnerabilities. Consequently, the time-tested approach is to strike a natural balance between these two extreme techniques by identifying shared algorithmic components underlying several classes of computations of high societal impact. After that, one designs optimized secure computation solutions for these individual components, thus, permeating the efficiency gains to all application domains relying on them.

This work follows the design principle of identifying and constructing fast and provably-secure MPC protocols to evaluate useful high-level algebraic abstractions. We consider secure multi-party polynomial evaluations of scalars and matrices, and present constant-round MPC solutions for them. This class of high-level algebraic primitive finds widespread applications ranging from approximating non-linear functions (e.g. the Sigmoid function) to evaluating decision trees [23].

1.1 Our Contribution

We propose Polymath, a versatile, constant-round protocol suite focusing on improving MPC performance of polynomial evaluation over both scalars and matrices. Our protocol suite applies to *any* arithmetic-circuit-based MPC computation, and the numbers of required communication rounds are independent of the number of participating parties, the degree of the evaluated polynomial as well as the matrix dimensions.

Specifically, we focus on the following two functionalities:

(a) Secure Evaluation of Polynomial over Finite Fields. We design 2-round protocols for evaluating multivariate polynomials, and the communication complexity (i.e., the number of communicated bits) only increases linearly with the number of variables and is independent of polynomial degree. Our highly parallelizable protocols employ specially crafted precomputations and offer significant improvements over the state of the art techniques that require communication rounds logarithmic in the number of variables. We also present 2-round protocols for evaluating univariate polynomials, which extends a secure exponentiation protocol by Damgård et al. [20].

(b) Secure Evaluation of Polynomials over Matrices. We present a 4-round protocol for secure matrix powering/exponentiation and make use of it to evaluate univariate polynomials over matrices securely. Our protocol is not only highly parallelizable but also with a communication complexity independent of the exponent.

Despite the conceptual similarity, our protocols for secure polynomial evaluation for scalars over finite fields and those over matrices are inherently different as the multiplication of finite field elements is commutative, while matrix multiplication is not commutative in general.

Polymath is not only theoretically interesting, but we find it to be practically relevant to any computation that can be represented as a polynomial. We demonstrate this practical relevance using two representative applications.

Application I: Privacy-Preserving Decision Tree Evaluation. We present a solution for privacy-preserving decision tree evaluation as an application of Polymath, and it illustrates that we can achieve high depth decision tree evaluation within a reasonable time. To the best of our knowledge, our solution is the first to support general n-party setting and high-depth tree evaluations simultaneously.

Polymath can be employed with any arithmetic-circuit MPC library such as SPDZ [28], or Scale MAMBA [19]. We implement our protocols using the HoneyBadgerMPC library [30] for robust execution in the asynchronous setting. For a depth 8 complete decision tree model, we can evaluate it with 12 seconds under 4-party setting or 13 seconds under 7-party setting.

Application II: Secure Credit Risk Analysis through a Markov Process. We use our matrix powering protocol to solve the evaluation of a Markov process. Furthermore, we show how one can use this computation to perform credit risk analysis in financial domains. The benchmark shows that in the 4-party setting, we can evaluate the power of 10×10 transition matrices (with the exponent being 1024) in (roughly) half a second. Furthermore, we can evaluate the power of 1024 for 320×320 transition matrices in around 20 seconds.

Paper Organization. In Section 2 we present the general system setting of Polymath. In sections 3 and 4, we introduce the protocols for multi-variate polynomial evaluation. We provide a security analysis for protocols in Section 5. Then in Section 6, we illustrate one of the applications of Polymath: privacy-preserving decision evaluation, which is of unique interests. Then we discuss related works in Section 8 and conclude the paper in Section 9.

2 Problem Setting

2.1 System Model

We consider a standard MPC setting with a set of parties $P_1, P_2, \dots P_n$ for $n \geq 2$, where the parties are connected via authenticated and secure point-to-point channels, where everyone can send messages to each other at the same time. In this setting, MPC has three distinct phases: parties share their input in phase 1, they participate in multi-party computation over the shared input in phase 2, and finally reconstruct output by combining their shares in phase 3.

The proposed Polymath techniques work across different communication settings and, thus, we do not make any assumption regarding the bounded-synchrony, partial-synchrony, or asynchrony of the underlying MPC setting; however, we measure our protocols' efficiency in terms of the number of rounds (or the round complexity) and we specify it below for different communication settings.

MPC protocols for the bounded-synchronous communication setting assume that parties proceed in rounds such that the messages sent by any honest party in any given round are delivered to every recipient in the same round. All parties here are assumed to be somewhat synchronized and to be in the same round at all times; thus, the number of rounds is evident from the protocol steps.

For the definition of rounds in the partially-synchronous and asynchronous settings, we follow [15]. Intuitively, when two messages m and m' sent by party P_i in an asynchronous MPC are considered to be sent in rounds r and r', $r' > r$, if m' can be computed only after P_i has sent m. Here, the number of asynchronous protocol rounds is the maximum (over all honest parties) number of rounds that an honest party uses in the protocol execution.

Polymath is also agnostic to the underlying adversary model, and we leave the adversary model as well as communication setting discussion to the individual application setting.

2.2 Linear Secret Sharing based MPC

We focus on the arithmetic operations for our setting and thus input as well as all intermediate results are represented in the form of a linear secret sharing[2] among n parties.

Based on the adversary assumption, we pick appropriate linear secret sharing (or secure representation) of the input and intermediate results. For the dishonest majority MPC (e.g., SPDZ [28]), the employed scheme is additive secret sharing, while we consider Shamir secret sharing [39] for the honest majority setting. Against a malicious adversary, the secret sharing choices can be verifiable secret sharing [10] or authenticated secret sharing [28].

An (n, t) Shamir secret sharing scheme, with $n > t \geq 0$ allows n parties $\{P_1, \dots, P_n\}$ to obtain shares of a secret in F_p, and the secret can be revealed if and only if $t+1$ or more parties combine their shares to reconstruct the secret value. Here, the secret is encoded into the constant coefficient of a degree-t polynomial over $F_p[x]$. We use $[s]_t$ to represent a Shamir secret shared value $s \in F_p$ with threshold t. Namely, a degree-t polynomial ϕ is used with $\phi(0) = s$. The share that each party P_i gets is the point $(i, \phi(i))$, which we denote as $[s]_t^{(i)}$.[3]

For the arithmetic-circuit MPC, a computed functionality is represented using addition and multiplication gates. As all linear secret sharing schemes are additive homomorphic, any linear combination of multiple sharings can be computed locally by party P_i by applying the same linear function on its shares. However, the multiplication of two shares cannot be realized in the same way. For Shamir secret sharing, the multiplication of two degree-t polynomials will result in a degree-$2t$ polynomial. As a result, in secret sharing based MPC we often follow the online/offline MPC paradigm and use Beaver triples [4] to deal with the multiplication of two secret shares.

The online/offline MPC paradigm leverages an offline phase to generate input and functionality-

[2] A linear secret sharing [17] is a standard cryptographic technique that allows a secret taken from a finite field F_p to be distributed among n parties such that each party's share is obtained by computing some linear function of the secret and the dealer's randomness, and that the secret can only be reconstructed when a sufficient portion of shares are combined with a linear function.

[3] In the malicious adversary setting, $[s]_t^{(i)}$ may contain further elements in the form of commitments and/or zero-knowledge proofs; however, we ignore those here as the Polymath techniques are independent to those.

independent preprocessed sharings so that these sharings are used to speed up the online phase where parties compute MPC functionalities. It is allowed that the offline phase is more costly as it can be run for a long time before the online phase. For the offline phase, many state-of-the-art offline protocols [3, 6] are perfectly compatible with our solutions.

Beaver Triple Assisted Multiplication. In linear secret sharing based MPC, additions are free and Beaver's triple are widely used to manage multiplication. To multiply two secret sharings $[x]$ and $[y]$ (with threshold t), a precomputed triple $([a],[b],[c])$ is generated during the offline phase, where $c = a \cdot b$. In the online phase, all parties compute and reconstruct (denoted as **Open**$(\cdot)$) $x-a$ and $y-b$, then the result of multiplication is $[xy] = (x-a)(y-b)+(x-a)[b]+(y-b)[a]+[c]$. The protocol is summarized in Algorithm 1.

```
Algorithm 1: Beaver Multiplication Mul([x],[y])
  Input           : $[x], [y]$
  Output          : $[xy]$
  Pre-computation: $[a], [b]$ and $[ab]$ where $a, b$
                    are random values..
  $[x-a] = [x] - [a]$
  $[y-b] = [y] - [b]$
  $(x-a) = $ Open$([x-a])$ // Round 1
  $(y-b) = $ Open$([y-b])$ // Round 1
  return
    $(x-a)(y-b) + (y-b)[a] + (x-a)[b] + [ab]$
```

Securely Evaluating the Power of a Secret Share. Damgård et al. [20] introduced an efficient method to calculate $[x^n]$ given $[x]$ where n is a publicly known value. The idea is to use the following pre-computed values: $[a]$ and $[a^{-n}]$ where $[a]$ is a random share. The protocol proceeds as follows: First we multiply $[a]$ and $[x]$ to get $[ax]$. The second round is to open the value $[ax]$. Then the result could be written as $(ax)^n[a^{-n}]$. The round complexity of the protocol is 2.

Beaver triple techniques for matrix multiplication. To multiply two k by k matrices, the naive approach is using k^3 Beaver multiplications to compute each cell, leading to $O(k^3)$ communication. Mohassel and Zhang [33] propose a generalized version of Beaver triple technique to deal with matrices. Instead of Beaver triples, three matrices A, B, C are generated in the offline phase where $C = A \cdot B$ and all elements in A and B are uniformly random. The online computation phase is almost the same as Beaver triple multiplication, and the only difference is that the additions/multiplications of field elements are replaced by those of matrices. For simplicity, we call this protocol Beaver matrix multiplication in the rest of the paper and we use it to multiply two secret-shared matrices by default. The communication complexity of the above protocol is $O(n^2)$.

2.3 Notations

Here we summarized notations employed in Algorithm 1 as well as those that appear in the following sections. We denote $[s]$ as a secret sharing with the secret field element s, and we use capital letters to represent matrices (S and $[S]$), where $[S]$ means that every cell/element in matrix S is secret shared. **Open**$([s])$ means the reconstruction phase where parties exchange their shares to recover the secret. Furthermore, we use **Mul**$([x]$,$[y])$ to represent the Beaver triple multiplication for two secret shares $[x]$ and $[y]$ or two matrices $[X]$ and $[Y]$. We use **Pow**$([x]$,$e)$ to compute x^e where x is a field element and e is a positive integer. In algorithm boxes, we use // as the symbol of comments, instead of division.

3 Secure Computations of Polynomial Evaluation

3.1 Secure Computation for Univariate Polynomials

Univariate polynomials can be written in a standard form: $P(x) = \sum_{i=0}^{n} a_i x^i \in F_p[x]$ where $a_i \in F_p$ are plaintext coefficients. As a result, the problem of evaluating univariate polynomials reduces to computing the power of the secret share $[x]$.

With the protocol proposed by Damgård et al. [20], which is mentioned in Section 2, we can compute all required $[x^i]$ in parallel, then multiply them with corresponding plaintext coefficients. The whole process takes two rounds since the second part is local computation. See Algorithm 2.

Algorithm 2: Univariate Polynomial Evaluation based on [20]

```
Input            : $[x]$, a polynomial $P(x) = \sum_{i=0}^{n} a_i x^i$
Output           : $P([x])$
Pre-computation: $[r_1], \ldots, [r_n]$ and $[r_1^{-1}], \ldots, [r_n^{-n}]$ where $r_i$s are random values.
for $i \leftarrow 1$ to $n$ do
    $[x^i] = \textbf{Pow}([x],\ i)$ // Round 1 & 2
for $i \leftarrow 0$ to $n$ do
    $P_i([x^i]) = [x^i] \cdot a_i$
return $P([x]) = \sum P_i([x^i])$
```

3.2 How to calculate multi-variable polynomials

We start with some basic protocols to compute simple multi-variable terms, then we extend the protocol step-by-step. Finally, we show how we achieve the evaluation of high-degree multi-variable polynomials.

3.2.1 Efficient Way to Calculate $[xyz]$

Beaver triple technique illustrates how to multiply 2 variables with pre-computed triples. We extend this idea and explore how to calculate multiplication of 3 variables $[x]$, $[y]$, and $[z]$ in one round. The precomputation required by our protocol is the following: $[a]$, $[b]$, $[c]$, and $[abc]$ where a, b, c are random elements. Similar to Beaver's idea, $[a]$, $[b]$, and $[c]$ are used to blind $[x]$, $[y]$, and $[z]$. The computation is based on the following formula:

$$[(x-a)(y-b)(z-c)] = [xyz] - [xyc] - [xbz] + [xbc] \\ -[ayz] + [ayc] + [abz] + [abc]$$

We first open the values $(x-a), (y-b)$, and $(z-c)$. Then we can get the following formula by taking the opened values as constants and combining terms with the same prefix:

$$(x-a)(y-b)(z-c) = [xyz] - [xc](y-b) \\ -[bz](x-a) - [ay](z-c) - [abc]$$

This formula illustrates that the multiplication of three variables reduces to three multiplication of two variables by the help of $[a]$, $[b]$, $[c]$, and $[abc]$. Three normal Beaver triples are required for solving 3 two-variable multiplication.

The protocol is formally described in Algorithm 3. The round complexity is 1 since the opening of $(x-a), (y-b)$, and $(z-c)$ could be done simultaneously with two-variable multiplications. The required invocation of broadcasts is 9 where 3 broadcasts are used to open $(x-a), (y-b)$, and $(z-c)$ and 6 broadcasts deal with two-variable multiplications. As a trade-off, the computation overhead is higher than simply using Beaver multiplications twice. So our method provides an alternative that achieves better performance in high-latency networks.

Algorithm 3: Tri-variate Term Evaluation

```
Input            : $[x], [y], [z]$
Output           : $[xyz]$
Pre-computation: $[a], [b], [c]$ and $[abc]$ where $a, b, c$ are random values. Three Beaver triples.
$[x-a] = [x] - [a]$
$[y-b] = [y] - [b]$
$[z-c] = [z] - [c]$
$(x-a) = \textbf{Open}([x-a])$ // Round 1
$(y-b) = \textbf{Open}([y-b])$ // Round 1
$(z-c) = \textbf{Open}([z-c])$ // Round 1
$[xc] = \textbf{Mul}([x],[c])$ // Round 1
$[bz] = \textbf{Mul}([b],[z])$ // Round 1
$[ay] = \textbf{Mul}([a],[y])$ // Round 1
return $(x-a)(y-b)(z-c) + [xc](y-b) + [bz](x-a) + [ay](z-c) - [abc]$
```

The proposed method implies that the idea of Beaver multiplication could be extended to more variables. However, the computation and the offline cost increase exponentially with the number of variables. As a result, we need a better technique to deal with the general multi-variable multiplication. Although we did not include this protocol in our final solution, it could potentially be useful and the same idea could be extended to other fields (e.g. it can be naturally extended to matrix multiplication).

3.2.2 Efficient Way to Calculate Multi-variable Multiplication

In this section we take one step further and introduce a method to solve multi-variable multiplication. Suppose we want to calculate the product of $[x_1]$, $[x_2]$, …, $[x_n]$, our protocol requires pre-computed values in the form of $[a_1]$, $[a_2]$, …, $[a_n]$, $[(a_1 a_2 \ldots a_n)^{-1}]$ where $a_1, a_2, \ldots, a_n$ are random values.

The protocol proceeds as follows: First we use Beaver multiplications to multiply $[x_i]$ and $[a_i]$. Then we open all the multiplication results $x_i a_i$. Finally, we get the result by multiplying all the opened values together with pre-computed value as follows: $x_1 a_1 x_2 a_2 \dots x_n a_n [(a_1 a_2 \dots a_n)^{-1}]$.

The round complexity of the protocol is 2. And the communication complexity increases linearly with the number of variables. Compared with the method in the previous section, this protocol has advantages both on the round complexity and offline costs. The protocol is formally described in Algorithm 4.

Algorithm 4: Evaluation of degree-1 arbitrary-variate term

```
Input           : $[x_1], [x_2], \dots, [x_n]$
Output          : $[x_1 x_2 \dots x_n]$
Pre-computation : $[r_1], [r_2], \dots, [r_n]$ and
                  $[(r_1 r_2 \dots r_n)^{-1}]$ where $r_i$s
                  are random values. $n$
                  Beaver triples.
for $i \leftarrow 1$ to $n$ do
    $[x_i r_i] = \mathbf{Mul}([x_i], [r_i])$ // Round 1
for $i \leftarrow 1$ to $n$ do
    $x_i r_i = \mathbf{Open}([x_i r_i])$ // Round 2
return $x_1 r_1 x_2 r_2 \dots x_n r_n \cdot [(r_1 r_2 \dots r_n)^{-1}]$
```

3.2.3 Efficient Way to Calculate $x_1^{e_1} x_2^{e_2} \dots x_n^{e_n}$

To solve the problem of polynomial evaluation, we need to deal with terms with the most generalized form: $x_1^{e_1} x_2^{e_2} \dots x_n^{e_n}$. We extend the ideas in previous sections to solve this problem. In our protocol we require the pre-computed values in the form of $[r_1]$, $[r_2]$, ..., $[r_n]$, and $[(r_1^{e_1} r_2^{e_2} \dots r_n^{e_n})^{-1}]$. The protocol proceeds as follows:

First we multiply $[x_i]$ with $[r_i]$ for i from 1 to n. Then we open all these products to get all $x_i r_i$. After that, the result could be written as $(x_1 r_1)^{e_1} \dots (x_n r_n)^{e_n} [(r_1^{e_1} r_2^{e_2} \dots r_n^{e_n})^{-1}]$.

The protocol is formally described in Algorithm 5. The round complexity is always 2 and the communication complexity is linear with the number of variables. Given a polynomial, we can apply this protocol to each term in parallel and as a result, any polynomial with arbitrary degrees can be evaluated in two rounds.

Algorithm 5: Evaluation of multi-variate polynomial term

```
Input           : $[x_1], [x_2] \dots [x_n]$ and
                  $e_1, e_2, \dots, e_n$
Output          : $[x_1^{e_1} x_2^{e_2} \dots x_n^{e_n}]$
Pre-computation : $[r_1], [r_2], \dots [r_n]$ and
                  $[(r_1^{-e_1} r_2^{-e_2} \dots r_n^{-e_n}]$ where
                  $r_i$s are random values. $n$
                  Beaver triples..
for $i \leftarrow 1$ to $n$ do
    $[x_i r_i] = \mathbf{Mul}([x_i],[r_i])$ // Round 1
for $i \leftarrow 1$ to $n$ do
    $x_i r_i = \mathbf{Open}([x_i r_i])$ // Round 2
for $i \leftarrow 1$ to $n$ do
    $y_i = (x_i r_i)^{e_i}$ // local computation
return $y_1 y_2 \dots y_n \cdot [(r_1^{-e_1} r_2^{-e_2} \dots r_n^{-e_n}]$
```

3.3 Offline Phase

For each polynomial term $x_1^{e_1} x_2^{e_2} \dots x_n^{e_n}$, the required pre-computed values are $[r_1], \dots, [r_n]$, and $[r_1^{-e_1} r_2^{-e_2} \dots r_n^{-e_n}]$. The straight forward way to generate these values is to calculate powers first then multiply all random terms together. Given a random share $[r]$, it requires approximately $log(e)$ multiplications to compute $[r^e]$ by running Beaver multiplication repeatedly.

As a result, the offline phase of Algorithm 5 requires the generation of $2n$ random elements, $n + \sum log(e_i)$ MPC multiplication and n reconstruction, where n is the number of variables. The communication complexity of each of above operations could be linear with number of parties if we pick proper sub-protocols (e.g. [3]). The overall round complexity is $O(log(max(e_i)) + log(n))$ considering both the computation of $[r^{e_i}]$ and the multiplication of all $[r_i^{e_i}]$ can be parallelized.

4 Secure Computation of Polynomials of Matrices

In this section, we discuss polynomial evaluation over matrices where each matrix entry is a finite field element. Although it is conceptually similar to the scalar polynomial evaluation, the protocols for matrices are inherently different because matrix multiplication is not commutative in general.

4.1 Multiplying an arbitrary number of matrices

To solve polynomials over matrices, we need to evaluate the terms that consist of the product of multiple matrices. Bar-Ilan and Beaver [2] propose a constant-round protocol to achieve it. To multiply n matrices $X_1, X_2, \ldots .X_n$ (for simplicity of description, we assume all of them are k by k square matrices, while this solution also works for matrices with arbitrary dimensions), it requires $n+1$ random matrices $R_0, R_1, R_2, \ldots .R_n$, which have the same dimensions as X_is and contain random values, and their corresponding inverse matrices $R_0^{-1}, R_1^{-1}, R_2^{-1}, \ldots .R_n^{-1}$.

The protocol is illustrated in Algorithm 6 and the round complexity is four.

```
Algorithm 6: Multiply an arbitrary number of matrices in [2]
  Input           : $[X_1], [X_2], \ldots, [X_n]$
  Output          : $[X_1 X_2 \ldots X_n]$
  Pre-computation: $[R_0], \ldots, [R_n]$ and
                   $[R_0^{-1}], \ldots, [R_n^{-1}]$
for $i \leftarrow 1$ to $n$ do
    $[R_{i-1}X_i] = \textbf{Mul}([R_{i-1}],[X_i])$
    $[R_{i-1}X_iR_i^{-1}] = \textbf{Mul}([R_{i-1}X_i],[R_i^{-1}])$
for $i \leftarrow 1$ to $n$ do
    $R_{i-1}X_iR_i^{-1} = \textbf{Open}([R_{i-1}X_iR_i^{-1}])$
$R_0X_1 \ldots X_nR_n^{-1} = \prod_{i=1}^{n} R_{i-1}X_iR_i^{-1}$
return $\textbf{Mul}([R_0^{-1}] \cdot R_0X_1 \ldots X_nR_n^{-1},\ [R_n])$
```

The other alternative is to apply Beaver matrix multiplication for $\log n$ rounds. In the first round, we multiply every two matrices together so that the number of resulted matrices is reduced by half. We keep doing it iteratively, and after $\log n$ rounds, there is only one matrix left which is the result. Compared with this idea, the protocol in [2] has constant round complexity but as a trade-off, it requires more local computation and communication.

We implement both of them in HoneybadgerMPC to compare the performance of these two protocols. the implementation detail and the benchmark result are elaborated later in Section 7.6. The result demonstrates that the second method outperforms the first in almost all test cases. The reason is due to the nature of matrix multiplication: the local computation and communication become the bottleneck quickly with the increase of the dimensions. And the local computation can easily be measured in terms of seconds, thus canceling the benefits gained by constant round complexity which is usually several hundred milliseconds.

However, we find out that by extending the protocol in [2], we can achieve a very efficient protocol for matrix powering.

```
Algorithm 7: Matrix Powering
  Input           : matrix in secret shared form with demension $k$ by
                    $k$: $[X]$ and the exponent $e$
  Output          : $[X^e]$
  Pre-computation: $[R]$ and $[R^{-1}]$ where $R$ consists of $k$ by $k$ random
                   values
$[RX] = \textbf{Mul}([R],[X])$ // Round 1
$[Y] = \textbf{Mul}([RX],[R^{-1}])$ // Round 2
Y = $\textbf{Open}([Y])$ // Round 3
return $\textbf{Mul}([R^{-1}] \cdot Y^e,\ [R])$ // Round 4
```

4.2 Calculating powers of a square matrix

The power of square matrices is a special case of matrix multiplication. In this section, we propose a method to calculate the powers of a square matrix, which saves significant precomputation and communication, especially when computing a large power. However, this method only achieves a lower level of privacy since some information about the input matrix is leaked (for example, the rank of the input matrix). Please refer to Section 5 for a detailed analysis regarding the privacy loss. We find this method meaningful as in many use cases it is acceptable to leak that information to save significant computation.

The protocol is described in Algorithm 7. The observation is that we only need one random matrix R in the case of matrix powering, and we only need to calculate and open $[RXR^{-1}]$ one time. It means the round complexity and communication complexity are independent of the power that we want to compute.

The state-of-the-art solution for matrix powering that we know so far is to use Beaver matrix multiplication for roughly $\log e$ times to get $[X^e]$[4]. As a result, it requires $\log(e)$ rounds involving opening $2 \cdot \log(e)$ matrices. Compared with it, our protocol requires 4 rounds and the communication cost is opening 7 matrices. As for the computation complexity. Each Beaver matrix multiplication takes 3 local matrix multiplications and 6 additions. Therefore, our protocol requires $9 + \log(e)$ local matrix multiplications and 18 additions. Meanwhile, the state-of-the-art protocol needs $O(\log(e))$ number of multiplications.

4.3 Offline Phase

For matrix powering protocol, the required precomputation is in the form of A, A^{-1}, namely a matrix with random elements and its inverse matrix. Here we provide an efficient 2-round protocol to compute the inverse matrix given a secret shared matrix. The protocol is summarized in Algorithm 8.

Algorithm 8: Precomputation for Matrix Powering

```
Input           : secret shared matrix $[X]$
Output          : $[X^{-1}]$
Pre-computation: $[R]$, matrix $R$ share the
                  same dimension with $X$
                  and contain random
                  values.
$[XR] = \mathbf{Mul}([X], [R])$ // Round 1
$XR = \mathbf{Open}([XR])$ // Round 2
$R^{-1}X^{-1} = (XR)^{-1}$
return $[R]R^{-1}X^{-1}$
```

The cost of this protocol includes generating k^2 random shares, one matrix multiplication, and one matrix reconstruction, all of which have communication complexity of $O(k^2)$ for k by k matrices. Thus the cost of the offline phase is of the same order of magnitude as the online phase.

The above protocol is efficient but there is a very small probability of failure since XR may not have an inverse. By Lemma 3 of Section 5, the probability of a random matrix being not invertible is less than $\frac{1}{p} + \frac{1}{p^2}$ where p is the size of the field, and it is small enough for us to use in practice.

5 Security Analysis

The correctness for these protocols is trivial from the protocol description and we defer it to an extended version. In this section, we analyze the security of the protocols. While most complete proofs are available in appendices, we prove here that the distributions of the transcripts of our protocol under different inputs are indistinguishable from one another, thereby proving that our protocol reveals no information about the input.

5.1 Intermediate Lemmas

We first provide a list of Lemmas, whose proofs are available in Appendix A.1. For generality, we are considering F_q, where q is a power of a prime p.

Lemma 1. *Let x', x'' be arbitrary field elements in $\mathbb{F}_q$, and r be an independent field element chosen uniformly at random from the field $\mathbb{F}_q$. Then, the distributions of $x' - r$ and $x'' - r$ are identical, and, therefore, (perfectly) indistinguishable from each other.*

Lemma 2. *Let x', x'' be arbitrary field elements in $\mathbb{F}_q \setminus \{0\}$, and r be an independent field element chosen uniformly at random from the field $\mathbb{F}_q$. Then, the distributions of $x' \cdot r$ and $x'' \cdot r$ are identical, and, therefore, (perfectly) indistinguishable from each other.*

Lemma 1 and Lemma 2 are used to prove Theorem 1 and Theorem 2.

Lemma 3. *Fix any $n \in \{1, 2, 3, \dots\}$. Let R be a matrix chosen uniformly at random from the set of all n-by-n matrices where each element is a field element from $\mathbb{F}_q$. The probability that the matrix R does not have an inverse (i.e., the matrix R is singular) is at most $\frac{1}{q} + \frac{1}{q^2}$.*

Let $GL(n, q)$ represent the set of all invertible n-by-n matrices with elements in $\mathbb{F}_q$ (read as, The General Linear Group). One can define a multiplicative group over this set using (standard) "matrix multiplication" as the operation.

Lemma 4. *Let X', X'' be arbitrary elements from $GL(n, q)$. Let R be a random n-by-n matrix chosen uni-*

[4] If e is not a power of two, we can represent e with its binary representation, then apply similar techniques to compute each binary term.

formly at random from the group $GL(n, q)$. Then, the distributions of $X' \cdot R$ and $X'' \cdot R$ are identical, and, therefore, (perfectly) indistinguishable, from each other.

Lemma 5. *Fix an arbitrary matrix $X \in GL(n, q)$. Let $Cl(X) \subseteq GL(n, q)$ be the conjugacy class of the matrix X. Fix any $\lambda \in Cl(X)$. For a matrix $R \in GL(n, q)$ chosen uniformly at random, we have the guarantee that* $\Pr[RXR^{-1} = \lambda] = \frac{1}{|Cl(X)|}$.

Observe that RXR^{-1} is always in the conjugacy class $Cl(X)$, for any $R \in GL(n, q)$. This lemma states that the random variable RXR^{-1} is uniformly random over the set $Cl(X)$, where R is uniformly random in $GL(n, q)$.

Above Lemmas are applied in the proof of Theorem 3.

5.2 Analysis regarding multi-variable polynomials

5.2.1 Security analysis for the protocol tri-variate term evaluation

The security of computing $[xyz]$ is very similar to traditional Beaver multiplication. Intuitively, since a, b, c are chosen uniformly at random, $(x - a), (y - b)$, and $(z - c)$ are all indistinguishable from random values. Therefore, revealing $(x - a), (y - b)$, and $(z - c)$ does not reveal anything about x, y, or z. The other operations are the addition of two values and multiplication by a public value, where the security comes from the security of additive homomorphic secret sharing, as well as operations involving traditional Beaver multiplications where the security comes from traditional Beaver multiplications.

We define the transcript as all the values an arbitrary party can see during the execution of our protocol. We also focus on parts of the transcript that are unique to our protocol, which in this case is $(x-a), (y-b), (z-c)$. We prove that given different values of x, y, z, the probability distribution of the transcript remains the same.

Theorem 1. *Let $x', y', z', x'', y'', z''$ be arbitrary values. The probability distribution of each transcript are identical when $x = x', y = y', z = z'$ and when $x = x'', y = y'', z = z''$.*

5.2.2 Security analysis for the protocol evaluation of degree-1 arbitrary-variate term

Intuitively, since each a_i is chosen uniformly at random, each $x_i a_i$ is indistinguishable from random values. Therefore, we leak no information by revealing $x_i a_i$. (Note however that there is a negligible probability that our protocol may be incorrect if $a_i = 0$).

We define the transcript as all the values an arbitrary party can see during the execution of our protocol. We also focus on parts of the transcript that are unique to our protocol, which in this case is $x_1 a_1, x_2 a_2, \ldots x_n a_n$. We prove that given different values of $x_1, x_2, \ldots x_n$, the probability distribution of the transcript remains the same.

Theorem 2. *Let $x'_1, x'_2, \ldots x'_n, x''_1, x''_2, \ldots x''_n$ be arbitrary values. The probability distribution of each transcript are identical when $x_1 = x'_1, x_2 = x'_2, \ldots x_n = x'_n$ and when $x_1 = x''_1, x_2 = x''_2, \ldots x_n = x''_n$ except with a negligible probability.*

5.2.3 Security analysis for the protocol of evaluation of multi-variate polynomial term

The security of calculating polynomial terms is the combination of the security for the secret powering protocol in [20] and the security for calculating multi-variable multiplication. We refer readers to [20] for more details about the security of the secret powering protocol.

5.3 Analysis Regarding Secure Computation of Matrix Powering

5.3.1 Security analysis for the protocol Matrix Powering

Intuitively, while R is random, R and R^{-1} are not independent. Therefore, RXR^{-1} will be in the same conjugacy class as X. We leak the conjugacy class of X by revealing RXR^{-1}.

The key observation is that if we restrict X' and X'' to be in the same conjugacy class, then the probability distribution of RXR^{-1} is identical when $X = X'$ and when $X = X''$.

Given that we know the conjugacy class of X, we prove that our protocol reveals no additional information about X by showing that given two different values of X, the probability distributions of possible transcripts remain the same. Below is the proof outline:

Theorem 3. *Let X', X'' be arbitrary invertible matrices under the constraint that they belong to the same conjugacy class. The probability distribution of $RX'R^{-1}$ is identical to $RX''R^{-1}$, where R is a random square matrix with the same dimension as X'*

Proof Outline. Note that with high probability, X', X'', R, R^{-1} are all in $GL(n, q)$ (The General Linear Group, where $GL(n, q)$ is the set of all n by n invertible matrices with elements from a finite field of size q) by lemma 3.

Define $Cl(X')$ as the conjugacy class of X'. $\forall\lambda, \Pr[RX'R^{-1} = \lambda] = \Pr[R \in S'] = \frac{|S'|}{|GL(n,q)|} = \frac{1}{|Cl(X')|}$ where S' is a specific subset of $GL(n, q)$ of size $\frac{|GL(n,q)|}{|Cl(X')|}$ by Lemma 5.

$\forall\lambda, \Pr[RX''R^{-1} = \lambda] = \Pr[R \in S''] = \frac{1}{|Cl(X'')|}$ where S'' is a specific subset of $GL(n, q)$ of size $\frac{|GL(n,q)|}{|Cl(X'')|}$.

$\forall\lambda, \Pr[RX'R^{-1} = \lambda] = \Pr[R \in S'] = \frac{1}{|Cl(X')|} = \frac{1}{|Cl(X'')|} = \Pr[R \in S''] = \Pr[RX''R^{-1} = \lambda]$ since X' and X'' belong to the same conjugacy class so $|Cl(X')| = |Cl(X'')|$. □

Above summarizes the key part of the proof. We refer readers to Appendix A.2.3 for the outline of the full proof.

5.4 Connection between Security and revealing its Conjugacy Class

As mentioned above, protocol Matrix Powering reveals the conjugacy class of the matrix X. We will now discuss the effects of revealing the conjugacy class of X.

The matrices in the same conjugacy class share the same rank and characteristic polynomial (and statistics that can be derived from them, such as the determinant, trace, and spectrum). Beyond that, the full impact of revealing the conjugacy class of the matrix specific to particular applications is not entirely well-understood, and needs further investigation. This impact on security potentially depends on the particular application. For example, if the output of the secure computation is ABA^{-1} for private matrices A and B, then revealing this information is secure. However, if the output of the secure computation hides some part of the conjugacy class information (for example, computing the trace securely), then revealing the conjugacy class is insecure.

There are approaches to mitigate the potential problems. For instance, we can add a slight perturbation to the matrix (e.g. change the last bit of one entry), such that the matrix falls into another conjugacy class, without decreasing its usability. The results in [33] illustrate that small errors caused by multiplication and truncation almost have no influence on the accuracy of privacy-preserving machine learning. And a slight perturbation of the matrix could be expected to have the same effect as those small errors.

5.5 Security Against Malicious Adversary

We claim our protocols' security in both semi-honest and malicious settings and elaborate our claim here. Our protocols utilize other protocols as building blocks (such as secret sharing schemes). Beyond that, our protocol does not reveal any additional information (with the exception of the protocol Matrix Powering, which reveals nothing beyond the conjugacy class). Therefore, to make our protocol secure against malicious adversaries, we only need to make sure we use linear secret sharing-based MPC (see Section 2.2) that are secure against malicious adversaries. In general, the malicious security of our protocols can be derived from the malicious security of the underlying linear secret sharing-based MPC library.

6 Privacy Preserving Decision Tree Evaluation

In recent years, privacy-preserving machine learning becomes more and more prominent and many works [1, 23, 41, 42] are deployed to achieve secure model training or secure inferences. In this work, we focus our efforts on decision trees, one of the common classifiers used in many fields such as medical treatment and finance. To the best of our knowledge, our work provides the first solution for privacy-preserving decision tree evaluation that can be applied to a general multi-party setting with the capability of evaluating high-depth models. We implement our protocols and show that our protocol can achieve very high-precision prediction for decision tree models within a reasonable time.

6.1 System Model

We consider three kinds of parties in our system model: model holders, service providers, and clients. We assume model holders are parties that hold the trained model and would like to outsource the models to service providers in a secure fashion. Clients have secret input data and expect the prediction value by the trained model. The data flow is as follows in a secret sharing based setting: model holders secret-share their models to service providers, and clients also secret-share the private input to service provides. The service providers run MPC protocols to achieve decision tree evaluation with the private tree model and private user input.

6.2 Decision Tree Representation

Decision tree representation with polynomials. Giacomelli et al. [23] describe how a tree can be represented by polynomials. That representation is a perfect match with our protocols described in Section 3 so we choose to use it in our solution. We give a brief introduction here and we refer readers to [23] for more details.

In general, a decision tree T can be represented by two types of nodes: non-leaf nodes and leaf nodes. We can always take T as a binary tree since all trees can be transformed into a binary fashion. Also to hide the topology of the decision tree, we simply transform the decision tree to be a complete tree by adding dummy nodes. The input to the tree is a vector $X = (X[1], X[2], \dots, X[n])$ where we call $X[i]$ features and the output is the prediction value y. In each non-leaf node, a comparison operation is defined by a pair (j_i, t_i) where j_i ranges from 1 to n and t_i is a threshold value. This means at this node, we will choose the left branch if $X[j_i] < t_i$, otherwise we choose the right branch. Given the input x, we follow this rule to traverse the tree starting from the root and finally reach a leaf node where the prediction value y is stored.

Assume the result of comparison in each non-leaf node N_i to be $x_i = (X[j_i] < t_i)? -1 : 1$. Then for each leaf node, we represent it with the product of d terms of the form $(x_i - 1)$ and $(x_i + 1)$ as follows: starting from the root node, $(x_i - 1)$ is chosen if the root-leaf path chooses the left branch at node N_i, otherwise $(x_i + 1)$ is chosen. Since we consider a complete binary tree. So each leaf node L_i can be represented as a polynomial P_i of degree d containing d factors of $(x_i - 1)$ or $(x_i + 1)$. Now given an input x, it can be observed that there is only one P_i that the evaluation of P_i is non-zero. And it means the value stored in the leaf-node L_i is the prediction y of the input x.

As a result, we can solve the decision tree evaluation with many comparisons and one single polynomial $T(x) = \sum P(i) \cdot inv_i \cdot y_i$ where $P(i)$ is the polynomial representing the leaf node L_i, inv_i is the inverse value of $P(i)$ when $P(i)$ is non-zero (the value of $P(i)$ is 2^d if there is even number of $(x_i - 1)$, or -2^d otherwise), and y_i is the prediction stored on the leaf node L_i. Given an input x, only one leaf node L_i is reached in the end and so that $P(i) \cdot inv_i = 1$ and $P(j) \cdot inv_j = 0$ for all $j \neq i$, therefore $T(x) = y_i$.

Fixed-point numbers representation. Considering the feature space is typically $\mathbb{R}$, we use fixed-point numbers as data format. Therefore we require fixed-point number algebra in secure computation scenario. For data representation, we follow the standard proposed in [12, 13] and consider a multi-party computation framework based on Shamir secret sharing over a prime field $\mathbb{Z}_q$. The data type that we consider are signed integers and signed fixed-point numbers. We encode a k-bit integer $x \in \mathbb{Z}_{<k>} = \{-2^k < x \leq 2^k - 1\}$ to field element x' in $\mathbb{Z}_q$ through a simple mod operation f: $x' = f(x) = x \bmod q$. As a result, for any two integers x and y and the operations $\circledcirc \in \{+, -, \cdot\}$, we have $x \circledcirc y = f^{-1}(f(x) \circledcirc f(y))$. We require $q > 2^{k+1}$ since we have 2^k non-negative numbers and 2^k negative numbers in total. For multiplication, we require $q > 2^{2(k+1)}$ to avoid multiplication overflow[12, 13].

As for fixed-point numbers, we encode them as follows: for a signed fixed-point number $x \in \mathbb{Q}_{<k,f>} = \{x = \bar{x} \cdot 2^{-f}, \bar{x} \in \mathbb{Z}_{<k>}\}$, we represent it using the corresponding integer $\bar{x} = x \cdot 2^f$. Therefore, the addition, subtraction and multiplication of fixed-point numbers could be achieved by directly doing computations over their integer representation. The multiplication requires more attention since a f-bit truncation on the result is needed to maintain the precision of the multiplication result. We directly use the truncation protocol introduced in [12] to achieve the goal.

6.3 Secure Comparison over fixed-point Numbers

All non-leaf nodes in the decision tree are represented as secure comparisons. Therefore, we require a general n-party secure comparison protocol for this task. In our solution, the secure comparison protocol can be treated as a plug-and-play module, and many protocols in the literature [12, 13, 37, 41] can be applied here.

We choose to implement the secure comparison protocol in [12] to make our decision tree solution complete. This secure comparison protocol is designed for the general n-party setting. It is a good match to our solution as we derive the fix-point number representation from the same work. Besides, it can be applied against both semi-honest and malicious adversaries. As mentioned in [12], security against a malicious adversary can be achieved through using malicious secure building blocks such as verifiable secret sharing(VSS)[5]. We refer readers to [12] for more details of the protocol.

The benchmark illustrates that the secure comparison protocol becomes the bottleneck as our polynomial evaluation protocol is significantly faster. Indeed, the overall performance could improve significantly once a more efficient secure comparison protocol available in the future. The goal of this work is to provide highly efficient building blocks for secure polynomial evaluation.

6.4 Secure Polynomial Evaluation

Recall that we can solve the decision tree evaluation by solving the polynomial $T(x) = \sum P(i) \cdot inv_i \cdot y_i$. We apply our polynomial evaluation protocol on each term of polynomials then sum them up to form the final $T(x)$. For a complete binary decision tree with depth d, $P(i) \cdot inv_i \cdot y_i$ can be treated as a degree $d+1$ polynomial where inv_i is a constant coefficient, $P(i)$ contains d variables in secret sharing form as results of comparisons, and y_i is the secret shared prediction for the corresponding leaf node. The degrees of all variables above are 1, so we only need to use the protocol introduced in Algorithm 4 and all the terms can be computed in parallel in two rounds. After that, we locally sum the terms up to get the final $T(x)$.

6.5 Prototype Implementation

We developed prototypes for our decision tree evaluation protocols. We choose the Nursery dataset from the UCI Machine Learning Repository [22] as the data source. Nursery dataset has 12960 number of instances, each of which has 8 features. The original dataset consists of only string data so we mapped them to fixed-point numbers. We used sk-learn library to achieve decision tree training. We directly use the trained model and imported it into the MPC codes.

The MPC protocols are implemented in HoneybadgerMPC [30], a secret sharing based MPC framework that supports malicious security. The implementations are written in Python3, parts of the fixed-point algebra codes are from HoneybadgerMPC itself and we extend its code to support the comparison protocol and secure polynomial evaluation protocol. As for the test environment, we deploy our codes in 4 machines with Intel Xeon E5-263040(40 cores) and 384 GB RAM. These machines are connected through LAN and the latency among each other is around 0.1 ms.

Prime q is a 160-bit integer and thus, the field can support the multiplication of 64-bit integers and 32-bit statistical security for fixed-point algebra [13]. While we do not instantiate malicious behaviors in our experiments, our protocol implementations are malicious secure, and the current benchmark includes the verification overhead. We directly use the malicious secure building blocks provided by the HoneybadgerMPC library.

6.6 Experimental Analysis

The original trained model has 385 nodes and achieves 99.6% accuracy. Through tuning the parameters, we are able to get models with different depths, and we extend them to be complete trees to hide the decision tree topology. The experiment result is shown in Table 1. In general it takes around 12 seconds to finish a depth-8 tree evaluation. And it takes around 100 seconds to evaluate a depth-11 tree. As we can see, the performance of our polynomial evaluation protocol takes only around 10% of the whole time, and the secure comparison protocol is the bottleneck. Overall, the performance of the proposed approach can improve significantly when a more efficient comparison protocol is available. As secure comparison protocol is not the main goal of this paper, we omit the further discovery for it.

To the best of our knowledge, our protocol is the first to support decision tree evaluation in a general n-party setting. In Section 6.7, we compare our protocol with state-of-the-art 2-PC protocol [31] and explain the difference between general n-PC protocols and 2-PC protocols.

[5] In our implementation, such malicious secure building blocks are provided by HoneybadgerMPC [30].

Table 1. Decision tree evaluation benchmark using Nursery dataset over 4 parties. (The unit for time is second, and MB for bandwidth)

Tree Depth	Time for Comparison	Time for Polynomial	Total Time	Total Bandwidth
8	11.16	1.20	12.46	3.73
9	22.02	2.72	24.99	7.55
10	44.30	5.69	50.53	15.3
11	87.37	12.67	101.32	30.9

Table 2. Offline cost of polynomial evaluation solution

Tree depth	Time for random generation (s)	Time for actual offline phase (s)	Total (s)
8	1.84	1.85	3.69
9	4.01	3.08	7.09
10	9.08	5.85	14.93
11	19.74	12.17	31.91

* The experiment is run under a 4-party setting with (4,1) Shamir secret sharing.

Introducing Higher Network Latency. Since we run the benchmark on 4 machines in a LAN where the network latency is below one millisecond, We launch another test where we change the communication layer of HoneybadgerMPC and add 100ms latency to simulate real-world settings. The benchmark result demonstrates that the total online time increases around $200 - 300$ms as expected.

Offline Phase Benchmark. We benchmark the offline phase of our protocol and the result is shown in Table 2. The result shows that the cost of the offline phase is acceptable.

Analysis with Another Dataset. To test if the performance is sensitive to the training dataset or resulted models. We test our protocols again using the Boston housing dataset, which is the largest dataset measured in the work of Kiss et al. [43].

The result illustrates that the general performance looks almost the same as the nursery dataset. The reason is that the performance is theoretically independent of the training dataset. Instead, it depends only on the depth of the tree. For a model with depth d, the number of comparisons required is always $2^d - 1$ (the internal nodes) and the number of polynomial evaluation calls is 2^d (leaf nodes).

6.7 Comparison with [31]

In this section, we provide a brief comparison to Ma et al. [31]. While the library used by the authors (emptoolkit) is public, to the best of our knowledge, the authors did not make their benchmarking codes public. Therefore, we only make comparisons based on the benchmarking numbers reported in the paper.

Overall, their benchmarking number suggests that their protocols are more efficient compared to ours, with their protocols taking between 10 and 100 milliseconds for most settings while our protocol takes 10 to 100 seconds.

For a more precise comparison, we look for datasets and settings where our benchmarking is similar to theirs. In terms of tree depth, feature/attribute amount, and the number of nodes, their diabetes dataset seems to be the closest to our nursery dataset, with their decision tree having 17 more depth, 2 more features/attributes, and 4385 more total nodes. Additionally, since we ran our experiment on machines connected through LAN with around 0.1ms latency, we compare with their LAN setting, which also has a latency of 0.1ms.

Table 3 summaries the comparison between our Nursery dataset and the Diabetes dataset from Ma et al. [31].

However, we would like to highlight the key differences between their protocol and ours. First, their protocol is a specialized two-party decision tree evaluation protocol, while our protocol is a general multiparty protocol. Our protocol focuses on situations where multiple servers jointly hold the decision tree, while Ma et al. [31] only extends to two non-colluding cloud servers. Therefore, our protocol and their protocol target completely different use cases. Our protocol is more suitable to the MPC-as-a-service setting, where a group of online servers run MPC protocols to provide a secure decision tree evaluation service. The model holders can secret-share their models to the online servers, then there is no need for them to stay online anymore. However, in most 2PC protocols such as the one of Ma et al., the

Table 3. Comparing Nursery dataset with Diabetes dataset

Dataset	Tree depth	Features	Original Nodes	Padded/Total Non-Leaf Nodes	Online time(ms)
Nursery	11	8	385	2047	101320
Diabetes	28	10	393	6432	< 100

* The Nursery dataset is run over 4 parties.
* The Diabetes dataset is run on LAN (1Gbps/0.1ms) [31]

model holders have to stay online to provide decision tree evaluation service, and in many cases, this is an unrealistic requirement for model holders. As a trade-off, 2PC protocols are often more efficient than ours as the building blocks of 2PC protocols are way more efficient than building blocks of n-PC protocols. Besides, with thresholds such as $t < \frac{n}{3}$, our implementation guarantees robustness and fairness, such that the honest parties always get correct evaluation results, no matter what the active malicious adversary do. This is impossible to achieve in 2 party setting. While Ma et al. [31] extended their protocols to be maliciously secure, their benchmarking was run on the semi-honest protocol while our benchmarking was run on the maliciously secure protocols.

7 Secure Markov Process Evaluation

In this section, we demonstrate how the matrix powering protocol can be used to solve secure Markov process evaluation and we provide a concrete example: credit risk analysis.

7.1 Markov Chain Introduction

Markov chain is a common-used model to describe a process with multiple states where the future state only depends on the current state and not on the past. It is used in many applications to model randomness, ranging from biology to economics. We include all possible states into a state space $\mathbf{S} = \{s_1, s_2, \ldots, s_k\}$. We say a process satisfies Markov property if the probability P_{ij} of state transitions from s_i to s_j only depends only on the current state s_i. We denote x_i to be the state of the process in the time point i, then Markov property can be described by the following equation:

$$P(x_{t+1}|x_1, x_2, \ldots x_t) = P(x_{t+1}|x_t)$$

In this work, we focus on the first-order stationary Markov processes where all probabilities P_{ij} are constant numbers and do not change with time. As a result, we can represent the probabilities with a k-by-k square matrix P, then we can compute $x_i = x_0 \cdot P^i$. This is the equation that we need to solve for Markov process evaluation at the time point i.

7.2 The Markov Chain Application: Credit Risk Analysis

Credit risk analysis is a significant task for banks since the profits and risks of banks are directly linked with the credit quality of their customers, and the credit quality of customers is modeled through the Markov process in many economy works [26, 36]. In these works, a transition matrix is used to explain the transition of creditor quality, and many methods are introduced to generate proper transition matrices given the data of creditors. The transition matrix is valuable in the sense that it can be used to predict the credit quality migration for a new customer who shares similar financial backgrounds. However, there is almost no research about privacy of credit risk analysis.

We observe that privacy can fit the problem of credit risk analysis quite well due to the following reasons: transition matrices are valuable assets since it is trained on sensitive data of creditors and can be used for predictions of new customers. As a result, it is a perfect match that multi-party computation can be applied here so that one bank can train transition matrices, sell matrices to other banks by providing prediction services, and meanwhile keep the transition matrices hidden. And this pattern can be applied to many Markov chain models in fields beyond finance.

Table 4. Decision tree evaluation benchmark using Nursery dataset over 7 parties with (7, 2) secret sharing. (The unit for time is second, and MB for bandwidth)

Tree Depth	Time for Comparison	Time for Polynomial	Total Time	Total Bandwidth
8	11.75	1.30	13.16	4.98
9	23.71	2.81	26.76	10.08
10	45.41	6.17	52.11	20.42
11	89.94	13.32	104.51	41.3

7.3 System Model

We abstract the scenarios into an MPC-as-a-service model. There are three types of parties: model holders who own the transition matrices, servers who collaborate to provide MPC services, and clients who want to get the prediction value given the Markov model. Both the transition matrices and the input of clients are considered private data and should keep hidden from other parties. The data flow is as follows: the model holders (e.g. banks) outsource their transition matrices to the servers through secret sharing, the clients also secret-share their private input to servers so that servers can run MPC protocols to finish the evaluation. We assume servers are fully connected with each other and keep online to provide MPC service. Model holders only need to upload their models to servers then they are not required to be online.

The adversary model is the same as our general setting: we support both semi-honest adversary and malicious adversary by picking different sub-protocols such as Shamir secret sharing and verifiable secret sharing.

7.4 Secure Evaluation of Markov Process

As mentioned, the equation that we need to solve is as follows: $x_i = x_0 \cdot P^i$ where x_0 is the private input from clients in the form of a vector, and P is the trained transition matrix. The core part of the solution is how to efficiently compute matrix powers and we can use our protocol to achieve it. Recall that we can finish the matrix power computation in four rounds with less communication and computation compared to previous works. As a result, we can achieve the evaluation for the Markov process in five rounds, where the first four rounds are used to compute the power of transition matrix and the last round is used to multiply P^i and x_0. Our protocol can be used to compute very high power p due to the fact that the computation complexity is $O(\log p)$.

7.5 Experiments for Evaluation of Markov Process

We build up the prototype using HoneybadgerMPC [30] as the MPC framework. The implementation of matrix multiplication and powering are implemented with both Python and C++. The Python codes are mainly responsible for the operations where communication is required. Batch structures are also applied to guarantee the correct round complexity of the protocol. Due to the heavy local computation included in the protocol, mainly matrix multiplications, We implemented C++ code using NTL library to implement them and used file IO as the connection between Python code and C++ code. This file IO caused some overhead that is only because we used a Python framework for MPC, and this overhead can be eliminated when a C/C++ framework is used, which means the performance could have been better. The test environment is 4 cluster machines, each with Intel Xeon E5-263040 (40 cores) cores and 384GB RAM. The latency between these machines is around 0.1ms. The benchmark result for matrix powering is demonstrated in Table 6. In the use case of credit risk analysis, the transfer matrix is often small, (e.g. a 9×9 matrix is used in [36]) and our benchmark shows that we can solve 10×10 matrix powering within one second. Besides, we observe that the bottleneck of the protocol is the communication time, and the communication time is independent of the power. Thus our performance almost remains unchanged when we compute higher powers. This observation remains true for small matrices.

To make a comparison, we also implement the typical protocol to compute matrix powering where Beaver matrix multiplication is applied $\log p$ times. The benchmark of the typical protocol is done using the same hardware and it turns out that our protocol outperforms the typical protocol by around 3x when e is small (less than 1024). When e is larger our protocol outperforms the typical protocols more as our communication is independent of the power.

Table 5. Offline cost of Matrix powering

Matrix dimension	Time for random generation (s)	Time for actual offline phase (s)	Total (s)
$10 * 10$	0.21	0.53	0.74
$40 * 40$	3.40	1.51	4.91
$80 * 80$	12.79	5.78	18.57

* The experiment is run under a 4-party setting with (4,1) Shamir secret sharing.

Table 6. Benchmark result for proposed Matrix Powering protocol under 4 parties

Dimension	Power	Online time(s)	Bandwidth
10×10	16	0.12	0.04
10×10	1024	0.147	0.04
40×40	16	0.465	0.9
40×40	1024	0.532	0.9
320×320	16	18.552	64.1
320×320	1024	19.367	64.1
320×320	8192	20.848	64.1

* The experiment is run under a 4-party setting with (4,1) Shamir secret sharing.

* Bandwidth is measured through total megabytes(MB) sent per party.

Offline phase cost. We benchmark the offline phase of our protocol and the result is shown in Table 5. The result shows that the cost of the offline phase is acceptable.

7.6 Experiments for Multiplying an Arbitrary Number of Matrices

As mentioned in Section 4.1, we implement both protocols for multiplying an arbitrary number of matrices. The test environment is the same as the experiment for Markov process. The protocols are instantiated on 4 parties with (4,1) Shamir secret sharing. The detailed benchmark result is shown in Table 7. We observe that Beaver matrix multiplication outperforms the protocol of Bar-Ilan and Beaver [2] in all test cases, which confirms our analysis in Section 4.1.

8 Related works and future directions

To the best of our knowledge, our work is the first to achieve efficient high-degree polynomials evaluation with arbitrary numbers of variables. In [25], Ishai and Kushilevitz try to solve the same problem as we do. The method they chose is to represent the high degree polynomials into multiple low degree polynomials with randomness, then solve many low degree polynomials. Regarding polynomial algebra, Mohassel and Franklin [34] works in the setting of directly performing operations on the polynomials, such as polynomial multiplication, division with remainder, polynomial interpolation, polynomial GCD, etc, while our work is primarily focused on evaluating the polynomials. Dachman-Soled et al. [18] work in the setting of evaluating multivariate polynomials where each party holds different variables as private inputs. In our setting, all the variables are shared among all the parties. Cramer and Damgård [16] also focus on operations of linear algebra, and their work aims at solving computation of determinant, characteristic polynomial, rank, and the solution space of linear systems of equations, which is a different set of operations than our paper.

Privacy-preserving decision tree evaluation was firstly proposed in [8]. Kiss et al. [43] systematize the recent privacy-preserving decision tree training/evaluation solutions and in general, these solutions can be divided into two types. The first type rely on homomorphic encryption. For example, Wu et al. [42], Joye and Salehi [27] use additive homomorphic encryption combined with different private comparison protocols to achieve decision tree evaluation. The second type makes use of garbled circuits with efficient private comparison. For example, Tueno et al. [40] represent the decision tree as an array and achieve privacy preserving evaluation through garbled circuits or oblivious RAM. Due to the nature of homomorphic encryption and garbled circuits, the above protocols focus only on two-party setting, where the server has private decision tree model and the client holds the private testing input. Another alternative solution is mentioned in [21], which is also based on secret sharing as we do. It requires full-threshold secret sharing and the comparison protocol is designed for 2 parties, therefore this solution is also limited to 2-party setting. Compared with these works, our work is built on secret sharing and inherently supports the general n-party setting.

Table 7. Benchmark comparison for multiplying multiple matrices between the method of Bar-Ilan and Beaver [2] and Beaver Matrix Multiplication. The experiments is under 4-party setting. Bandwidth is measured through total megabytes sent out per party.

Dimension	Num of matrices	Bar-Ilan and Beaver [2]			Beaver Matrix Multiplication		
		Computation time(s)	Communication time(s)	Bandwidth (MB)	Computation time(s)	Communication time(s)	Bandwidth (MB)
10×10	16	0.07s	0.3	0.5	0.03	0.173	0.17
10×10	64	0.16s	0.86	2.1	0.12	0.71	0.7
20×20	16	0.2s	1.1	2.32	0.07	0.51	0.72
20×20	64	0.82s	2.82	9.07	0.31	0.95	2.99
80×80	16	3.23s	9.07	43.3	0.89	3.06	11.67
80×80	64	20.883s	35.71	169.5	5.82	14.65	49.3

9 Conclusion

To conclude, we propose Polymath, a secure multi-party computation toolkit that supports the evaluation of multi-variate high degree polynomials. The protocols in Polymath have constant round complexity and require less communication than other works. We also demonstrate a complete solution for privacy-preserving decision tree evaluation in general n-party setting, and we make it practical to evaluate high-depth decision tree models and get prediction results with very high accuracy.

References

[1] Adi Akavia, Max Leibovich, Yehezkel S. Resheff, Roey Ron, Moni Shahar, and Margarita Vald. Privacy-preserving decision tree training and prediction against malicious server. Cryptology ePrint Archive, Report 2019/1282, 2019. https://eprint.iacr.org/2019/1282.

[2] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, PODC '89, page 201–209, 1989.

[3] Assi Barak, Martin Hirt, Lior Koskas, and Yehuda Lindell. An end-to-end system for large scale p2p mpc-as-a-service and low-bandwidth mpc for weak participants. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 695–712, 2018.

[4] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991.

[5] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology – CRYPTO'91*, pages 420–432, August 11–15, 1992.

[6] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure mpc with linear communication complexity. In *Theory of Cryptography*, pages 213–230, 2008.

[7] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 1–10, New York, NY, USA, 1988. Association for Computing Machinery.

[8] Justin Brickell, Donald E Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 498–507, 2007.

[9] Owen Brown and David Joseph. Secure digital escrow account transactions system and method, June 5 2003. US Patent App. 10/010,340.

[10] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 88–97, 2002.

[11] John Cartlidge, Nigel P. Smart, and Younes Talibi Alaoui. Mpc joins the dark side. Cryptology ePrint Archive, Report 2018/1045, 2018. https://eprint.iacr.org/2018/1045.

[12] Octavian Catrina and Sebastiaan de Hoogh. Improved primitives for secure multiparty integer computation. In Juan A. Garay and Roberto De Prisco, editors, *Security and Cryptography for Networks*, pages 182–199, 2010.

[13] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In Radu Sion, editor, *Financial Cryptography and Data Security*, pages 35–50, 2010.

[14] European Commission. 2018 reform of eu data protection rules.

[15] Sandro Coretti, Juan Garay, Martin Hirt, and Vassilis Zikas. Constant-round asynchronous multi-party computation based on one-way functions. In *Advances in Cryptology — ASIACRYPT 2016*, page 998–1021, 2016.

[16] Ronald Cramer and Ivan Damgård. Secure distributed linear algebra in a constant number of rounds. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 119–136, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[17] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 316–334. Springer, 2000.

[18] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Secure efficient multiparty computing of multivariate polynomials and applications. In Javier Lopez and Gene Tsudik, editors, *ACNS 11: 9th International Conference on Applied Cryptography and Network Security*, volume 6715 of *Lecture Notes in Computer Science*, pages 130–146, Nerja, Spain, June 7–10, 2011. Springer, Heidelberg, Germany.

[19] I. Damgard, V. Pastro, N.P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/535, 2011. https://eprint.iacr.org/2011/535.

[20] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 285–304, 2006.

[21] I. Damgård, D. Escudero, T. Frederiksen, M. Keller, P. Scholl, and N. Volgushev. New primitives for actively-secure mpc over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1102–1120, 2019.

[22] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[23] Irene Giacomelli, Somesh Jha, Ross Kleiman, David Page, and Kyonghwan Yoon. Privacy-preserving collaborative prediction using random forests. *CoRR*, abs/1811.08695, 2018.

[24] Eric Goldman. An introduction to the california consumer privacy act (ccpa). *Santa Clara Univ. Legal Studies Research Paper*, 2020.

[25] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science*, pages 294–304, 01 2000.

[26] Matthew Jones. Estimating markov transition matrices using proportions data: An application to credit risk. *IMF Working Papers*, 05, 01 2006.

[27] Marc Joye and Fariborz Salehi. Private yet efficient decision tree evaluation. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 243–259. Springer, 2018.

[28] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. Cryptology ePrint Archive, Report 2020/521, 2020. https://eprint.iacr.org/2020/521.

[29] Benjamin Kuykendall, Hugo Krawczyk, and Tal Rabin. Cryptography for #metoo. *Proceedings on Privacy Enhancing Technologies*, 2019(3):409 – 429, 2019.

[30] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 887–903, New York, NY, USA, 2019. Association for Computing Machinery.

[31] Jack P. K. Ma, Raymond K. H. Tai, Yongjun Zhao, and Sherman S.M. Chow. Let's stride blindfolded in a forest: Sublinear multi-client decision trees evaluation. In *ISOC Network and Distributed System Security Symposium – NDSS 2021*. The Internet Society, 2021.

[32] F. Massacci, C. N. Ngo, J. Nie, D. Venturi, and J. Williams. Futuresmex: Secure, distributed futures market exchange. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 335–353, 2018.

[33] P. Mohassel and Y. Zhang. SecureML: a system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38, 2017.

[34] Payman Mohassel and Matthew Franklin. Efficient polynomial operations in the shared-coefficients setting. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 44–57, New York, NY, USA, April 24–26, 2006. Springer, Heidelberg, Germany.

[35] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, page 35–52, New York, NY, USA, 2018. Association for Computing Machinery.

[36] Hadad Muliaman, Wimboh Santoso, Bagus Santoso, Dwityapoetra Besar, and Ita Rulina. Rating migration matrices: empirical evidence in indonesia. *IFC Bulletin*, 01 2009.

[37] Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography – PKC 2007*, pages 343–360, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[38] Rahul Rachuri and Ajith Suresh. Trident: Efficient 4pc framework for privacy preserving machine learning. Cryptology ePrint Archive, Report 2019/1315, 2019. https://eprint.iacr.org/2019/1315.

[39] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[40] Anselme Tueno, Florian Kerschbaum, and Stefan Katzenbeisser. Private evaluation of decision trees using sublinear cost. *Proceedings on Privacy Enhancing Technologies*, 2019(1):266–286, 2019.

[41] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019(3):26 – 49, 2019.

[42] David J. Wu, Tony Feng, Michael Naehrig, and Kristin Lauter. Privately evaluating decision trees and random forests. *Proceedings on Privacy Enhancing Technologies*, 2016(4), 2016.

[43] Ágnes Kiss, Masoud Naderpour, Jian Liu, N. Asokan, and Thomas Schneider. Sok: Modular and efficient private decision tree evaluation. *Proceedings on Privacy Enhancing Technologies*, 2019(2):187 – 208, 2019.

A Proofs

A.1 Lemmas

Proof of Lemma 1. $\forall\lambda, \Pr[x' - r = \lambda] = \Pr[r = x' - \lambda] = \frac{1}{q}$ $\forall\lambda, \Pr[x'' - r = \lambda] = \Pr[r = x'' - \lambda] = \frac{1}{q}$ $\forall\lambda, \Pr[x' - r = \lambda] = \Pr[r = x' - \lambda] = \frac{1}{q} = \Pr[r = x'' - \lambda] = \Pr[x'' - r = \lambda]$ □

Proof of Lemma 2. $\forall\lambda, \Pr[x'r = \lambda] = \Pr[r = x'^{-1}\lambda] = \frac{1}{q}$ $\forall\lambda, \Pr[x''r = \lambda] = \Pr[r = x''^{-1}\lambda] = \frac{1}{q}$ $\forall\lambda, \Pr[x'r = \lambda] = \Pr[r = x'^{-1}\lambda] = \frac{1}{q} = \Pr[r = x''^{-1}\lambda] = \Pr[x''r = \lambda]$ □

Proof of Lemma 3. The number of all n by n matrix with elements from $\mathbb{F}_q$ is q^{nn}. The size of $GL(n,q)$ (the amount of invertible n by n matrix with elements from $\mathbb{F}_q$) is $\Pi_{i=0}^{n-1}(q^n - q^i)$. Therefore, the probability of a matrix chosen uniformly at random being singular (not having an inverse) is $1 - \frac{\Pi_{i=0}^{n-1}(q^n - q^i)}{q^{nn}}$, which is less than $\frac{1}{q} + \frac{1}{q^2}$. Therefore the probability of a matrix chosen uniformly at random being singular (not having an inverse) is less than $\frac{1}{q} + \frac{1}{q^2}$. We still need to prove that $1 - \frac{\Pi_{i=0}^{n-1}(q^n - q^i)}{q^{nn}}$ is less than $\frac{1}{q} + \frac{1}{q^2}$.

If $n = 1$, then the matrix is the same as the field and all matrices are invertible (since all field elements have inverses), and the probability of a random 1 by 1 matrix being singular is $\frac{1}{q}$, which is less than $\frac{1}{q} + \frac{1}{q^2}$.

For $n = 2$:

$\Pi_{i=0}^{n-1}(q^n - q^i) = q^{nn} - q^{nn-1} - q^{nn-2} + q^{nn-3} = q^{nn} - q^{nn-1} - q^{nn-2} + O(q^{nn-3})$

For $n = 3$:

$\Pi_{i=0}^{n-1}(q^n - q^i) = q^{nn} - q^{nn-1} - q^{nn-2} + q^{nn-4} + \Sigma_{i=1}^{nn-4} a_i q^{nn-4-i} = q^{nn} - q^{nn-1} - q^{nn-2} + O(q^{nn-3})$

For $n = 4$:

$\Pi_{i=0}^{n-1}(q^n - q^i) = q^{nn} - q^{nn-1} - q^{nn-2} + 2q^{nn-4} + \Sigma_{i=1}^{nn-4} a_i q^{nn-4-i} = q^{nn} - q^{nn-1} - q^{nn-2} + O(q^{nn-3})$

For $n \geq 5$:

$\Pi_{i=0}^{n-1}(q^n - q^i) = q^{nn} - q^{nn-1} - q^{nn-2} + 2q^{nn-5} + \Sigma_{i=1}^{nn-5} a_i q^{nn-5-i} = q^{nn} - q^{nn-1} - q^{nn-2} + O(q^{nn-3})$

$$\begin{aligned} 1 - \frac{\Pi_{i=0}^{n-1}(q^n - q^i)}{q^{nn}} &= \frac{1}{q} + \frac{q^{nn-2} - O(q^{nn-3})}{q^{nn}} \\ &\leq \frac{1}{q} + \frac{q^{nn-2}}{q^{nn}} \\ &\leq \frac{1}{q} + \frac{1}{q^2} \end{aligned}$$

□

Proof of Lemma 4. $\forall\lambda, \Pr[X'R = \lambda] = \Pr[R = X'^{-1}\lambda] = \frac{1}{|GL(n,q)|}$ $\forall\lambda, \Pr[X''R = \lambda] = \Pr[R = X''^{-1}\lambda] = \frac{1}{|GL(n,q)|}$ $\forall\lambda, \Pr[X'R = \lambda] = \Pr[R = X'^{-1}\lambda] = \frac{1}{|GL(n,q)|} = \Pr[R = X''^{-1}\lambda] = \Pr[X''R = \lambda]$ □

Proof of Lemma 5. Define X_i for $i = 0$ to $i = |GL(n,q)| - 1$ to be all elements in $GL(n,q)$. Define S_i as the multiset where $S_i = \{\forall Y \in GL(n,q), YX_iY^{-1}\}$. $\forall X_j \in Cl(X_i), \exists X_{ij} | X_{ij}X_iX_{ij}^{-1} = X_j$ Define S_{ij} as the multiset where $S_{ij} = \left\{\forall Y \in GL(n,q), YX_{ij}X_iX_{ij}^{-1}Y^{-1}\right\}$. $S_{ij} = \left\{\forall Y \in GL(n,q), YX_{ij}X_iX_{ij}^{-1}Y^{-1}\right\} = \left\{\forall Y \in GL(n,q), (YX_{ij})\, X_i \left(X_{ij}^{-1}Y^{-1}\right)\right\} = \{\forall Y \in GL(n,q), YX_iY^{-1}\} = S_i$ $S_{ij} = \left\{\forall Y \in GL(n,q), YX_{ij}X_iX_{ij}^{-1}Y^{-1}\right\} = \left\{\forall Y \in GL(n,q), Y\left(X_{ij}X_iX_{ij}^{-1}\right)Y^{-1}\right\} = \{\forall Y \in GL(n,q), YX_jY^{-1}\} = S_j$ $\forall i \forall j | X_j \in Cl(X_i), S_i = S_{ij} = S_j$

Define $m_{X_j}(X_i)$ as the number of elements X_a such that $X_aX_jX_a^{-1} = X_i$. We now prove that $\forall i,j,k | X_j \in Cl(X_i) \wedge X_k \in Cl(X_i), m_{X_k}(X_i) = m_{X_k}(X_j)$. Assume $\exists i,j,k | X_i \in Cl(X_j) \wedge m_{X_k}(X_i) > m_{X_k}(X_j)$ $\forall X_a | X_aX_kX_a^{-1} = X_i, X_a^{-1}X_iX_a = X_k$ $\forall X_a | X_aX_kX_a^{-1} = X_j, X_a^{-1}X_jX_a = X_k$ Therefore $m_{X_i}(X_k) > m_{X_j}(X_k)$. By the statement above, $S_k = S_i = S_j$. This is a contradiction. $\forall i,j,k | X_j \in Cl(X_i), m_{X_k}(X_i) = m_{X_k}(X_j)$ $\forall i,j,k | X_j \in Cl(X_i) \wedge X_k \in Cl(X_i), m_{X_k}(X_j) = \frac{|GL(n,q)|}{|Cl(X_i)|}$ $\forall\lambda, \Pr[RXR^{-1} = \lambda] = \Pr[R \in S] = \frac{|S|}{|GL(n,q)|} = \frac{1}{|Cl(X)|}$ where S is a specific subset of $GL(n,q)$ of size $\frac{|GL(n,q)|}{|Cl(X)|}$ □

A.2 Proof of Protocols

A.2.1 Proof for the protocol Tri-variate Term Evaluation

We prove that our protocol reveals no information about x, y, z by showing that given two different values of x, y, z, the probability distributions of possible transcripts remain the same. We define the transcript as all the values an arbitrary party P_i can see during the execution of our protocol. For this protocol, the transcript includes:

- Messages related to the secret sharing of x, y, z, a, b, c
- All other parties' shares of $x - a$
- All other parties' shares of $y - b$

- All other parties' shares of $z - c$
- Messages related to doing multiplication of two variables with traditional Beaver triples

For messages related to doing multiplication of two variables with traditional Beaver triples [5], the security is already proven and well understood. For messages related to the secret sharing of x, y, z, a, b, c, the security is proven by the specific secret sharing scheme. We assume that the secret sharing scheme is information theoretically secure against semi-honest adversaries, such as Shamir's Secret Sharing Scheme [39]. Therefore we ignore those messages in our security analysis. Additionally, given these messages, the party P_i can also calculate the values of some of the secret shared values. Therefore we focus on the following items as the transcript specific to our protocol:

- All other parties' shares of $x - a$
- All other parties' shares of $y - b$
- All other parties' shares of $z - c$
- The value of $x - a$
- The value of $y - b$
- The value of $z - c$

Proof of Theorem 1. By Lemma 1, the value distribution for $x - a$, $y - c$, $z - c$ are identical for different values for x, y, z.

Furthermore, since the probability distributions of $x' - a$, $y' - c$, $z' - c$, and $x'' - a$, $y'' - c$, $z'' - c$, are uniformly random, the probability distributions of the secret shares of $x' - a$, $y' - c$, $z' - c$ are identical to that of $x'' - a$, $y'' - c$, $z'' - c$. □

A.2.2 Proof for the protocol evaluation of degree-1 arbitrary-variate term

We prove that our protocol reveals no information about $x_1, x_2, \ldots x_n$ by showing that given two different sets of values for $x_1, x_2, \ldots x_n$, the probability distributions of possible transcripts remain the same.

We define the transcript as all the values an arbitrary party P_i can see during the execution of our protocol. For this protocol, the transcript includes:

- Messages for Party P_i related to the secret sharing of $x_1, x_2, \ldots x_n$
- Messages for Party P_i related to the secret sharing of $a_1, a_2, \ldots a_n$
- Messages for Party P_i related to the secret sharing of $(a_1 a_2 \ldots a_n)^{-1}$
- All other parties' shares of $x_1 a_1, x_2 a_2, \ldots x_n a_n$
- Messages related to doing multiplication of two hidden variables with traditional Beaver triples

The Proof here is similar to the Proof for the protocol Tri-variate Term Evaluation above.

Proof of Theorem 2. With high probability, none of a_i is 0. By Lemma 2, the value distribution for $x_1 a_1, x_2 a_2, \ldots x_n a_n$ are identical for different values for $x_1, x_2, \ldots x_n$, namely, all universally random. Furthermore, since the probability distributions of $x'_1 a_1$, $x'_2 a_2$, $\ldots$ $x'_n a_n$, and $x''_1 a_1$, $x''_2 a_2$, $\ldots$ $x''_n a_n$, are uniformly random, the probability distributions of the secret shares of $x'_1 a_1$, $x'_2 a_2$, $\ldots$ $x'_n a_n$ are identical to that of $x''_1 a_1$, $x''_2 a_2$, $\ldots$ $x''_n a_n$. □

A.2.3 Proof For the protocol Matrix Powering

By Lemma 3, with high probability, X, R are in $GL(n, q)$ (The General Linear Group, where $GL(n, q)$ is the set of all n by n invertible matrices with elements from a finite field of size q), which means R^{-1} exist with high probability.

Given that We know the conjugacy class of X, we prove that our protocol reveals no additional information about X by showing that given two different values of X, the probability distributions of possible transcripts remain the same. We define the transcript as all the values an arbitrary party P_i can see during the execution of our protocol. For this protocol, the transcript includes:

- Messages related to the secret sharing of X
- Messages related to the secret sharing of R
- Messages related to the secret sharing of R^{-1}
- All other parties' shares of RXR^{-1}
- Messages related to doing multiplication of two secret matrices with Beaver Matrix Multiplication
- Messages related to doing multiplication of three secret matrices

For messages related to doing multiplication of two secret matrices with Beaver matrix multiplication [33], the security is already proven and well understood. For messages related to the secret sharing of X, R, R^{-1}, the security is proven by the specific secret sharing scheme. We assume that the secret sharing scheme is information theoretically secure against semi-honest adversaries, such as Shamir's Secret Sharing Scheme [39].

Therefore, we ignore those messages in our security analysis.

For Messages related to doing multiplication of three secret matrices, we can either use a specialized protocol or simply perform 2 Beaver Matrix Multiplications. Additionally, given these messages, the party P_i can also calculate the values of some of the secret shared values. Therefore we focus on the following items as the transcript specific to our protocol:

- All other parties' shares of RXR^{-1}
- The value of RXR^{-1}

Proof of Theorem 3. By lemma 4, with high probability, X', X'', R, R^{-1} are all in $GL(n,q)$ (The General Linear Group, where $GL(n,q)$ is the set of all n by n invertible matrices with elements from a finite field of size q).

$\forall\lambda, \Pr[RX'R^{-1} = \lambda] = \Pr[R \in S'] = \frac{|S'|}{|GL(n,q)|} = \frac{1}{|Cl(X')|}$ where S' is a specific subset of $GL(n,q)$ of size $\frac{|GL(n,q)|}{|Cl(X')|}$ by Lemma 5.

$\forall\lambda, \Pr[RX''R^{-1} = \lambda] = \Pr[R \in S''] = \frac{1}{|Cl(X'')|}$ where S'' is a specific subset of $GL(n,q)$ of size $\frac{|GL(n,q)|}{|Cl(X'')|}$.

$\forall\lambda, \Pr[RX'R^{-1} = \lambda] = \Pr[R \in S'] = \frac{1}{|Cl(X')|} = \frac{1}{|Cl(X'')|} = \Pr[R \in S''] = \Pr[RX'R^{-1} = \lambda]$ since X' and X'' belong to the same conjugacy class so $|Cl(X')| = |Cl(X'')|$. □