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

NDSS 2021 Fall Submission #483

***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. Improving the efficiency of these MPC protocols by identifying tailored optimizations within specific computation tasks, while possible, is not scalable for the growing demand for MPC. Ad hoc protocol design, on the other hand, has historically been riddled with latent security vulnerabilities. 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 (multivariate) 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 library over the HoneyBadgerMPC framework and apply it to two prominent secure computation tasks: privacy-preserving inference of decision trees, and privacy-preserving evaluation of the Markov process. For the decision tree inference problem, we demonstrate the feasibility of evaluating high-depth decision tree models in a general n-party setting, where we outperform previous works on both accuracy and performance. For the Markov process application, we demonstrate that Polymath can compute large powers of transition matrices with better online time and less communication.

I. Introduction

Secure multi-party computation (MPC) [3], [7], [27] 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. 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 training and inference [29], [31], [33], [35], privacy-preserving financial solutions [11], [28], and secure information escrows [9], [26].

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], [26], [29], [36]. 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 only 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 and CCPA) 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 scalable. 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, similar to the work of Bar-Ilan and Beaver [2]. We clarify that, in the sequel, a *round* refers to a step in the interactive protocol where every pair of the parties exchanges a message. This class of high-level algebraic primitive finds widespread applications ranging from approximating non-linear functions (for example, the Sigmoid function) to decision trees [21] to time-series analysis [24]. Moreover, beyond polynomials over scalar values, polynomials of matrices (or, equivalently, matrix polynomials)[1] is also an excellent abstraction with applicability to privacy-preserving Markov processes and neural network training.

A. 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

[1]Note that we do *not* intend to use "polynomial matrix," or "matrix of polynomials." Our focus instead is polynomials of matrices or matrix polynomials. That is, this paper considers the secure evaluation of $p(X_1, \ldots, X_k)$, where $p(\cdots)$ is a multivariate polynomial, and $X_1, \ldots, X_k$ are either scalars or matrices.

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. We specifically focus on the following functionalities.

(i) 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. [19].

(ii) 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 its communication complexity is independent of the exponent. For multivariate polynomials over matrices, we evaluate two alternatives for securely multiplying an arbitrary number of matrices. Using these protocols, we can securely evaluate terms of a matrix polynomial in parallel offering trade-off in terms of communication complexity and round complexity.

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.

As discussed earlier, there are several privacy-preserving applications where our Polymath protocol suite can be employed towards reducing the latency utilizing prevailing multiprocessor computing parties. In this work, we consider two representative tasks to demonstrate the applicability of protocols. We apply our multi-variate polynomial evaluation over scalars for privacy-preserving decision tree evaluation and our secure matrix exponentiation protocol for secure credit risk analysis through a Markov process.

(a) Privacy-Preserving Decision Tree Evaluation. The decision tree is a prominent model used in machine learning with applications to various classification problems. However, previous works all focus on low-degree polynomials due to the performance bottlenecks. Using our polynomial evaluation protocols, we present a solution for privacy-preserving decision tree evaluation, and it illustrates that we can achieve very-high accuracy prediction within a reasonable time. To the best of our knowledge, our solution is the first to support general n-parties and high-depth tree evaluations simultaneously.

Polymath can be employed with any arithmetic-circuit MPC library such as SPDZ [25], or Scale MAMBA [18]. We choose the HoneyBadgerMPC library [27] for robust execution in the asynchronous setting. In our experiment, we use depth 13 trees to achieve 99.6% accuracy for the Nursery dataset from the UCI Machine Learning Repository [20] and other works mainly focus on two-party setting and tree models with depth at most 8. The solution consists of representing decision trees with polynomials, our polynomial evaluation protocol, and our 2-round secure comparison protocols. Our benchmark demonstrates that we can evaluate a depth 13 decision tree model with 16383 nodes in around 75 seconds. For a depth 8 complete decision tree model, we can evaluate it within 2 seconds.

(b) Secure Credit Risk Analysis through a Markov Process. We use our matrix powering protocol to solve the evolution of a Markov process. Furthermore, we show how one can use this computation to perform credit risk analysis in financial domains. We outperform the previous works by a constant round complexity while using lower communication and computation. The benchmark shows that in the 4-party setting where at most one malicious party is tolerated, 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 II we present the general system setting and adversary setting for Polymath. In Section III, we introduce the protocols for polynomial evaluation over scalars. In Section IV, we demonstrate the protocols dealing with polynomial over matrices. We provide a security analysis for protocols over both scalars and matrices in Section V. Then in Section VI and Section VII, we illustrate two applications of Polymath: privacy-preserving decision evaluation and privacy-preserving Markov process evaluation. Both applications are followed by detailed benchmark. Then we discuss related works in Section VIII and conclude the paper in Section IX.

II. Problem Setting

A. System Model

We consider a standard MPC setting with a set of parties $P_1, P_2, \ldots P_n$ 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.

B. 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 [25]), the employed scheme is additive secret sharing, while we consider Shamir secret sharing [34] for the honest majority setting. Against malicious adversary, the secret sharing choice can be verifiable secret sharing [10] or authenticated secret sharing [25].

As an illustrative example, we consider Shamir secret sharing [34] in our applications. An (n, t) Shamir secret sharing scheme, with $n > t \geq 0$ allows n parties $\{P_1, \ldots, P_n\}$ 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 are encoded into the constant coefficient of a degree-t polynomial over $F_q[x]$. 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 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-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 the offline phase can be run for a long time before the online phase. For the offline phase, many state of the art offline protocols [6], [14] 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 open $x - a$ 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.

1 $[x - a]$ = $[x]$ - $[a]$
2 $[y - b]$ = $[y]$ - $[b]$
3 $(x - a) =$ **Open**$([x - a])$ `// Round 1`
4 $(y - b) =$ **Open**$([y - b])$ `// Round 1`
5 **return** $(x - a)(y - b) + (y - b)[a] + (x - a)[b] + [ab]$

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 s. **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 $[y]$. We use **Pow**($[x]$,e) to compute x^e where x is a field element, when x is a secret share, we use the method introduced in Section III-A. We use capital letters (e.g. X) to represent matrices and the protocols like **Open** and **Mul** also apply to matrices.

III. Secure Computations of Polynomial over Scalars

A. 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 can be reduced into computing the power of the secret share $[x]$.

Damgård et al. [19] 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 $(ax)^n[a^{-n}]$.

With this protocol, 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.

[2] A linear secret sharing [16] 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 a linear function.

[3] Against 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 agnostic to those.

Algorithm 2: Univariate Polynomial Evaluation

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_is are random values.

for $i \leftarrow 1$ *to* n **do**
  $[x^i]$ = **Pow**($[x]$, i) // Round 1 & 2
for $i \leftarrow 0$ *to* n **do**
  $P_i([x^i])$ = $[x^i] \cdot a_i$
return $P([x]) = \sum P_i([x^i])$

The above secure univariate polynomial techniques does not extend to the mutli-variate setting. Next, we solve this generalized problem: secure multi-variate polynomial evaluation.

B. 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.

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 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 can be reduced 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 number of needed 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.

The proposed method implies that the idea of Beaver multiplication could be extended to more variables. However, the computation and the number of required pre-computed values increase exponentially with the number of variables. As a result, we need a better way 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).

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.

$[x-a]$ = $[x]$ - $[a]$
$[y-b]$ = $[y]$ - $[b]$
$[z-c]$ = $[z]$ - $[c]$
$(x-a) =$ **Open**($[x-a]$) // Round 1
$(y-b) =$ **Open**($[y-b]$) // Round 1
$(z-c) =$ **Open**($[z-c]$) // Round 1
$[xc]$ = **Mul**($[x]$,$[c]$) // Round 1
$[bz]$ = **Mul**($[b]$,$[z]$) // Round 1
$[ay]$ = **Mul**($[a]$,$[y]$) // Round 1
return $(x-a)(y-b)(z-c) + [xc](y-b) + [bz](x-a) + [ay](z-c) - [abc]$

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: $x_1 a_1 x_2 a_2 \ldots x_n a_n [(a_1 a_2 \ldots a_n)^{-1}]$.

The round complexity of the protocol is 2, regardless of the number of variables. 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 : $[x_1], [x_2], \ldots, [x_n]$
Output : $[x_1 x_2 \ldots x_n]$
Pre-computation: $[r_1], [r_2], \ldots, [r_n]$ and $[(r_1 r_2 \ldots r_n)^{-1}]$ where r_is are random values. n Beaver triples.

for $i \leftarrow 1$ *to* n **do**
  $[x_i r_i]$ = **Mul**($[x_i]$, $[r_i]$) // Round 1
for $i \leftarrow 1$ *to* n **do**
  $x_i r_i$ = **Open**($[x_i r_i]$) // Round 2
return $x_1 r_1 x_2 r_2 \ldots x_n r_n \cdot [(r_1 r_2 \ldots r_n)^{-1}]$

3) Efficient Way to Calculate $x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}$: To solve the problem of polynomial evaluation, we need to deal with terms

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_is are random values. n Beaver triples..

for $i \leftarrow 1$ *to* n **do**
  $[x_i r_i]$ = **Mul**($[x_i]$,$[r_i]$) `// Round 1`
for $i \leftarrow 1$ *to* n **do**
  $x_i r_i$ = **Open**($[x_i r_i]$) `// Round 2`
for $i \leftarrow 1$ *to* n **do**
  y_i = **Pow**($[x_i r_i]$, e_i) `// local computation`
return $y_1 y_2 \dots y_n \cdot [(r_1 r_2 \dots r_n)^{-1}]$

with the most generalized form: $x_1^{i_1} x_2^{i_2} \dots x_n^{i_n}$. We extent the idea in previous sections and generalize it to solve the problem. In our protocol we require the pre-computed values in the form of $[r_1]$, $[r_2]$, ..., $[r_n]$, and $[(r_1^{i_1} r_2^{i_2} \dots r_n^{i_n})^{-1}]$. 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 $(x_1 r_1)^{i_1} \dots (x_n r_n)^{i_n} [(r_1^{i_1} r_2^{i_2} \dots r_n^{i_n})^{-1}]$.

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.

C. Offline Phase

For each polynomial term $x_1^{i_1} x_2^{i_2} \dots x_n^{i_n}$, the required pre-computed values are $[r_1], \dots, [r_n]$, and $[r_1^{-i_1} r_2^{-i_2} \dots r_n^{-i_n}]$. The straight forward way to generate these values is to use beaver triple multiplications to calculate powers then to multiply all random terms together. While techniques introduced in [19], [27] can be applied here to speed up the computation of $[r_i^{i_i}]$.

D. Optimization on Communication Complexity

It can be observed that in our protocols there are many cases where the input $[x]$ is masked by a random share $[r]$ through multiplication $[xr]$. The simple way to multiply $[x]$ and $[r]$ is through Beaver's triple multiplication that takes one round and two reconstructions. We propose that the communication complexity could be improved due to the fact that $[r]$ is a precomputed share instead of a random multiplicand.

The protocol proceeds as follows:

- Offline Phase: All parties generate random share $[r]$, $[a]$ and $[ar]$.
- Online Phase: Parties calculate $[x - a]$ locally, reconstruct $(x - a)$, then $[xr] = (x - a)[r] + [ar]$.

This protocol also takes one round, however, it only requires one reconstruction, thus saving the communication by half. This method works not only on single element $[x]$ but also on matrix $[X]$.

IV. Secure Computation of Polynomials of Matrices

In this section, we discuss polynomial evaluation over matrices with each cell contents a finite field element. Although it is conceptually similar to the scalar polynomial evaluation, the protocols for matrices are inherently different due to the fact that matrix multiplication is not commutative.

A. Beaver triple techniques for matrix multiplication

Mohassel and Zhang [29] propose a generalized version of Beaver triple technique to deal with the multiplication of two 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.

1) How to multiply 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 $X_1, X_2, \dots . X_n$ (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 $R_0, R_1, R_2, \dots . R_n$, which have the same dimensions as X_is and contain random values, and their corresponding inverse matrices $R_0^{-1}, R_1^{-1}, R_2^{-1}, \dots . R_n^{-1}$.

The general idea is that for each input matrix X_i, we mask it by multiplying $R_{i-1}^{-1} X_i R_i$. To multiply three matrices, we apply Beaver matrix multiplication twice. All $R_{i-1}^{-1} X_i R_i$ can be multiplied in parallel. After masking we reconstruct all the results then multiply them locally. As a result, all the random matrices in the middle cancel each other and we get $R_0^{-1} X_1 X_2 \dots X_n R_n$. To get the final results, we multiply it with secret shares $[R_0]$ and $[R_n^{-1}]$ to cancel the randomness on the sides. A formal description of the protocol could be found in Algorithm 6.

The round complexity of the protocol is four. The first two rounds are used to do multiplications for all X_is and the third round is to open them. The final round is used to multiply $[R_0]$, $R_0^{-1} X_1 X_2 \dots X_n R_n$, and $[R_n^{-1}]$. The protocol also works well when X_is are not square matrices and have different dimensions. Parties only need to generate the precomputed random matrices R_i accordingly so that dimensions of matrices match each other.

The other alternative to multiplying an arbitrary number of matrices 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 VII-F. The result demonstrates that the second method outperforms the method of [2] 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 6: Multiply an arbitrary number of matrices [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**
  $[R_{i-1}X_i]$ = **Mul**($[R_{i-1}]$,$[X_i]$)
  $[R_{i-1}X_iR_i^{-1}]$ = **Mul**($[R_{i-1}X_i]$,$[R_i^{-1}]$)
for $i \leftarrow 1$ *to* n **do**
  $R_{i-1}X_iR_i^{-1}$ = **Open**($[R_{i-1}X_iR_i^{-1}]$)
$R_0X_1 \ldots X_nR_n^{-1} = \sum_{i=1}^{n} R_{i-1}X_iR_i^{-1}$
return Mul($[R_0^{-1}] \cdot R_0X_1 \ldots X_nR_n^{-1}$, $[R_n]$)

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 = **Open**($[Y]$) `// Round 3`
return Mul($[R^{-1}] \cdot Y^e$, $[R]$) `// Round 4`

2) How to calculate 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 a lot of precomputation and communication especially when computing a large power. However, this method only achieves a lower level of privacy since some knowledge about the input matrix is leaked (for example, the rank of the input matrix). Please refer to Section V for a detailed analysis regarding the privacy loss. We find this method meaningful as in many use cases it is affordable to leak that information and a lot of computation could be saved.

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 $\log p$ times to get $[X^p]$. As a result, it requires $\log(p)$ rounds involving opening $2 \cdot \log(p)$ 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. so our protocol requires $9+\log(p)$ local matrix multiplications and 18 additions. and state of the art protocol needs $3 \cdot \log(p)$ number of multiplications and $6 \cdot \log(p)$ additions.

B. Offline Phase

For matrix powering protocol, the required precomputation is in the form of A, A^{-1}, basically a matrix with random elements and its inverse matrix. It serves as masks when we multiply an arbitrary number of matrices at the same time, or calculating powers of matrices. 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.

$[XR] = \textbf{Mul}([X], [R])$ `// Round 1`
XR = **Open**($[XR]$) `// Round 2`
$R^{-1}X^{-1} = (XR)^{-1}$
return $[R]R^{-1}X^{-1}$

The above protocol is efficient but there is a very small probability of failure since XR may not have an inverse. Bar-Ilan and Beaver [2] also propose a protocol to generate a guaranteed random invertible matrix $[R]$. Which is summarized in Algorithm 9. We can also easily modify Algorithm 9 to test whether a secret matrix $[M]$ is invertible or not. Throughout our protocols, if we need to generate a random invertible matrix or we need to verify that our input matrix is invertible, we can use this algorithm. However, by Lemma 6, the probability of a random matrix being not invertible is less than $\frac{1}{q} - \frac{1}{q^2}$ where q is the size of the field. Therefore, our protocol should succeed with probability greater than $1 - \frac{1}{q} + \frac{1}{q^2}$, and thus this algorithm for verifying that a matrix is invertible may not be necessary.

V. Security Analysis

The correctness for these protocols are trivial from the protocol description and we do not include a proof here.

In this section, we analyze of 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.

Algorithm 9: Generating Random Invertible Matrices

Input : n
Output : $[R]$

Generate n random matrices $[R_1], [R_2], \dots [R_n]$ and n^2 random matrices $[S_{11}], \dots [S_{nn}]$
$[T_{ij}] = \textbf{Mul}([R_i], [S_ij])$ // Round 1
T_{ij} = **Open**($[T_{ij}]$) // Round 2
Locally check if each T_{ij} is invertible
If no T_{ij} is invertible, rerun the protocol. Otherwise output the first $[R_i]$ such that there exists a T_{ij} that is invertible

A. Intermediate Lemmas

We first provide a list of Lemmas, whose proofs are available in Appendix A.

Lemma 1. *Define $\mathbb{F}_q$ as an arbitrary field of size q. Let x', x'' be arbitrary fields element from $\mathbb{F}_q$ and r be an independent random field element chosen uniformly at random from the field $\mathbb{F}_q$, then the distribution of $x' - r$ and $x'' - r$ are identical, and therefore indistinguishable, from each other.*

Lemma 2. *Define $\mathbb{F}_q$ as an arbitrary field of size q. Let R be a matrix chosen uniformly at random from the set of all n by n matrix where each element is a field element from $\mathbb{F}_q$. The probably that R does not have an inverse (R is singular) is less than $\frac{1}{q} - \frac{1}{q^2}$.*

Lemma 3. *Let X', X'' be arbitrary elements from $GL(n, q)$ (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). Let R be an independent random field element chosen uniformly at random from the field $GL(n, q)$. Then the distribution of $X'R$ and $X''R$ are identical, and therefore indistinguishable, from each other.*

Lemma 4. *Let R be a random matrix chosen uniformly at random from $GL(n, q)$ (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). Let X be an arbitrary matrix in $GL(n, q)$. $\forall \lambda, \Pr[RXR^{-1} = \lambda] = \Pr[R \in S] = \frac{|S|}{|GL(n,q)|} = \frac{1}{|Cl(X)|}$ where S is a specific subset of $GL(n, q)$ of size $\frac{|GL(n,q)|}{|Cl(X)|}$ ($Cl(X)$ is the conjugacy class of X).*

B. Analysis regarding multi-variable polynomials

1) Secure analysis for the protocol Mul-triple-term: 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 $x = x'', y = y'', z = z''$.*

The full proof is available in Appendix B1.

2) Secure analysis for the protocol Mul-arbitrary-term: Intuitively, since each a_i are chosen uniformly at random, each $x_i a_i$ is indistinguishable from random values. Therefore, we leak no information by revealing $x_i a_i$.

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 $x_1 a_1, x_2 a_2, \dots x_n a_n$. We prove that given different values of $x_1, x_2, \dots x_n$, the probability distribution of the transcript remains the same.

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

The full proof is available in Appendix B2.

3) Secure analysis for the protocol Mul-poly-term: The security of calculating polynomial terms is the combination of the security for the secret powering protocol in [19] and the security for calculating multi-variable multiplication. We refer readers to [19] for more details about the security of the secret powering protocol.

C. Analysis Regarding Secure Computation of Matrix Powering

1) Secure 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. However, since R and R^{-1} is otherwise random, we only 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 $X = X''$.

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 RXR^{-1}. We prove that given different values of X, the probability distribution of the transcript remains the same.

Theorem 3. *Let X', X'' be arbitrary values under the constraint that they belong to the same conjugacy class. The probability distribution of RXR^{-1} is identical when $X = X'$ and when $X = X''$.*

Proof: 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 6.

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 8.

$\forall\lambda, \Pr[RX''R^{-1} = \lambda] = \Pr[R \in S''] = \frac{1}{|Cl(X'')|}$ 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 $|Cl(X')| = |Cl(X'')|$. ■

The full proof is available in Appendix B3.

VI. Privacy Preserving Decision Tree Evaluation

In recent years, machine learning is applied in many areas where both the user data and the trained model are considered as sensitive data(e.g. health information, financial information). Thus privacy-preserving machine learning becomes more and more prominent and many works [1], [21], [35], [36] 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.

A. 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 services 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.

B. Technical Overview

To give a general view of our protocol, we represent the trained decision tree models as two kinds of nodes: leaf nodes and non-leaf nodes. The non-leaf node contains a comparison operation so that different branches will be chosen based on the comparison. The leaf nodes store the result of predictions and inspired by [21], we leverage polynomials to represent the leaf node where the degree of the polynomial equals the length of the path between the leaf node and the root. In previous works, evaluating high degree polynomials is treated as a hard problem so most works only focus on decision trees with low depth(e.g. [21] chose the max depth to be 8). However, with our protocols we are able to solve high-degree polynomial evaluation more efficiently, thus giving us the advantages that we can evaluate higher depth trees to get better accuracy.

As a result, we need to solve two problems in a privacy-preserving manner: secure comparison and secure polynomial evaluation. For secure comparison, our method is inspired by [33] and we extend their 4-party private comparison protocol so that it can be used in a general n-party setting. For secure polynomial evaluation, we use the protocol introduced in Section III.

Secure decision tree forest evaluation can also be achieved by combining our solution with secure multi-party aggregation protocols [8].

C. Decision Tree Representation

Decision tree representation with polynomials. Giacomelli et al. [21] describe how a tree can be represented by polynomials. That representation is a perfect match with our protocols described in Section III so we choose to use it in our solution. We give a brief introduction here and we refer readers to [21] 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 $x = (x[1], x[2], \ldots, x[n])$ 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 leaf 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 numbers 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 a bunch of 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 signed integer $x \in \mathbb{Z}_{<k>} = \{-2^{k-1} \leq x \leq 2^{k-1}-1\}$ to field element x' in $\mathbb{Z}_q$ through a simple mod operation f: $x' = f(x) = x \mod q$. As a result, for any two integers x and y and the operations $\circledcirc \in \{+, -, \cdot\}$, we have $x \circledcirc y = f^{-1}(f(x) \circledcirc f(y))$.

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. Besides, we require $q > 2^{2k}$ to avoid multiplication overflow.

D. Secure Comparison over fixed-point Numbers

All non-leaf nodes are represented as secure comparisons. There are many comparison protocols introduced in recent MPC works [12], [33], [35]. In our solution, we extend the protocol in [33], which is a 4-party secure comparison protocol, into a multi-party version to achieve our goal. We focus on the less-than-zero protocol and with our data representation, it is equivalent to finding the sign bit(most significant bit) of a secret share. The general idea is that we prepare a random share $[r]$ and its most significant bit $[r_{msb}]$ in the offline phase. In the online phase, we multiply input $[x]$ and $[r]$, open the result and get the sign bit $b = sign(xr)$. Then the comparison result could be written as $b \oplus [r_{msb}]$. The protocol takes only two rounds for multiplication and opening. In decision tree evaluation, we can run all comparisons in parallel so it also takes only two rounds to deal with all non-leaf nodes. The protocol is formally summarized in Algorithm 10.

Algorithm 10: Secure Comparison LTZ($[x]$)

Input : $[x]$, public parameter $\mathbb{Z}_p$ where $x \in \mathbb{Z}_p$
Output : $[b]$ where $b = 1$ if $x < 0$. $b = 0$ otherwise
Pre-computation: $[r]$ and $[r_{msb}]$ where r is a random value and r_{msb} is the most significant bit(sign bit) of r

$[xr] =$ **Mul**$([x], [r])$ `// Round 1`
$xr =$ **Open**$([xr])$ `// Round 2`
if $xr > p/2$ **then**
    msb = 1
else
    msb = 0
return $msb + [r_{msb}] - 2 \cdot msb \cdot [r_{msb}]$

E. 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 $T(x)$.

F. Discussions

Complexity of our protocols. Our secure comparison protocol takes two rounds where one round is to multiply $[x]$ and $[r]$ and the other is to open the product. The communication complexity is $O(1)$ that only requires 3 invocations of multicast. The xor operation is local since one operand is in cleartext. For a complete decision tree with depth d, there are in total $2^d - 1$ non-leaf nodes so the same amount of comparisons are required. All of them can be run in parallel so the round complexity remains the same and communication complexity increase linearly with the number of non-leaf nodes($O(2^d)$ for complete trees).

The polynomial evaluation protocol also takes two rounds. The first round is to multiply all $[x_i]$ and $[r_i]$ in parallel and the other round is to open all the products. The communication complexity is $O(n)$ where n is the number of variables. For a complete decision tree with depth d, there are in total 2^d leaf nodes and each node is a degree$d+1$ polynomial. So the overall communication complexity is $O(d \cdot 2^d)$ for complete trees.

Complete decision tree vs Non-complete decision tree. Due to the fact that the topology of the trained decision tree model may leak some information about the model itself and the training data, we only consider the complete binary tree in our discussion. However, we want to talk about the overhead introduced by the complete tree and the trade-off between performance and privacy. The observation is that the number of nodes can vary significantly between the original model and the complete tree model by adding dummy nodes. For instance, we trained the decision tree model for the Nursery dataset from the UCI Machine Learning Repository [20] using the sk-learn python library, and the resulted decision tree has depth 13 and the number of nodes is 385, which illustrates that the tree is pretty sparse. And as a comparison, a complete binary decision tree with depth 13 has in total 16383 nodes. In the experimentation, we observe that the running time of the protocol is mostly linear with the number of tree nodes because that the bottleneck of the protocol is computation and communication which both increase linearly with the number of nodes. So we can get a 50x performance improvement if we can use the origin model instead of the complete tree model, and that means we can finish the evaluation with around a second.

Wu et al. [36] also discuss the performance difference between complete trees and "sparse trees" where the number of nodes increases linearly with the depth. The number of nodes

in sparse trees actually match the number of nodes in our origin trained model. And it can be approximated that if using a sparse tree, we can evaluate a decision tree with depth 20 within a minute.

To conclude, there is a trade-off between privacy and performance. If it is tolerable to reveal the structure of the tree, a huge performance improvement can be achieved. And this effect is more obvious when the depth of the tree is higher. It is also an interesting problem that if we can find a balance point by adding dummy nodes to the original model but not completing the tree. We leave this question as our future work to discover.

G. Prototype Implementation

We developed prototypes for our decision tree evaluation protocols. We choose the Nursery dataset from the UCI Machine Learning Repository [20] as the data source. Nursery dataset has 12960 number of instances, each of which has 8 features. The origin 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 [27], 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 our 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 384GB RAM. These machines are connected through LAN and the latency among each other is around 0.1 ms.

H. Experiments

The original trained model has 385 nodes and achieves 99.6% accuracy. We extended the model to be a complete binary tree by adding dummy nodes and it resulted in a depth-13 tree with 16383 nodes. Our experiments were based on this complete tree model to keep the best accuracy and the best privacy. The result showed that it took around 75 seconds to finish a secure evaluation, and this the best performance we know so far in evaluating high-depth complete tree models. We also discovered the relationship between the online running time and the depth of the tree by adjusting parameters in sk-learn to generate models with specific depth, and we summarize the performance in Table I.

Compared with previous works, our result for depth-13 tree is around 3x better than the result shown in [36], and their result is based on two-party setting while our result is in 4-party setting tolerating at most one malicious party. If considering sparse tree structures, our online time for the evaluation of depth-13 sparse tree is around 15 seconds while the results in [36] for the same setting is a few minutes. The benchmark in [21] only considers depth-8 trees in the two-party setting and their performance is dependent on the number of features. Instead, our online performance depends only on the number of nodes, regardless of the number of features. Thus this benchmark successfully demonstrates that our protocol achieves better accuracy and performance at the same time compared with previous works. Besides, our protocol could be extended to more parties while most previous works only support the two-party setting.

TABLE I. BENCHMARK RESULT FOR DECISION TREE EVALUATION

Tree depth	Accuracy	Online time	Bandwidth
8	95.0%	1.57s	0.7MB
9	96.9%	3.43s	1.6MB
10	97.7%	7.52s	3.4MB
11	99.0%	15.71s	7.5MB
12	99.4%	34.4s	16.3MB
13	99.6%	75.6s	32.2MB

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

TABLE II. BENCHMARK RESULT FOR DECISION TREE EVALUATION OVER 7 PARTIES

Tree depth	Accuracy	Online time	Bandwidth
8	95.0%	3.1s	1.0MB
9	96.9%	4.94s	2.16MB
10	97.7%	8.781s	4.69MB
11	99.0%	17.75s	10.1MB
12	99.4%	37.04s	21.95MB
13	99.6%	79.94s	47.06MB

* The experiment is run under a 7-party setting with (7,2) Shamir secret sharing.

Analysis with More Parties. To evaluate the scalability of our solution, we repeat the experiment on 7 machines where the machines have the same hardware as before. We pick (7,2) secret sharing to tolerate at most two malicious parties in this setting. The benchmark result is available at Table II. Adding more parties does not influence the local computation time, and only slightly increases the communication time since there are more data to send. So it is fair to say our method scales to a larger number of parties quite well.

Analyzing Network Latency. Due to the fact that 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. our protocol is expected to perform better than previous works when network latency is high because of the constant round complexity.

VII. SECURE MARKOV PROCESS EVALUATION

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

A. 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 the 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:

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

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.

B. 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 [23], [32]. In these works, a transition matrix is used to explain the transition of creditor quality and many methods are introduces to generate proper transition matrix 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 are almost no researches about privacy on credit risk analysis.

In this work, 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 by 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.

C. 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.

D. 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 $O(\log p)$.

E. Experiments for Evaluation of Markov Process

We build up the prototype using HoneybadgerMPC [27] 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 memories. The latency between these machines is around 0.1ms. The benchmark result for matrix powering is demonstrated in Table III. In the use case of credit risk analysis, the transfer matrix is often small, (e.g. a 9×9 matrix is used in credit2) 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 regardless of the power. Thus our performance almost remains unchanged when we compute higher powers. This observation is true for small matrices and computation becomes the bottleneck for matrices with dimensions larger than 600×600, while such matrices are beyond our use cases.

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 p is small (less than 1024). When p is larger our protocol outperforms the typical protocols more as our communication is regardless of the power.

F. Experiments for Multiplying an Arbitrary Number of Matrices

As mentioned in Section IV-A1, 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 IV and Table V.

TABLE III. BENCHMARK RESULT FOR MATRIX POWERING

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

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

TABLE IV. BENCHMARK FOR MULTIPLYING MULTIPLE MATRICES FOLLOWING BAR-ILAN AND BEAVER [2]

Dimension	Num of matrices	Computation time	Communication time	Bandwidth
10×10	16	0.07s	0.3s	0.5MB
10×10	64	0.16s	0.86s	2.1MB
20×20	16	0.2s	1.1s	2.32MB
20×20	64	0.82s	2.82s	9.07MB
80×80	16	3.23s	9.07s	43.3MB
80×80	64	20.883s	35.71s	169.5MB

* Bandwidth is measured through total megabytes sent per party.

VIII. RELATED WORKS AND FUTURE DIRECTIONS

Our work mainly focuses on two fields: polynomial evaluation and matrix powering. For the secure polynomial evaluation, to the best of our knowledge, our work is the first to achieve efficient high degree polynomials evaluation with arbitrary numbers of variables. In [22], Ishai and Kushilevitz try to solve the same problem as we do, namely to solve high degree polynomials. 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 [30] 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. [17] work in the setting of evaluating multivariate polynomials where each party holds different variables as private inputs while our work focuses on evaluating polynomials in general.

As for the decision tree evaluation, Kiss et al. [37] systematize the decision tree evaluation solutions based on garbled circuits and homomorphic encryptions. And our work focuses on secret sharing based MPC. Giacomelli et al. [21] represents decision trees with polynomials. Both [21] and [36] used homomorphic encryption to achieve 2-party secure decision tree evaluation. Compared with their works, we achieved general n-party protocol and the depth of decision tree in our experiment is much higher than [21] and [36] which means orders of magnitude more workload, and we achieve better performance meanwhile.

TABLE V. BENCHMARK FOR MULTIPLYING MULTIPLE MATRICES THROUGH $\log n$ ROUND BEAVER MATRIX MULTIPLICATION

Dimension	Num of matrices	Computation time	Communication time	Bandwidth
10×10	16	0.03s	0.173s	0.17MB
10×10	64	0.12s	0.71s	0.7MB
20×20	16	0.07s	0.51s	0.72MB
20×20	64	0.31s	0.95s	2.99MB
80×80	16	0.89s	3.06s	11.67MB
80×80	64	5.82s	14.65s	49.3MB

There are some recent works focusing on the secure linear algebra for matrices. The study of matrix operations is mostly inspired by privacy-preserving machine learning where matrix multiplication is treated as a key operation. Mohassel and Zhang [29] were the first to generalize beaver triple ideas on matrices so that matrix multiplication could be computed more efficiently with precomputed beaver matrices. Moreover, many other works [31], [35] for privacy-preserving machine learning take this idea to deal with linear operations in machine learning tasks. To the best of our knowledge, our work is the first to bring the computation of matrix powering to constant round complexity, and communication complexity without the power p.

There are also some works focusing on the offline phase. Wagh et al. [35] use generalized beaver matrices multiplication applied the idea on a three-party honest majority setting. Due to the fact that only one party could be malicious in such a setting, it is feasible to let one party play as a helper by generating beaver matrices locally in the online phase, send it to the other two parties, then let the other two parties run 2PC. This method gets rid of the expensive offline phase but it can only be used in a 3PC setting. Chen et al. [14] focus on matrix triples generation using homomorphic encryption under dishonest majority settings and the technique could be also used in our work to achieve the efficient offline phase.

IX. CONCLUSION

To conclude, we propose Polymath, a secure multi-party computation toolkit that supports the evaluation of polynomials over both scalars and matrices. The protocols in Polymath have constant round complexity and require less communication than other works, thus making the evaluation of high-degree multi-variate polynomials efficient. 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. As for the polynomial of matrices, we show how our protocol can be used to solve credit risk analysis by providing an efficient constant-round protocol for matrix powering.

For future works, we shall further optimize the performance of multiplying an arbitrary number of matrices and providing more constant-round protocols to speed up the operations for intermediate algebraic structures like vectors and matrices.

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] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[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] Hao Chen, Miran Kim, Ilya Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously secure matrix multiplication with applications to private deep learning. Cryptology ePrint Archive, Report 2020/451, 2020. https://eprint.iacr.org/2020/451.

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

[17] 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.

[18] 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.

[19] 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.

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

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

[22] 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.

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

[24] T. Jung, J. Han, and X. Li. Pda: Semantically secure time-series data analytics with dynamic user groups. *IEEE Transactions on Dependable and Secure Computing*, 15(2):260–274, 2018.

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

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

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

[35] 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.

[36] 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.

[37] 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.

APPENDIX

A. Lemmas

Lemma 5. *Define $\mathbb{F}_q$ as an arbitrary field of size q. Let x', x'' be arbitrary fields element from $\mathbb{F}_q$ and r be an independent random field element chosen uniformly at random from the field $\mathbb{F}_q$, then the distribution of $x'-r$ and $x''-r$ are identical, and therefore indistinguishable, from each other.*

Proof: $\forall\lambda, \Pr[x' - r = \lambda] = \Pr[r = x' - \lambda] = \frac{1}{q}$ $\forall\lambda, \Pr[x'' - r = \lambda] = \Pr[r = x'' - \lambda] = \frac{1}{q}$ $\forall\lambda, \Pr[x' - r =$

$\lambda] = \Pr[r = x' - \lambda] = \frac{1}{q} = \Pr[r = x'' - \lambda] = \Pr[x'' - r = \lambda]$ ■

Lemma 6. *Define $\mathbb{F}_q$ as an arbitrary field of size q. Let R be a matrix chosen uniformly at random from the set of all n by n matrix where each element is a field element from $\mathbb{F}_q$. The probably that R does not have an inverse (R is singular) is less than $\frac{1}{q} - \frac{1}{q^2}$.*

Proof: The size 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 0, 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 $n \geq 5$:

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

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

■

Lemma 7. *Let X', X'' be arbitrary elements from $GL(n, q)$ (The General Linear Group, where $GL(n, q)$ is the set of all n by n invertible matrices whose elements are from a finite field of size q). Let R be an independent random field element chosen uniformly at random from the field $GL(n, q)$. Then the distribution of $X'R$ and $X''R$ are identical, and therefore indistinguishable, from each other.*

Proof: $\forall \lambda, \Pr[X'R = \lambda] = \Pr[R = X'^{-1}\lambda] = \frac{1}{|GL(n,q)|}$ $\forall \lambda, \Pr[X''R = \lambda] = \Pr[R = X''^{-1}\lambda] = \frac{1}{|GL(n,q)|}$ $\forall \lambda, \Pr[X'R = \lambda] = \Pr[R = X'^{-1}\lambda] = \frac{1}{|GL(n,q)|} = \Pr[R = X''^{-1}\lambda] = \Pr[X''R = \lambda]$ ■

Lemma 8. *Let R be a random matrix chosen uniformly at random from $GL(n, q)$ (The General Linear Group, where $GL(n, q)$ is the set of all n by n invertible matrices whose elements are from a finite field of size q). Let X be an arbitrary matrix in $GL(n, q)$. $\forall \lambda, \Pr[RXR^{-1} = \lambda] = \Pr[R \in S] = \frac{|S|}{|GL(n,q)|} = \frac{1}{|Cl(X)|}$ where S is a specific subset of $GL(n, q)$ of size $\frac{|GL(n,q)|}{|Cl(X)|}$ where $Cl(X)$ is the conjugacy class of X.*

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

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 $\frac{|GL(n,q)|}{|Cl(X)|}$ ■

B. Proof of Protocols

1) Proof for the protocol Mul-triple-term:
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 [34]. 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 $z - c$

Theorem 4. *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 $x = x'', y = y'', z = z''$.*

Proof: By Lemma 5, 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 $x'' - a$, $y'' - c$, $z'' - c$. ■

2) Proof for the protocol Mul-arbitrary-term:
We prove that our protocol reveals no information about $x_1, x_2, \dots x_n$ by showing that given two different sets of values for $x_1, x_2, \dots x_n$, 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 $x_1, x_2, \dots x_n$
- Messages for Party P_i related to the secret sharing of $a_1, a_2, \dots a_n$
- Messages for Party P_i related to the secret sharing of $(a_1 a_2 \dots a_n)^{-1}$
- All other parties' shares of $x_1 - a_1, x_2 - a_2, \dots x_n - a_n$
- Messages related to doing multiplication of two hidden 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 $x_1, x_2, \dots x_n, a_1, a_2, \dots a_n, (a_1 a_2 \dots a_n)^{-1}$, 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 [34]. 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 $x_1 - a_1, x_2 - a_2, \dots x_n - a_n$
- The value of $x_1 - a_1, x_2 - a_2, \dots x_n - a_n$

Theorem 5. *Let $x'_1, x'_2, \dots x'_n, x''_1, x''_2, \dots x''_n$ be arbitrary values. The probability distribution of each transcript are identical when $x_1 = x'_1, x_2 = x'_2, \dots x_n = x'_n$ and when $x_1 = x''_1, x_2 = x''_2, \dots x_n = x''_n$.*

Proof: By Lemma 5, the value distribution for $x_1 - a_1, x_2 - a_2, \dots x_n - a_n$ are identical for different values for $x_1, x_2, \dots x_n$, namely, all universally random. Furthermore, since the probability distributions of $x'_1 - a_1$, $x'_2 - a_2$, $\dots$ $x'_n - a_n$, and $x''_1 - a_1$, $x''_2 - a_2$, $\dots$ $x''_n - a_n$, are uniformly random, the probability distributions of the secret shares of $x'_1 - a_1$, $x'_2 - a_2$, $\dots$ $x'_n - a_n$ are identical to that of $x''_1 - a_1$, $x''_2 - a_2$, $\dots$ $x''_n - a_n$. ■

3) Proof For the protocol Matrix Powering:
By Lemma 6, 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 matrix with Beaver Matrix Multiplication
- Messages related to doing multiplication of three secret matrix

For messages related to doing multiplication of two secret matrix with Beaver matrix multiplication [29], 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 [34]. Therefore, we ignore those messages in our security analysis.

For Messages related to doing multiplication of three secret matrix, 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}

Theorem 6. *Let X', X'' be arbitrary values under the constraint that they belong to the same conjugacy class. The*

probability distribution of each transcript are identical when $X = X'$ *and when* $X = X''$.

Proof: By lemma 7, 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).

$\forall \lambda, \Pr[RX'R^{-1} = \lambda] = \Pr[R \in S'] = \frac{|S'|}{|GL(n,q)|} = \frac{1}{\frac{|Cl(X')|}{|GL(n,q)|}}$ where S' is a specific subset of $GL(n, q)$ of size $\frac{|GL(n,q)|}{|Cl(X')|}$ by Lemma 8.

$\forall \lambda, \Pr[RX''R^{-1} = \lambda] = \Pr[R \in S''] = \frac{1}{|Cl(X'')|}$ 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 $|Cl(X')| = |Cl(X'')|$. ■