Multi-Server Verifiable Computation of Low-Degree Polynomials

Liang Feng Zhang
ShanghaiTech University

Huaxiong Wang
Nanyang Technological University

2022 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-1316-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/SP46214.2022.9833792

***Abstract*—The conflicts between input privacy and efficiency in single-server non-interactive verifiable computation (NIVC) makes it interesting to consider the multi-server models of NIVC. Although the existing multi-server NIVC schemes provide meaningful improvements, they either require the servers to communicate or leave the client's data unprotected. It has been an open problem to design multi-server NIVC with both input privacy and non-communicating servers. In this paper we define a multi-server verifiable computation (MSVC) model where the client secret-shares its input x among non-communicating servers, each server locally computes a function F to get a partial result, and finally the client reconstructs $F(x)$ from all partial results. We construct five MSVC schemes for outsourcing low-degree polynomials and thus answer the open question for such polynomials. Our schemes are t-private such that any t servers learn no information about x. Our schemes are t-secure such that any t servers cannot persuade the client to output wrong results. The privacy and security can be either information-theoretic or computational. Comparing with the existing schemes, our servers can be at least two orders faster.**

I. Introduction

Outsourcing computation has been very popular in recent years due to the prevalence of cloud computing and the proliferation of mobile devices. It allows computationally weak devices to offload heavy computations to powerful cloud servers in a scalable pay-per-use manner. The outsourced computations are usually modeled as evaluating a function F at an input x. There are two fundamental security concerns in outsourcing computation: (1) The servers may be malicious or malfunctioning and return incorrect results. (2) The servers may be curious about the client's data (e.g., input x) and abuse it. In [89], the problems of verifying the correctness of the server's computation and protecting the client's data have been termed as the *computation integrity* problem and the *data confidentiality* problem, respectively.

In the theoretical community, solutions to the computation integrity problem date back to as early as the interactive proofs of [6], [56] and the efficient arguments of [69], [70]. Goldwasser et al. [57] constructed interactive proofs that are suitable for outsourcing the computation of log-space uniform boolean circuits. In particular, for circuits of depth d and input length n, the prover is efficient and runs in time $\mathrm{poly}(n)$ and the verifier is super-efficient and runs in time $(n+d)\cdot \mathrm{polylog}(n)$ and space $O(\log n)$. They found it very interesting to outsource computations with a *non-interactive* or single-round scheme, where the server sends at most one message to the client, and extended their result to a non-interactive argument for a more restricted class of functions. Such schemes are particularly interesting because the client can farm out computations without preserving active connections to servers and later the result can be returned via email with a fully written down "certificate" of correctness.

Since [57], theoretical research in the field of outsourcing computation has largely focused on non-interactive schemes for ensuring computation integrity and results in many different models [19], [30], [51], [52], [59]. We are mostly interested in the non-interactive *verifiable computation* (NIVC) model of Gennaro et al. [51]. This single-server model consists of two phases [4]: an *offline phase*, where the client sends an encoding of F to the server; and an *online phase*, where the client sends an encoding of x to the server, the server replies with an encoding of $F(x)$, and the client performs verification and reconstructs $F(x)$. The client's offline computation is executed only once and the cost can be *amortized* over many evaluations of F. The client's online computation should be *substantially faster* than the *native computation* of $F(x)$. The server's computation should be as fast as possible.

There are two different lines in the study of single-server NIVC. One of the lines [4], [38], [51] focuses on the outsourcing of *generic* functions such as any boolean circuits. These schemes provide not only computation integrity but also *input privacy*. However, the client/server's computations in these schemes are quite *impractical* due to their dependance on the expensive cryptographic primitives such as fully homomorphic encryptions (FHEs) and garbled circuits (GCs). The other line [17], [46], [48], [49], [80] focuses on more efficient schemes that are free of FHEs and GCs, at the price of *sacrificing input privacy* or the *generality* of functions.

In a single-server NIVC for outsourcing generic functions, it is quite challenging to achieve both input privacy and high efficiency. In fact, both Ananth et al. [2] and Schoenmakers et al. [81] believe that some form of FHE is inherently required by such schemes: in order to keep x private, the encoding of x in single-server NIVC can be viewed as an "encryption" of x and the encoding of $F(x)$, which is computed with F and the "encryption" of x, must allow the recovery of $F(x)$. The question is still challenging even if we consider the outsourcing of specific functions. The client may have to send an SHE (somewhat homomorphic encryption) ciphertext of x to the server and the server has to perform many expensive public-key operations to generate an encoding of $F(x)$.

A number of recent works have resorted to multiple servers,

DOI 10.1109/SP46214.2022.00087

in order to resolve the conflicts between input privacy and efficiency in NIVC. Canetti et al. [27], [28] constructed multi-server schemes where the client's input is always given to the server *in clear*. Ananth et. al. [2] constructed multi-server schemes for outsourcing boolean circuits, where the client's input is hidden with the expensive primitive of *GCs*. Their schemes require *sequential communications*: the client sends a message to the first server; from the second server on, each server receives a message from the previous server and sends a message to the next server, and the last server sends a message to the client. By distributing Pinocchio [79] to three (or more) servers, Schoenmakers et al. [81] constructed schemes for outsourcing any boolean/arithmetic circuits, where the client's input is information-theoretically private from each server. In Trinocchio [81], the servers run an MPC protocol to evaluate F and the MPC requires the servers to *communicate with each other*; the security relies on *non-falsifiable* assumptions [54].

Therefore, if we restrict our attention to NIVC schemes that use multiple servers, the state of the art offers protocols that either provide no input privacy (e.g. [27], [28]) or require the servers to communicate with each other (e.g. [2], [81]). In this paper, we are interested in *efficient multi-server* schemes that achieve *input privacy* with *non-communicating* servers.

Input privacy is extremely important and enables the client to save time by outsourcing computations, even if the input is sensitive [81] and the servers are not expected to learn any partial information. *Free of server communications* is also an important feature, from which both the cloud servers and the client may benefit. Communications among servers require the servers to operate sequentially. Very complex coordinations among servers may be needed and therefore significantly diminish the efficiency of the entire system, especially when the servers belong to competing cloud services. In multi-server NIVC schemes, the privacy of the client's input usually relies on the assumption that servers do not collude with each other (e.g., [81]). Requiring servers to communicate means that each server may know which other servers are working on the same computation and facilitate them to collude. Without server communications, it may be possible for the client to keep the leased servers anonymous and thus reduce the potential threat. In fact, Ananth et al. [2] put forward a very interesting open question of constructing schemes where the client sends a single message to each of the servers and receives a single message from each of the servers, and can obtain the correct result from this (i.e., a model in which the servers do not communicate with each other at all).

A. Our Contributions

We answer the open question of [2] for outsourcing low-degree multivariate polynomials, which may have many interesting applications (see Section I-B). We propose a new multi-server verifiable computation (MSVC) model. In our model, a k-server verifiable computation is a protocol between a *client* and k *servers*. In such a protocol, the client distributes both a share of the function F and a share of the input x to each server; each server locally computes a partial result; and finally the client performs verification and reconstructs $F(x)$ from all servers' partial results. An MSVC scheme is *t-secure* if no t servers can persuade the client to output a wrong result. Both *information-theoretic* (i.t.) security and *computational* security will be considered. An MSVC scheme is *t-private* if no t servers can distinguish between two inputs. We shall consider information-theoretic privacy. We construct five schemes (see Table I) for securely outsourcing $\mathcal{P}(q, m, d)$, the set of polynomials that have coefficients in a finite field $\mathbb{F}_q$, m variables, and total degree $\leq d$.

TABLE I
t-private (secure) k-server VC schemes for degree-d polynomials

	k	t-security	t-privacy	verification	delegation
Π_1	$d(t+1)+1$	i.t.	i.t.	private	public
Π_2	$(d+1)t+1$	i.t.	i.t.	private	public
Π_3	$dt+1$	i.t.	i.t.	private	public
Π_4	$d(t+1)+1$	DHI	i.t.	public	public
Π_5	$(d+1)t+1$	DLog	i.t.	public	public

Input privacy & security. The client's input x is information-theoretically t-private in all schemes. The schemes Π_1, Π_2, and Π_3 also have information-theoretic t-security; Π_4 and Π_5 are computationally t-secure under the DHI assumption [30] and the DLog assumption, respectively.

Delegatability. In all schemes, any client can prepare its input x, verify the servers' partial results and reconstruct $F(x)$, i.e., all schemes are *publicly delegatable* [80].

Verifiability. The schemes Π_1, Π_2 and Π_3 are *privately verifiable* such that only the client that prepared x is able to verify the servers' partial results. Π_4 and Π_5 are *publicly verifiable* such that any third party can verify.

Outsourceability. The client in Π_1 and Π_4 has to perform an offline preprocessing of F. We follow the amortized model of [51], [79], [80] and call an MSVC *outsourceable* if the client's online computation is substantially faster than the native computation of $F(x)$. Our schemes are all outsourceable.

Server's computation. In all schemes, the computation of each server is very efficient and roughly equivalent to evaluating the outsourced function F once.

Number of servers. To outsource degree-d polynomials with t-privacy, Π_3 requires $dt+1$ servers. This number of servers is *optimal/least* provided that information-theoretic t-privacy is needed and the servers do not communicate. In fact, such MSVC implies a t-private k-player d-multiplicative secret sharing [12], which exists if and only if $k > dt$.

Limitations. The number of servers required by our schemes is linear in the degree d of the outsourced polynomial. A large d may result in a large number of servers. Therefore, although our schemes can handle polynomials of any degree, for sake of practicality they are more suitable for outsourcing low-degree polynomials, which have interesting applications.

B. Applications

Curve fitting on private data points. Curve fitting [5] is the process of constructing a curve or function $y = f(x)$ that has the best fit to a set of data points $\{(x_i, y_i)\}_{i=1}^m$. Depending on whether the data points exhibit a significant

degree of error, curve fitting may be realized with *least squares regression* [34] or *interpolation* [34]. In least squares regression, to fit a degree-d polynomial, one has to evaluate polynomials in $\{(x_i, y_i)\}_{i=1}^{m}$, whose degrees are determined by d. Small values of d are usually preferred and approached first [58]. Thus, low-degree polynomial evaluations will be frequently used. The same situation occurs in multiple linear regression and interpolation. When the data points contain highly sensitive personal information [65], [77], our schemes provide solutions that are both private and secure.

Private information retrieval. In a t-private k-server PIR, each server stores a database $F = (F_1, F_2, \ldots, F_n)$; by sending a query to each server and learn the servers' answers, the client can reconstruct a block F_i of its choice but any t servers cannot learn i. The t-private t-*Byzantine robust* k-server PIR schemes of [15], [73] allow the client to reconstruct correctly, even if t servers respond incorrectly. They also enable the *identification* of cheating servers. For general t, the best such schemes to date have communication complexity $O(n^{1/(\lfloor(2k-1)/t\rfloor-4)})$. Identification of cheating servers may be overly strong in many scenarios such as private media browsing [62], where it suffices for the client to detect the existence of cheating. Our MSVC gives t-private k-server PIR with *private (*resp. *public) detection* of cheating and *better* communication complexity of $O(n^{1/\lfloor(2k-1)/t\rfloor})$ (resp. $O(n^{1/\nu(k,t)})$, where $\nu(k,t) = \max\{\lfloor\frac{2k-1}{t+1}\rfloor, \lfloor\frac{2k-1}{t}\rfloor - 1\}$). Comparing with PIR for honest servers, our schemes incur *limited extra cost*.

Privacy preserving statics and GAS. Computations of many meaningful statistics such as average, variance, covariance, RMS, correlation coefficient and many more, and the Genetic Risk Scores in GAS (Genetic Association Studies) [10] heavily depend on the low-degree multivariate polynomial evaluations. Our schemes provide secure outsourcing solutions that preserve the privacy of sensitive data.

C. Efficiency

Our schemes are all outsourceable in the amortized model of [51], [79], [80]. Nevertheless, we focus on the more realistic metrics of performance such as the client's *time cost* and *monetary cost* in MSVC. The time cost is the total time spent by the client in preparing input, waiting for the latest answer from a server, and verifying the results. The monetary cost is measured with the equivalent local computing cost of the input preparation, the result verification, and all servers' computations. We test the proposed schemes by evaluating polynomials of degree 2, 4, 8, 16, 32 and 64. The experimental results show that the ratio of the client's time cost to the time of locally computing $F(\mathbf{x})$ tends to the ratio of the computing speed of the client to that of each server; and the ratio of the client's monetary cost to the cost of locally computing $F(\mathbf{x})$ tends to a multiple of the previous ratio, where the multiple is roughly equal to the number of servers. In the schemes with preprocessing (i.e., Π_1 and Π_4), the client will benefit only if the same function is evaluated multiple times. In our experiments for $d = 2$, this number is ≤ 20.

D. Our Techniques

t-Privacy. In our constructions, t-privacy is achieved by secret-sharing the input $\mathbf{x} \in \mathbb{F}_q^m$ among all servers using Shamir's threshold scheme [87] for vectors. The client chooses a random degree-t curve $c(u)$ that passes through $(0, \mathbf{x})$ and distributes $k > dt$ curve points to k servers; the k servers provide k evaluations of F; finally the client interpolates $\phi(u) = F(c(u))$ and learns $F(\mathbf{x}) = \phi(0)$.

t-Security. The verification techniques in five schemes are different. In Π_1, the client picks a random line $\ell(u)$ and makes both the line and the restriction of F on the line (i.e., $f(u) = F(\ell(u))$) public. It also increases the degree of $c(u)$ by 1 such that $c(u)$ intersects $\ell(u)$ at a random point. With this choice, $k = d(t+1)+1$ evaluations of F suffice to recover $\phi(u)$. For verifications, it evaluates F at the random point in two different ways: one is with $\phi(u)$ and the other is with $f(u)$. It accepts only if two results agree. The main idea is leaving the heavy computation of $f(u)$ to the offline phase such that the client is able to quickly retrieve the value for verification in the online phase.

In Π_2, the client additionally secret-shares a random field element α among the servers, by using a degree-t univariate polynomial $b(u)$. Each server is giving a point on $c(u)$ and a share of α, which is an evaluation of $b(u)$. The client offloads the computation of $F(\mathbf{x})$ and $\alpha F(\mathbf{x})$ to $k = (d+1)t+1$ servers. This is done by every server providing both a share of $F(\mathbf{x})$ and a share of $\alpha F(\mathbf{x})$ such that the client is able to recover $\phi(u) = F(c(u))$ and $\psi(u) = F(c(u))b(u)$. If the free terms of both polynomials differ by a factor α, then the client believes that $\phi(u)$ is correct. Without knowing α, any t servers can cheat successfully only with a very small probability.

In Π_3, the client will choose $c(u)$ such that it passes through through $(\alpha, \mathbf{x})$ for a random field element α. It makes a critical change by choosing both the coefficients of $c(u)$ and α from an extension field of $\mathbb{F}_q$ such as $\mathbb{F}_{q^2}$, instead of $\mathbb{F}_q$. Then any $k = dt+1$ evaluations of F suffice to recover $\phi(u) = F(c(u))$ and give $F(\mathbf{x}) = \phi(\alpha)$. Without knowing α, the wrong partial results provided by any t servers will result in the client reconstructing a value in $\mathbb{F}_{q^2} \setminus \mathbb{F}_q$ with overwhelming probability and be rejected by the client.

Π_4 is obtained from Π_1 by converting the private verification to public. In Π_1, the client has to memorize the locations where $c(u)$ and $\ell(u)$ intersect. This leads to a private verification that compares the evaluations of two polynomials, i.e., $\phi(u)$ and $f(u)$. Given a cyclic group $\mathbb{G} = \langle g \rangle$ of order q, we publish the exponentiation of two locations and thus move the comparison to the exponent of g. We prove the scheme is secure under the *DHI assumption* in $\mathbb{G}$.

The scheme Π_5 is obtained from Π_2 with a public key g^α. The verification is moved to the exponent of g as checking whether $(g^\alpha)^{F(\mathbf{x})} = g^{\alpha F(\mathbf{x})}$. We prove the scheme is secure under the *DLog assumption* in $\mathbb{G}$.

E. Related Work

In the security community, correctness of outsourced computations may be verified with replication, audit, or secure

co-processors. The replication based solutions [3], [29], [63], [68] may be not viable and provide no input privacy. The audit-based solutions [16], [78], [82] require recalculation of the server's work or knowledge of the server's hardware. The secure co-processors [90], [95] are either poor in physical tamper resistance or expensive. In the cryptographic community, expensive cryptographic operations had been outsourced to semi-trusted devices [35].

Interactive proofs. Interactive proofs [6], [56] allow a powerful prover to convince a weak verifier of the truth of statements that could be too complex to be computed by the verifier. The statement may take the form $F(x) = y$. Early research in this field focused on how to use limited resources to verify complex statements. The IP=PSPACE theorem of [75], [88] shows that polynomial space computations is verifiable in polynomial time. It was scaled down in [50] for a superpolynomial time prover. For NC circuits, Goldwasser et al. [57] proposed interactive proofs where the prover runs in polynomial time and the verifier runs in nearly linear time. The protocol of [57] has been refined and implemented in [39], [91], [93]. For NC circuits, these systems do not require preprocessing, have a highly efficient verifier, achieve low overhead for the prover, and have information-theoretic security. However, they are *interactive* and leave the client's input *unprotected*.

Interactive arguments. The MIP=NEXP theorem of [8] constructed multi-prover interactive proofs with a polynomial time prover and a polylogarithmic time verifier, and led to the notion of probabilistically checkable proofs (PCP) [7]. Although a few locations of PCP suffice for verification, PCP is too long and infeasible for the verifier to process. Kilian [69], [70] suggested the prover send a Merkle tree based commitment of PCP and then interactively open the verifier-requested locations. Kilian's idea results in interactive arguments [24] where the soundness holds for computationally bound provers. Ishai et al. [67] simplified the PCP by using a homomorphic encryption based technique. The simplified PCP was refined and implemented in [25], [84]–[86]. These systems can outsource generic functions, require a preprocessing for the verifier and have high prover overhead. These *interactive* systems provide *no input privacy*.

Succinct non-interactive arguments of knowledge. The non-interactive argument system of Micali [76] removed the interactions in [69], [70] by applying a random oracle to the commitment and then using the output to choose the locations that will be opened. More efficient non-interactive argument systems [19], [52], [60], [61], [79] were based on the CRS model and called succinct non-interactive arguments (SNARGs). SNARGs were then strengthened to succinct non-interactive arguments of knowledge (SNARKs) [19], [45], [52], [61], [79] such that the prover producing a convincing proof must "know" a witness. Recently many efficient SNARKs implementations [36], [83] have been proposed. An updated survey for both interactive proofs and SNARKs can be found in Thaler [92]. Most of the SNARKs require non-falsifiable assumptions [54], though some of the recent works such as [61] allows one to sidestep non-falsifiable assumptions by using stronger computational models. SNARKs in general provide *no input privacy*.

Homomorphic authenticators (HAs). HAs [1], [21], [22], [30], [32], [53] allow a client to compute authenticators for the elements of a dataset $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ such that the server is able to generate authenticator for a computation $F(\mathbf{x})$. The verification however is as heavy as the native computation. Catalano et al. [31], [33] constructed outsourceable HAs with multilinear maps. By considering multiple datasets, Backes et al. [9] and Gorbunov et al. [59] constructed HAs that have efficient amortized verification. None of them keeps data private. Fiore et al. [49] constructed HAs for encrypted data that can only compute quadratic polynomials.

Multiplicative secret sharing (MSS). In MSS [12], a client secret-shares the data $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ among multiple servers; each server can locally compute a partial result such that all partial results sum to $\prod_{i=1}^{d} x_i$. Verifiably MSS (VMSS) [96] additionally enables the client to verify the servers' partial results and achieves information-theoretic privacy and security. However, it only allows the computation of *monomials*.

Homomorphic secret sharing (HSS). HSS [23], [74] allows a client to secret-share the data $\mathbf{x}$ among servers and then offload the computation of $F(\mathbf{x})$ to servers. HSS provides *computational* input privacy and *no verifiability*.

Multiparty computation. In the client-servers setting, Barkol et al. [11] constructed MPC protocols that allow a client to privately compute constant-depth circuits, but *without verification*. Dachman-Soled et al. [42] proposed an MPC protocol for computing multivariate polynomials where parties hold different *variables* as private inputs. The work of making MPC practical has been successful [72]. However, the primary focus of MPC is not outsourcing and each party's computation is typically *as heavy as the native computation*.

Multi-prover interactive proofs. The multi-prover interactive proofs (arguments) [20], [71] are quite efficient. However, they are *interactive* and leave the client's input *unprotected*.

Verifiable secret sharing (VSS). In VSS [26], [47], the adversary can completely dictate the behavior of the participants under its control and may also control the dealer. The verifiability of VSS guarantees that even if the dealer is corrupted, it still has consistently shared some value among the participants and the same value is later reconstructed. In our MSVC, the client may be considered as a dealer that shares $\mathbf{x}$ among the servers (participants). There are four fundamental differences between MSVC and VSS. First, the client in MSVC is never cheating. Second, the servers in MSVC never directly reconstruct $\mathbf{x}$. Instead, each server locally computes a partial result with its share. The client is responsible to reconstruct $F(\mathbf{x})$ from partial results. Third, although VSS is applicable to MPC, the resulting protocols require communications among the parties. Fourth, MSVC allows the detection of cheating but provides no guarantee of reconstructing a unique value.

Verifiable PIR. In the *verifiable* PIR protocols of [97], [98],

TABLE II
The running time T_{client} of the client and the total running time T_{servers} of all servers (seconds)

case	Π_4		Π_5		Trinocchio	
	T_{client}	T_{servers}	T_{client}	T_{servers}	T_{client}	T_{servers}
medium	0.29	33	0.07	**17**	**0.04**	6561
large	0.49	115	0.13	**59**	**0.05**	20106

any server that provided a wrong answer will be identified. Although these protocols use no additional servers, both the client and the servers in these protocols have to do a lot of public-key operations. For example, in the protocols of [97], [98], the client has to spend thousands of seconds in retrieving an item from a database of 2^{16} items. In contrast, our client and severs only need tens of milliseconds and tens of seconds respectively to retrieve an item from a database of 10^6 items.

Trinocchio. Comparing with [81], our schemes are free of server communications and non-falsifiable assumptions. Trinocchio [81] has a case study of multivariate polynomial evaluations (a medium case and a large case). We do the same case studies with Π_4 and Π_5, which share the same properties of public delegation/verification with [81]. The experimental results (Table II) show that the clients in our schemes may be slightly slower but the servers are at least two orders faster. Moreover, Π_5 is free of a heavy preprocessing of F, which is needed by Trinocchio. Trinocchio represents any function as an arithmetic circuit and the servers need to perform a large number of exponentiations, which result in worse performance. The number of required servers in MSVC depends on the degree of F. However, this is a theoretical consequence of information-theoretic privacy plus non-communicating servers. In [81], the servers have to interactively run MPC to evaluate $F(\mathbf{x})$. The same number of servers would be needed if the Trinocchio servers are not allowed to communicate.

II. Multi-Server Verifiable Computation

For any set S, we denote with "$s \leftarrow S$" the procedure of choosing an element s uniformly from S. For an algorithm $\mathcal{A}$, we denote with "$y \leftarrow \mathcal{A}(x)$" the procedure of running $\mathcal{A}$ on an input x and assigning its output to y. For any integer $k > 0$, we denote $[k] = \{1, 2, \ldots, k\}$. For any set T and any vector x, x_T will stand for the subvector $(x_i)_{i \in T}$.

A k-server verifiable computation scheme is a protocol between a *client* and k *servers* $\mathcal{S}_1, \ldots, \mathcal{S}_k$. In such a protocol, the client provides both a share of the input x and a share of the function F to each server; each server computes a partial result; and finally the client recovers $F(x)$ from the k partial results. Let $\mathcal{F}$ be a set of functions. A *k-server verifiable computation* scheme $\Pi = (\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ for $\mathcal{F}$ consists of the following algorithms:

- $(\mathsf{pk}_F, \mathsf{vk}_F, \{\rho_i\}_{i=1}^k) \leftarrow \mathsf{KeyGen}(\lambda, F)$*: The key generation algorithm takes a security parameter λ and any function $F \in \mathcal{F}$ as input and generates a value pk_F, which will be used by the client to prepare its input $x \in \mathrm{Dom}(F)$, k function shares $\rho_1, \ldots, \rho_k$, which will be used by the servers to compute partial results, and a value vk_F, which will be used by the client to perform verifications.*
- $(\mathsf{vk}_x, \{\sigma_i\}_{i=1}^k) \leftarrow \mathsf{ProbGen}(\mathsf{pk}_F, x)$*: The problem generation algorithm uses pk_F to encode any input $x \in \mathrm{Dom}(F)$ as k input shares $\sigma_1, \ldots, \sigma_k$, which will be given to the servers to compute with, and a value vk_x, which will used by the client to perform verifications.*
- $\pi_i \leftarrow \mathsf{Compute}(i, \rho_i, \sigma_i)$*: For every $i \in [k]$, the server's algorithm computes a partial result π_i with (ρ_i, σ_i).*
- $\{F(x), \perp\} \leftarrow \mathsf{Verify}(\mathsf{vk}_F, \mathsf{vk}_x, \{\pi_i\}_{i=1}^k)$*: The verification algorithm uses vk_F and vk_x to determine if $\{\pi_i\}_{i=1}^k$ form a valid encoding of $F(x)$. If it is invalid, outputs $\perp$; otherwise, reconstructs $F(x)$ from $\{\pi_i\}_{i=1}^k$.*

An MSVC scheme is *publicly delegatable* if pk_F is public such that anyone is able to run $\mathsf{ProbGen}$ to delegate computations. An MSVC scheme is *publicly verifiable* if both vk_F and vk_x are public. Otherwise, the scheme is *privately verifiable*. Public delegation/verification are generally preferred. In this paper, we will construct publicly delegatable schemes, among which some are publicly verifiable and the others are privately verifiable. In particular, all schemes have a public vk_F.

An MSVC scheme is *correct* if KeyGen and $\mathsf{ProbGen}$ produce values that always enable the honest servers to compute values that will verify successfully and be converted into $F(x)$.

Definition 1. (Correctness) *An MSVC scheme Π is said to be $\mathcal{F}$-*correct *if for any $F \in \mathcal{F}$, any $(\mathsf{pk}_F, \mathsf{vk}_F, \{\rho_i\}_{i=1}^k) \leftarrow \mathsf{KeyGen}(\lambda, F)$, any $x \in \mathrm{Dom}(F)$, any $(\mathsf{vk}_x, \{\sigma_i\}_{i=1}^k) \leftarrow \mathsf{ProbGen}(\mathsf{pk}_F, x)$, any $\{\pi_i \leftarrow \mathsf{Compute}(i, \rho_i, \sigma_i)\}_{i=1}^k$, it holds that $\Pr[\mathsf{Verify}(\mathsf{vk}_F, \mathsf{vk}_x, \{\pi_i\}_{i=1}^k) = F(x)] \geq 1 - \mathrm{negl}(\lambda)$.*

In MSVC, a set of colluding servers may try to persuade the client to output a wrong value with incorrect partial results. For privately verifiable schemes, security against such attacks may be defined by properly generalizing the security experiment of [17], where the adversary can make a number of trials, to our multi-server setting (see Experiment 1). While the trials are essential in [17], they are not necessary here because the vk_F in MSVC is always public and the colluding servers can finish the trials on their own. In Experiment 1, the adversary $\mathcal{A}$ models a set $\mathcal{S}_T$ of colluding servers for some $T \subseteq [k]$. Given $F, \mathsf{pk}_F, \mathsf{vk}_F$ and $\mathcal{S}_T$'s shares of the function, $\mathcal{A}$ picks an input x, learns $\mathcal{S}_T$'s shares of the input, and then chooses a set of partial results for $\mathcal{S}_T$. It breaks the security of MSVC if finally the challenger accepts and reconstructs a wrong value.

Experiment 1. $(\mathbf{Exp}^{\mathrm{PriV}}_{\mathcal{A},\Pi}(T, F, \lambda))$

(a) $(\mathsf{pk}_F, \mathsf{vk}_F, \{\rho_i\}_{i=1}^k) \leftarrow \mathsf{KeyGen}(\lambda, F)$;
(b) $x \leftarrow \mathcal{A}(F, \mathsf{pk}_F, \mathsf{vk}_F, \rho_T)$;
(c) $(\mathsf{vk}_x, \{\sigma_i\}_{i=1}^k) \leftarrow \mathsf{ProbGen}(\mathsf{pk}_F, x)$;
(d) $\hat{\pi}_T \leftarrow \mathcal{A}(F, \mathsf{pk}_F, \mathsf{vk}_F, \rho_T, x, \sigma_T)$;
(e) $\hat{\pi}_i \leftarrow \mathsf{Compute}(i, \rho_i, \sigma_i)$ *for every* $i \in [k] \setminus T$;
(f) $\hat{y} \leftarrow \mathsf{Verify}(\mathsf{vk}_F, \mathsf{vk}_x, \{\hat{\pi}_i\}_{i=1}^k)$;
(g) *if* $\hat{y} \notin \{\perp, F(x)\}$, *output 1; otherwise, output 0.*

Definition 2. (Security for privately verifiable schemes) *For $T \subseteq [k]$ and $\epsilon > 0$, an MSVC scheme Π is said to be (T, ϵ)-*secure *if for any function $F \in \mathcal{F}$, any input $x \in \mathrm{Dom}(F)$, and any adversary $\mathcal{A}$, $\Pr[\mathbf{Exp}^{\mathrm{PriV}}_{\mathcal{A},\Pi}(T, F, \lambda) = 1] \leq \epsilon$, where the probability is taken over the randomness of $\mathcal{A}$ and the*

experiment. Moreover, Π *is* (t,ϵ)-secure *if it is* (T,ϵ)*-secure for any set* $T \subseteq [k]$ *of cardinality* $\leq t$.

While Experiment 1 deals with schemes where vk_F is public but vk_x is private. In our publicly verifiable schemes, both vk_F and vk_x are public. The security of such schemes may defined by generalizing the security experiment of [80] in *single-server* setting to our multi-server setting. The resulting experiment $\mathbf{Exp}^{\mathrm{PubV}}_{\mathcal{A},\Pi}(T,F,\lambda)$ is identical to $\mathbf{Exp}^{\mathrm{PriV}}_{\mathcal{A},\Pi}(T,F,\lambda)$ (as shown in Experiment 1), except that the item (d) is replaced with $\hat{\pi}_T \leftarrow \mathcal{A}(F,\mathsf{pk}_F,\mathsf{vk}_F,\rho_T,x,\mathsf{vk}_x,\sigma_T)$ such that $\mathcal{A}$'s strategy is also based on the public key vk_x.

Definition 3. (Security for publicly verifiable schemes) *For* $T \subseteq [k]$*, an MSVC scheme* Π *is* T-secure *if for any* $F \in \mathcal{F}$*, for any PPT adversary* $\mathcal{A}$*,* $\Pr[\mathbf{Exp}^{\mathrm{PubV}}_{\mathcal{A},\Pi}(T,F,\lambda)=1] \leq \mathrm{negl}(\lambda)$*, where the probability is taken over the randomness of* $\mathcal{A}$ *and the experiment. In particular,* Π *is* t-secure *if it is* T*-secure for any set* $T \subseteq [k]$ *of cardinality* $\leq t$.

Procedure. $\sigma_\Pi(T,F,x)$

$(\mathsf{pk}_F,\mathsf{vk}_F,\{\rho_i\}_{i=1}^k) \leftarrow \mathsf{KeyGen}(\lambda,F)$;
$(\mathsf{vk}_x,\{\sigma_i\}_{i=1}^k) \leftarrow \mathsf{ProbGen}(\mathsf{pk}_F,x)$;
output σ_T.

In MSVC, a set of colluding servers may try to learn the client's input from their input shares. For any $T \subseteq [k]$, $F \in \mathcal{F}$, and $x \in \mathrm{Dom}(F)$, $\mathcal{S}_T$ will learn $|T|$ input shares, which can be generated by the procedure $\sigma_\Pi(T,F,x)$. Informally, we say that Π is T*-private* if no strategy of $\mathcal{S}_T$ is able to distinguish between two inputs.

Definition 4. (Input privacy) *For* $T \subseteq [k]$*, an MSVC scheme* Π *is* T-private *if for any* $F \in \mathcal{F}$*, any* $x^0, x^1 \in \mathrm{Dom}(F)$*,* $\sigma_\Pi(T,F,x^0)$ *and* $\sigma_\Pi(T,F,x^1)$ *are identically distributed. If* Π *is* T*-private for any* $T \subseteq [k]$ *of cardinality* $\leq t$*, then* Π *is said to be* t-private.

Remark. The adversaries in Definition 2 and 4 are unbounded. Thus, both the private verifiability and the input privacy are information-theoretic.

III. Privately Verifiable Schemes

In this section we show three privately verifiable schemes for $\mathcal{P}(q,m,d)$. Each polynomial $F(\mathbf{x}) = F(x_1,\ldots,x_m)$ in this family has up to $n = \binom{m+d}{d}$ terms. For any t, we shall show k-server schemes that are both t-secure and t-private. The three schemes will require $d(t+1)+1, (d+1)t+1$ and $dt+1$ servers, respectively. The third scheme is optimal in terms of the number of servers; and the other two will be turned into publicly verifiable schemes in Section IV.

In all constructions, the basic idea of achieving t-privacy is secret-sharing the input $\mathbf{x} \in \mathbb{F}_q^m$ among all servers using Shamir's threshold scheme [87] for vectors. That is, the client draws a random degree-t curve

$$c(u) = \mathbf{x} + \mathbf{r}_1 u + \cdots + \mathbf{r}_t u^t \tag{1}$$

that resides in $\mathbb{F}_q^m$ and passes through $(0,\mathbf{x})$, and then distributes k curve points to k servers. The k servers return the evaluations of F on k points, which will allow the client to interpolate a degree $\leq dt$ polynomial

$$\phi(u) = F(c(u)) \tag{2}$$

and then learn $F(\mathbf{x}) = F(c(0)) = \phi(0)$. For the sake of polynomial interpolation, we always assume that $k < q$ and each server $\mathcal{S}_i$ is associated with a field element $i \in \mathbb{F}_q$.

The three schemes are mainly different in their verification techniques, which result in different numbers of servers.

A. The First Construction

In our first construction (Scheme 1), the client will be convinced that the $\phi(u)$ in (2) is correct if it takes the correct value at a random point $\alpha \leftarrow \mathbb{F}_q^* \backslash [k]$. In fact, when some of the servers' partial results are wrong, the $\phi(u)$ interpolated from partial results will not take the correct value at a random point, except with very small probability. To verify, the client must be able to learn $\phi(\alpha) = F(c(\alpha))$, the value of F at a random point $\mathbf{a} = c(\alpha)$. The client could choose $\mathbf{a}$ in $\mathsf{KeyGen}(\lambda,F)$ and precompute $F(\mathbf{a})$ for all future verifications. However, that will require the client to memorize $\mathbf{a}$, in order to choose a random curve $c(u)$ that passes through $\mathbf{a}$ in $\mathsf{ProbGen}(\mathsf{pk}_F,\mathbf{x})$, and thus result in a scheme *without public delegation*. If the client picks $\mathbf{a}$ in $\mathsf{ProbGen}(\mathsf{pk}_F,\mathbf{x})$ and after $c(u)$ has been chosen, then it will have to locally compute $F(\mathbf{a})$, which is as heavy as the outsourced computation.

Scheme 1. The k-Server Scheme Π_1 ($k = d(t+1)+1$)

$\mathsf{KeyGen}(\lambda,F)$*: Choose* $\boldsymbol{\ell}_0,\boldsymbol{\ell}_1 \leftarrow \mathbb{F}_q^m$*, let* $\ell(u) = \boldsymbol{\ell}_0 + \boldsymbol{\ell}_1 u$ *and* $\rho_i = F$ *for every* $i \in [k]$*, compute* $f(u) = F(\ell(u))$*, and output* $\mathsf{pk}_F = \ell(u), \mathsf{vk}_F = (\ell(u), f(u))$ *and* $\{\rho_i\}_{i=1}^k$.

$\mathsf{ProbGen}(\mathsf{pk}_F,\mathbf{x})$*: Choose* $a \leftarrow \mathbb{F}_q^*, \alpha \leftarrow \mathbb{F}_q^* \setminus [k], \mathbf{r}_1,\ldots,\mathbf{r}_t \leftarrow \mathbb{F}_q^m$*, let* $\mathbf{a} = \ell(a)$*,* $\mathbf{r}_{t+1} = \alpha^{-t-1}(\mathbf{a} - (\mathbf{x} + \sum_{s=1}^t \mathbf{r}_s \alpha^s))$ *and* $c(u) = \mathbf{x} + \sum_{s=1}^{t+1} \mathbf{r}_s u^s$*, compute* $\sigma_i = c(i)$ *for every* $i \in [k]$*, and output* $\mathsf{vk}_{\mathbf{x}} = (a,\alpha)$ *and* $\{\sigma_i\}_{i=1}^k$.

$\mathsf{Compute}(i,\rho_i,\sigma_i)$*: Parse* ρ_i *as* F *and output* $\pi_i = F(\sigma_i)$.

$\mathsf{Verify}(\mathsf{vk}_F,\mathsf{vk}_{\mathbf{x}},\{\pi_i\}_{i=1}^k)$*: Interpolate a polynomial* $\phi(u)$ *of degree* $< k$ *such that* $\phi(i) = \pi_i$ *for all* $i \in [k]$*. If* $\phi(\alpha) = f(a)$*, output* $\phi(0)$*; otherwise, output* $\perp$.

The client may bypass this difficulty by choosing a random line $\ell(u) = \boldsymbol{\ell}_0 + \boldsymbol{\ell}_1 u$ in KeyGen, precomputing the restriction of F on the line as $f(u) = F(\ell(u))$, and making $(\ell(u), f(u))$ public. Then the client may choose a random curve

$$c(u) = \mathbf{x} + \mathbf{r}_1 u + \cdots + \mathbf{r}_t u^t + \mathbf{r}_{t+1} u^{t+1} \tag{3}$$

that passes through $(0,\mathbf{x})$ and intersects with $\ell(u)$ at a random point $\mathbf{a} = \ell(a) = c(\alpha)$, where $a \leftarrow \mathbb{F}_q^*$ and $\alpha \leftarrow \mathbb{F}_q^* \setminus [k]$. After interpolating $\phi(u) = F(c(u))$ from the servers' partial results, the verification may be done by checking whether $\phi(\alpha) = f(a)$. Now the client only needs to evaluate two low-degree polynomials, i.e., $\phi(u)$ and $f(u)$. The main idea of speeding up the precomputation of $F(\mathbf{a})$ and thus achieving fast verification is dividing the computation of $F(\mathbf{a}) = f(a)$ into two steps: (i) computing $f(u) = F(\ell(u))$; and (ii) computing $f(a)$; and then leaving the heavier step (i) to KeyGen such that it can be amortized.

When the partial results $\{\pi_i\}_{i=1}^k$ are all correctly computed, we have that $\pi_i = F(c(i))$ for all $i \in [k]$. Then the polynomial $\phi(u)$ interpolated in Verify is $F(c(u))$. The correctness of Π_1 follows from the facts that $\phi(\alpha) = F(c(\alpha)) = F(\mathbf{a}) = F(\ell(a)) = f(a)$ and $\phi(0) = F(c(0)) = F(\mathbf{x})$. Regarding input privacy and security, we have the following:

Theorem 1. *The scheme Π_1 is t-private and (t, ϵ)-secure with $\epsilon = \frac{d(t+1)}{q-2-d(t+1)}$ (see Appendix A).*

B. The Second Construction

In our second construction (Scheme 2), the client will use (1) to secret-share its input among the servers and use (2) to reconstruct the output. To add verifiability, the client will additionally secret-share a random field element $\alpha \leftarrow \mathbb{F}_q^*$ among the k servers using a random polynomial

$$b(u) = \alpha + \gamma_1 u + \cdots + \gamma_t u^t \tag{4}$$

of degree $\leq t$. Each server returns not only $F(c(i))$ but also $b(i)F(c(i))$. Finally, the client interpolates two polynomials

$$\phi(u) = F(c(u)), \quad \psi(u) = F(c(u))b(u) \tag{5}$$

of degree $\leq dt$ and $\leq (d+1)t$, respectively. The verification is done by checking whether $\psi(0) = \alpha\phi(0)$.

Scheme 2. The k-Server Scheme Π_2 ($k = (d+1)t + 1$)

$\mathsf{KeyGen}(\lambda, F)$: *Let $\rho_i = F$ for all $i \in [k]$. Output $\mathsf{pk}_F = \perp$, $\mathsf{vk}_F = \perp$ and $\{\rho_i\}_{i=1}^k$.*

$\mathsf{ProbGen}(\mathsf{pk}_F, \mathbf{x})$: *Choose $\alpha \leftarrow \mathbb{F}_q^*, \mathbf{r}_1, \ldots, \mathbf{r}_t \leftarrow \mathbb{F}_q^m, \gamma_1, \ldots, \gamma_t \leftarrow \mathbb{F}_q$, let $c(u) = \mathbf{x} + \sum_{s=1}^t \mathbf{r}_s u^s$ and $b(u) = \alpha + \sum_{s=1}^t \gamma_s u^s$, and compute $\sigma_i = (c(i), b(i))$ for all $i \in [k]$. Output $\mathsf{vk}_\mathbf{x} = \alpha$ and $\{\sigma_i\}_{i=1}^k$.*

$\mathsf{Compute}(i, \rho_i, \sigma_i)$: *Parse ρ_i as F, compute $v_i = F(c(i))$ and $w_i = v_i b(i)$, and output $\pi_i = (v_i, w_i)$.*

$\mathsf{Verify}(\mathsf{vk}_F, \mathsf{vk}_\mathbf{x}, \{\pi_i\}_{i=1}^k)$: *Interpolate a polynomial $\phi(u)$ of degree $\leq dt$ such that $\phi(i) = v_i$ for all $i \in [k]$ and a polynomial $\psi(u)$ of degree $\leq (d+1)t$ such that $\psi(i) = w_i$ for all $i \in [k]$. If $\psi(0) = \alpha\phi(0)$, output $\phi(0)$; otherwise, output $\perp$.*

When the partial results $\{(v_i, w_i)\}_{i=1}^k$ are all correctly computed, we have that $v_i = F(c(i))$ and $w_i = F(c(i))b(i)$ for all $i \in [k]$. Then the polynomials $\phi(u)$ and $\psi(u)$, which are interpolated in Verify, will satisfy (5). The correctness of Π_2 then follows from the facts that $\psi(0) = \phi(0)b(0) = \alpha\phi(0)$ and $\phi(0) = F(c(0)) = F(\mathbf{x})$. The cheating servers may forge a set of partial results which then result in two polynomials $\hat{\phi}(u)$ and $\hat{\psi}(u)$ in Verify. However, without knowing α, $\hat{\psi}(0) = \alpha\hat{\phi}(0)$ will be false except with very small probability.

Theorem 2. *The scheme Π_2 is t-private and (t, ϵ)-secure with $\epsilon = \frac{1}{q-1}$ (see Appendix B).*

C. The Third Construction

In our third construction (Scheme 3), the client will work over $\mathbb{F}_{q^2}$. Instead of choosing a random curve $c(u)$ that passes through $(0, \mathbf{x})$, it will choose a random curve

$$c(u) = \mathbf{x} + \mathbf{r}_1(u - \alpha) + \cdots + \mathbf{r}_t(u^t - \alpha^t) \tag{6}$$

that passes through $(\alpha, \mathbf{x})$ at a random point $\alpha \leftarrow \mathbb{F}_{q^2}^* \setminus [k]$. Each server $\mathcal{S}_i$ will be given a curve point $(i, c(i))$ such that any t servers learn no information about $\mathbf{x}$. Given $k = dt + 1$ servers, the client would be able to reconstruct the degree $\leq dt$ polynomial $\phi(u) = F(c(u))$ and learn $F(\mathbf{x}) = \phi(\alpha) \in \mathbb{F}_q$.

Scheme 3. The k-Server Scheme Π_3 ($k = dt + 1$)

$\mathsf{KeyGen}(\lambda, F)$: *Let $\rho_i = F$ for all $i \in [k]$. Output $\mathsf{pk}_F = \perp$, $\mathsf{vk}_F = \perp$ and $\{\rho_i\}_{i=1}^k$.*

$\mathsf{ProbGen}(\mathsf{pk}_F, \mathbf{x})$: *Choose $\alpha \leftarrow \mathbb{F}_{q^2}^* \setminus [k]$ and $\mathbf{r}_1, \ldots, \mathbf{r}_t \leftarrow \mathbb{F}_{q^2}^m$, let $c(u) = \mathbf{x} + \sum_{s=1}^t \mathbf{r}_s(u^s - \alpha^s)$, set $\sigma_i = c(i)$ for all $i \in [k]$. Output $\mathsf{vk}_\mathbf{x} = \alpha$ and $\{\sigma_i\}_{i=1}^k$.*

$\mathsf{Compute}(i, \rho_i, \sigma_i)$: *Parse ρ_i as F and output $\pi_i = F(\sigma_i)$.*

$\mathsf{Verify}(\mathsf{vk}_F, \mathsf{vk}_\mathbf{x}, \{\pi_i\}_{i=1}^k)$: *Interpolate a polynomial $\phi(u)$ of degree $\leq dt$ such that $\phi(i) = \pi_i$ for all $i \in [k]$. If $\phi(\alpha) \in \mathbb{F}_q$, output $\phi(\alpha)$; otherwise, output $\perp$.*

When the partial results $\{\pi_i\}_{i=1}^k$ are all correctly computed, we have $\pi_i = F(c(i))$ for all $i \in [k]$. Then the polynomial $\phi(u)$ interpolated in Verify is $F(c(u))$. The correctness of Π_3 follows from the fact that $\phi(\alpha) = F(c(\alpha)) = F(\mathbf{x}) \in \mathbb{F}_q$. The cheating servers may provide incorrect partial results, which then result in a polynomial $\hat{\phi}(u) \neq F(c(u))$ in Verify. The cheating will be detected if $\hat{\phi}(\alpha) \notin \mathbb{F}_q$. Therefore, $\hat{\phi}(u)$ will not allow the cheating servers to break the security unless $\hat{\phi}(\alpha) \in \mathbb{F}_q \setminus \{\phi(\alpha)\}$. However, without knowing α, the last event will not occur except with very small probability.

Theorem 3. *The scheme Π_3 is t-private and (t, ϵ)-secure for $\epsilon = \frac{(q-1)dt}{q^2-2-dt}$ (see Appendix C).*

IV. Publicly Verifiable Schemes

The schemes in Section III are privately verifiable. Public verifiability allows a third party to verify as well and thus settle the possible disputes between client and servers. We shall convert Π_1 and Π_2 to schemes with public verification. The security of the new schemes is not information-theoretic but relies on cryptographic assumptions (see Appendix D).

A. The First Construction

In Π_1, the client accepts the servers' partial results if and only if $\phi(\alpha) = f(a)$, where (a, α) is a private key for verification. To obtain a publicly verifiable scheme (Scheme 4), we shall choose a cyclic group $\mathbb{G} = \langle g \rangle$ of prime order q

Scheme 4. The k-Server Scheme Π_4 ($k = d(t+1) + 1$)

$\mathsf{KeyGen}(\lambda, F)$: *Choose $\boldsymbol{\ell}_0, \boldsymbol{\ell}_1 \leftarrow \mathbb{F}_q^m$, let $\ell(u) = \boldsymbol{\ell}_0 + \boldsymbol{\ell}_1 u$, compute $f(u) = F(\ell(u))$; let $\rho_i = F$ for every $i \in [k]$; output $\mathsf{pk}_F = \ell(u), \mathsf{vk}_F = (\ell(u), f(u))$ and $\{\rho_i\}_{i=1}^k$.*

$\mathsf{ProbGen}(\mathsf{pk}_F, \mathbf{x})$: *Choose $a \leftarrow \mathbb{F}_q^*, \alpha \leftarrow \mathbb{F}_q^* \setminus [k], \mathbf{r}_1, \ldots, \mathbf{r}_t \leftarrow \mathbb{F}_q^m$, let $\mathbf{a} = \ell(a)$, $\mathbf{r}_{t+1} = \alpha^{-t-1}(\mathbf{a} - (\mathbf{x} + \sum_{s=1}^t \mathbf{r}_s \alpha^s))$ and $c(u) = \mathbf{x} + \sum_{s=1}^{t+1} \mathbf{r}_s u^s$; choose a cyclic group $\mathbb{G} = \langle g \rangle$ of order q; let $\mathsf{vk}_\mathbf{x} = (g, g^a, \ldots, g^{a^d}, g^\alpha, \ldots, g^{\alpha^{k-1}})$; let $\sigma_i = c(i)$ for every $i \in [k]$; output $\mathsf{vk}_\mathbf{x}$ and $\{\sigma_i\}_{i=1}^k$.*

$\mathsf{Compute}(i, \rho_i, \sigma_i)$: *Parse ρ_i as F and output $\pi_i = F(\sigma_i)$.*

$\mathsf{Verify}(\mathsf{vk}_F, \mathsf{vk}_\mathbf{x}, \{\pi_i\}_{i=1}^k)$: *Interpolate a polynomial $\phi(u)$ of degree $\leq d(t+1)$ such that $\phi(i) = \pi_i$ for all $i \in [k]$. If $g^{\phi(\alpha)} = g^{f(a)}$, output $\phi(0)$; otherwise, output $\perp$.*

and apply a new verification as follows:

$$g^{\phi(\alpha)} = g^{f(a)}. \tag{7}$$

To enable the verification, a key of the form $\mathsf{vk}_{\mathbf{x}} = \{g^{a^i}\} \cup \{g^{\alpha^i}\}$ will be made public.

Proofs for the correctness and t-privacy of Π_4 are identical to those of Π_1 and omitted. Regarding security, we show a reduction from the $(k-1)$-DHI problem to the problem of satisfying (7) with the partial results crafted by an adversary.

Theorem 4. *The scheme Π_4 is t-secure under the $(k-1)$-DHI assumption in $\mathbb{G}$ (see Appendix E).*

B. The Second Construction

In Π_2, not only the client's input $\mathbf{x}$ is secret-shared with a degree $\leq t$ curve $c(u)$ but also a random field element α is secret-shared using a degree $\leq t$ polynomial $b(u)$. Each server evaluates F on a point of the curve and returns both the result and the product of this result with a share of α. The client then interpolates $\phi(u) = F(c(u))$ and $\psi(u) = F(c(u))b(u)$. It accepts the servers' partial results if and only if $\psi(0) = \alpha\phi(0)$, where α is a private key for verification. To obtain a publicly verifiable scheme (Scheme 5), we shall choose a cyclic group $\mathbb{G} = \langle g \rangle$ of prime order q and apply the verification

$$g^{\psi(0)} = g^{\alpha\phi(0)}. \tag{8}$$

To enable the verification, $\mathsf{vk}_{\mathbf{x}} = g^{\alpha}$ is made public.

Scheme 5. The k-Server Scheme Π_5 ($k = (d+1)t+1$)

$\mathsf{KeyGen}(\lambda, F)$: *Let $\rho_i = F$ for every $i \in [k]$. Output $\mathsf{pk}_F = \perp$, $\mathsf{vk}_F = \perp$ and $\{\rho_i\}_{i=1}^k$.*

$\mathsf{ProbGen}(\mathsf{pk}_F, \mathbf{x})$: *Choose $\alpha \leftarrow \mathbb{F}_q^*, \mathbf{r}_1, \ldots, \mathbf{r}_t \leftarrow \mathbb{F}_q^m, \gamma_1, \ldots, \gamma_t \leftarrow \mathbb{F}_q$, let $c(u) = \mathbf{x} + \sum_{s=1}^t \mathbf{r}_s u^s$, $b(u) = \alpha + \sum_{s=1}^t \gamma_s u^s$, and $\sigma_i = (c(i), b(i))$ for all $i \in [k]$. Choose a cyclic group $\mathbb{G} = \langle g \rangle$ of order q. Output $\mathsf{vk}_{\mathbf{x}} = g^{\alpha}$ and $\{\sigma_i\}_{i=1}^k$.*

$\mathsf{Compute}(i, \rho_i, \sigma_i)$: *Parse ρ_i as F, σ_i as $(c(i), b(i))$, compute $v_i = F(c(i)), w_i = v_i b(i)$, output $\pi_i = (v_i, w_i)$.*

$\mathsf{Verify}(\mathsf{vk}_F, \mathsf{vk}_{\mathbf{x}}, \{\pi_i\}_{i=1}^k)$: *Interpolate a polynomial $\phi(u)$ of degree $\leq dt$ such that $\phi(i) = v_i$ for all $i \in [k]$ and a polynomial $\psi(u)$ of degree $\leq (d+1)t$ such that $\psi(i) = w_i$ for all $i \in [k]$. If $g^{\psi(0)} = g^{\alpha\phi(0)}$, output $\phi(0)$; otherwise, output $\perp$.*

Proofs for the correctness and t-privacy of Π_4 are exactly identical to those of Π_2 and omitted. Regarding security, we show a reduction from the DLog problem to the problem of satisfying (8) with the partial results crafted by an adversary.

Theorem 5. *The scheme Π_5 is t-secure under the DLog assumption in $\mathbb{G}$ (see Appendix F).*

V. Performance Evaluation

In this section, we evaluate the performance of our schemes in Section III and IV. Our evaluation will cover not only the client's *time cost* but also its *monetary cost*, in executing the schemes. In particular, for monetary cost, although consumptions of various computing resources could be accounted, our evaluation will be simplified and mainly base on the running times of all algorithms.

We denote respectively with $T_{\mathrm{k}}, T_{\mathrm{p}}, T_{\mathrm{c}}^i, T_{\mathrm{v}}$ and T_{n} the time needed by the client running $\mathsf{KeyGen}(\lambda, F)$, the client running $\mathsf{ProbGen}(\mathsf{pk}_F, \mathbf{x})$, the ith server running $\mathsf{Compute}(i, F, \sigma_i)$, the client running $\mathsf{Verify}(\mathsf{vk}_F, \mathsf{vk}_{\mathbf{x}}, \{\pi_i\}_{i=1}^k)$, and the client locally computing $F(\mathbf{x})$. Let T_{c} and T_{c}^* be the *total* and *maximum* running time of the servers, i.e.,

$$T_{\mathrm{c}} = \sum_{i=1}^{k} T_{\mathrm{c}}^i; \quad T_{\mathrm{c}}^* = \max\{T_{\mathrm{c}}^1, T_{\mathrm{c}}^2, \ldots, T_{\mathrm{c}}^k\}. \tag{9}$$

Our evaluation will focus on the following question: ($\triangle$) *In terms of the time/monetary cost, under what conditions and to what extent the client will benefit from using MSVC*?

A. Time Cost

Among the five schemes, Π_1 and Π_4 have a non-empty KeyGen, which is heavier than the native computation of $F(\mathbf{x})$. In all schemes, the client has to prepare its input $\mathbf{x}$ for computation and verify the servers' partial results, by running ProbGen and Verify respectively.

In the field of verifiable computation, many pioneer works such as [51], [79], [80] have suggested that the one-time cost T_{k} of executing $\mathsf{KeyGen}(\lambda, F)$ can be *amortized* when the same function F is evaluated many times, and a scheme is *outsourceable* if the total running time of ProbGen and Verify is far smaller than that of the native computation, i.e.,

$$T_{\mathrm{p}} + T_{\mathrm{v}} \ll T_{\mathrm{n}}. \tag{10}$$

Following this idea, a proof for (10) appears in Section G and shows that our schemes are all outsourceable provided that

$$kmt^2 + k^3 + (k+d)\lambda = o(nd), \tag{11}$$

where $n = \binom{m+d}{d}$. Given a security parameter λ, the parameters (q, m, d, t) in our schemes may be chosen such that

$$q \approx 2^{O(\lambda)}, \ m = \mathrm{poly}(\lambda), \ d = O(1), \ t = O(1), \tag{12}$$

to satisfy (11) and thus result in outsourceable schemes.

In terms of time cost, we believe that a client may benefit from MSVC, only if the client spends less time in learning $F(\mathbf{x})$, compared with locally computing $F(\mathbf{x})$. While the requirement of (10) is interesting and may realize this idea in a setting where the servers' computations take essentially no time (i.e., $T_{\mathrm{c}} = 0$), it is generally not realistic.

In MSVC, some of the servers may finish their computations earlier than the others and the client may receive their partial results earlier. However, the client will not be able to verify until all partial results arrive. If all servers initiate their computations simultaneously, the time that has to be spent by the client before actually learning $F(\mathbf{x})$ will be $T_{\mathrm{p}} + T_{\mathrm{v}} + T_{\mathrm{c}}^*$, where T_{c}^* is the maximum of the servers' running time. In the amortized model of [51], [79], [80], a client will benefit from MSVC with reduced time cost if and only if

$$T_{\mathrm{p}} + T_{\mathrm{v}} + T_{\mathrm{c}}^* < T_{\mathrm{n}}. \tag{13}$$

As per (13), in terms of time cost, the aforementioned question ($\triangle$) may be answered by analyzing the parameter

$$\mathtt{Rt} = (T_{\mathrm{p}} + T_{\mathrm{v}} + T_{\mathrm{c}}^*)/T_{\mathrm{n}}, \tag{14}$$

the ratio of the client's time cost in MSVC to its time cost of locally computing $F(\mathbf{x})$. In particular, the client may benefit from MSVC if and only if $\mathtt{Rt} < 1$; the smaller the $\mathtt{Rt}$ is, the more the client will benefit from using MSVC. Ideally, we would like $\mathtt{Rt}$ to be as small as possible.

The actual values of $\mathtt{Rt}$ may depend on factors such as the scale of the problem instance $(F, \mathbf{x})$. As there is a gap between the computing speed of the client and that of every server, $\mathtt{Rt} < 1$ may be easily satisfied by large problem instances. Ideally, we would like to have $\mathtt{Rt} < 1$ for small problem instances as well. Given (q, d), the scale of the problem $(F, \mathbf{x})$ may be measured with m, the number of variables in F. In order to answer the question of $(\triangle)$, we need to decide an m_0 such that $\mathtt{Rt} < 1$ for all $m > m_0$, and analyze how $\mathtt{Rt}$ will vary as m is increasing.

B. Monetary Cost

Cloud computing aims to cut costs. Computing resources provided by the cloud servers are usually much cheaper than those owned by the client. A main motivation for the client to outsource computations is reducing the monetary cost.

By locally computing $F(\mathbf{x})$, the client has to spend time T_{n} and the monetary cost of this local computation may be quite high. By using MSVC, the amount of local computations is reduced to $T_{\mathrm{p}} + T_{\mathrm{v}}$. However, the client has to additionally pay for the T_{c} unit of cloud servers' computations. Let $\mathtt{Rp}$ be the ratio of the price of the client's local computation to that of the cloud servers' computation. Then the client's total monetary cost in MSVC will be equivalent to $T_{\mathrm{p}} + T_{\mathrm{v}} + T_{\mathrm{c}}/\mathtt{Rp}$ units of its local computation. In terms of monetary cost, the client will benefit from using MSVC if and only if

$$T_{\mathrm{p}} + T_{\mathrm{v}} + T_{\mathrm{c}}/\mathtt{Rp} < T_{\mathrm{n}}. \tag{15}$$

As per (15), in terms of monetary cost, the aforementioned question $(\triangle)$ may be answered by analyzing the parameter

$$\mathtt{Rp}^{\star} = T_{\mathrm{c}}/(T_{\mathrm{n}} - T_{\mathrm{p}} - T_{\mathrm{v}}), \tag{16}$$

the least value of $\mathtt{Rp}$ such that (15) holds for all $\mathtt{Rp} > \mathtt{Rp}^{\star}$.

The actual values of $\mathtt{Rp}^{\star}$ may depend on factors such as the scale of $(F, \mathbf{x})$. The smaller the $\mathtt{Rp}^{\star}$ is, the easier it is for the client to benefit from using MSVC. We are mostly interested in the values of $\mathtt{Rp}^{\star}$ when $m > m_0$, in order to decide when the client will benefit in terms of both the time cost and the monetary cost. More precisely, we need to analyze how $\mathtt{Rp}^{\star}$ will vary as $m > m_0$ is increasing, and decide an upper bound, say $\mathtt{Rp}_0^{\star}$, of $\mathtt{Rp}^{\star}$ for all $m > m_0$.

C. Break-Even Points

In Π_1 and Π_4, the client has to perform a one-time execution of $\mathsf{KeyGen}(\lambda, F)$. Although these schemes are outsourceable according to (10) and under the condition of (11), the client will not really benefit from using them unless the same function F is evaluated multiple times.

If the time T_{k} needed by $\mathsf{KeyGen}(\lambda, F)$ is accounted and the function F is evaluated at N inputs, then the client's time cost per input in MSVC will be $T_{\mathrm{p}} + T_{\mathrm{v}} + T_{\mathrm{c}}^{*} + T_{\mathrm{k}}/N$ and the client may benefit from using MSVC if and only if

$$T_{\mathrm{p}} + T_{\mathrm{v}} + T_{\mathrm{c}}^{*} + T_{\mathrm{k}}/N < T_{\mathrm{n}}. \tag{17}$$

Given an m_0 from Section V-A, we call the smallest integer $N = \mathtt{Nt}$ such that (17) holds for all $m > m_0$ a *break-even point* for the client's time cost. On the other hand, the client's monetary cost per input will be $T_{\mathrm{p}} + T_{\mathrm{v}} + T_{\mathrm{c}}/\mathtt{Rp} + T_{\mathrm{k}}/N$ and the client may benefit from using MSVC if and only if

$$T_{\mathrm{p}} + T_{\mathrm{v}} + T_{\mathrm{c}}/\mathtt{Rp} + T_{\mathrm{k}}/N < T_{\mathrm{n}}. \tag{18}$$

Given an m_0 from Section V-A and an $\mathtt{Rp}_0^{\star}$ from Section V-B, we call the smallest integer $N = \mathtt{Nm}$ such that (18) holds for all $m > m_0$ and $\mathtt{Rp} = \mathtt{Rp}_0^{\star}$ a *break-even point* for the client's monetary cost. Ideally, we would like $\mathtt{Nt}$ and $\mathtt{Nm}$ to be as small as possible. In order to answer the question of $(\triangle)$, we need to analyze how $\mathtt{Nt}$ and $\mathtt{Nm}$ vary as well.

D. Our Implementation

In this section, we describe our implementations in detail. The main parameters in our schemes include λ, q, m, d and t, where λ is a security parameter, (q, m, d) specify a function family $\mathcal{P}(q, m, d)$, and t is the threshold for privacy and security. We will describe both the choices of these parameters and the choice of the cyclic group $\mathbb{G}$ in Π_4 and Π_5. We will also describe the libraries and platforms with which our experiments will be conducted.

Security parameters, cyclic groups and finite fields. Our implementations achieve λ-bit security for $\lambda = 128$. Our privately verifiable schemes Π_1, Π_2 and Π_3 are ϵ-secure for $\epsilon = O(1/q)$. In order to assure 128-bit security, a sensible choice of q would be $q \approx 2^{128}$. In particular, we shall choose $q = 2^{128} + 51$ in our implementation of these schemes. Then the servers in Π_1 and Π_2 will perform polynomial evaluations over a 129-bit finite field $\mathbb{F}_q$, except in Π_3 where the computation is done over a 257-bit finite field $\mathbb{F}_{q^2}$. The security of our publicly verifiable schemes Π_4 and Π_5 relies on the DLog assumption in a cyclic group $\mathbb{G}$ of prime order q. For the sake of efficiency, we choose $\mathbb{G}$ to be the ristretto255 group [43], a cyclic group of prime order $q = 2^{252} + 27742317777372353535851937790883648493$ that is obtained from Bernstein's Curve25519 [18] by applying Hamburg's Decaf point compression technique [64] for constructing prime-order groups. It is believed that such a group provides 128-bit of security strength [41]. As a result, we have that $q \approx 2^{253}$ and the servers in Π_4 and Π_5 will perform polynomial evaluations over a 253-bit finite field.

Libraries. The field operations in $\mathbb{F}_q$ and $\mathbb{F}_{q^2}$ are implemented with FLINT (v2.8.0), the fast library for number theory, which is based on GMP. The ristretto255 cyclic group $\mathbb{G}$ is implemented with libsodium (v1.0.18). The programming of all schemes is done in C.

Platforms. We perform each server's computation (i.e., the Compute in MSVC) on a virtual machine VM_s that runs a Ubuntu 20.04 operating system with a single core of a 3.60

GHz CPU and 8GB of RAM; and perform both the client's computation (i.e., KeyGen, ProbGen and Verify) and the native computation on a virtual machine VM_c that runs a Ubuntu 20.04 operating system with a single core of a 1.9 GHz CPU and 4GB of RAM. With these choices we try to mimic the gap of computing speed between the client's device (e.g., a netbook) and the servers' machines.

Degree of polynomials. In theory there is no limitation on the degree d of the polynomials that can be outsourced by MSVC. However, due to three reasons, our schemes are more suitable for outsourcing *low-degree* polynomials.

- First, the number k of servers required by each of our schemes is linear in d. A large d would result in a large number of servers. While $k \geq dt + 1$ is necessary in order to provide information-theoretic t-privacy with non-communicating servers, and thus one of our schemes (i.e., Π_3) is already optimal in terms of the number of servers and can hardly be improved, it may be still undesirable to use too many servers. Therefore, we prefer to use the proposed MSVC to evaluate polynomials of small degree d, in order to limit the number of required servers.
- Second, the Theorem 8 in Appendix G shows that our schemes are outsourceable (i.e., $T_{\mathrm{p}}+T_{\mathrm{v}} \ll T_{\mathrm{n}}$), provided that (11) is satisfied. As we are working over finite fields of order $q \approx 2^{O(\lambda)}$, many choices of (m, d, t) would meet the requirement of (11), e.g., $(m, d, t) = (\mathrm{poly}(\lambda), O(1), O(1))$ or $(m, d, t) = (O(1), \mathrm{poly}(\lambda), O(1))$. Among these choices, the client will benefit most from the former one, which requires $d = O(1)$.
- Third, comparing with high-degree polynomials, the low-degree ones are preferred in many scenarios. For example, the information-theoretic PIR protocols [37], [94] require low-degree polynomial interpolations and give quite practical communication complexity; in the polynomial regression based curve fitting, low-degree polynomials are usually preferred and approached first [58], and require low-degree multivariate polynomial evaluations.

Many existing works experiment with low-degree polynomials. For example, the degree-2 and degree-3 polynomial evaluations have been the focus of the implementations of [39], [85]. Pinocchio [79] and Trinocchio [81] also experiment with polynomials of degree $6, 8$ and 10. In Section V-E, we will provide a detailed analysis of the parameters $\mathtt{Rt}, \mathtt{Rp}, \mathtt{Nt}$ and $\mathtt{Nm}$ with the experimental results of degree-2 polynomial evaluations. However, besides $d = 2$, we will also consider polynomials of degree $d = 4, 8, 16, 32, 64$ and show slightly rough analysis of the same parameters, in order to have a more complete overview of the performance.

Threshold for privacy and security. Our schemes in Section III and IV are both information-theoretically t-private and t-secure. Any choice of the threshold t may have its pros and cons. Besides privacy and security, the t also has direct impact on the number of required servers and the client's computation complexity. A larger t not only implies stronger privacy and security, but also results in a larger number k of servers and a heavier workload of the client. A smaller t would imply less servers and faster verification, but also lower security level. How to choose t may not only depend on the client's preference, but also depend on the actual situation of how many servers can collude with each other. In our model and constructions, the servers *do not need to communicate* with each other, in order for the client to reconstruct $F(\mathbf{x})$. Such model and constructions allow the client to keep every server anonymous from the others. Then it would be difficult for even two servers to collude. Thus, it seems not unreasonable to choose a small t. Based on these observations, we will experiment with relatively small t such as $t = 1, 2, 3$.

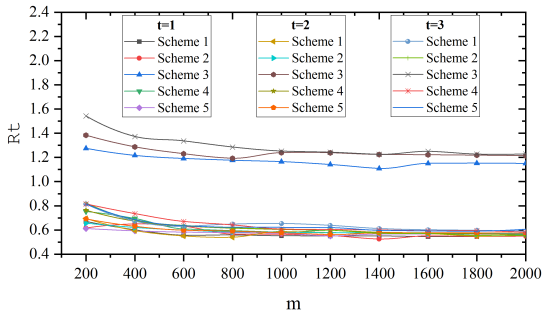

Fig. 1. The value of $\mathtt{Rt} = (T_{\mathrm{p}} + T_{\mathrm{v}} + T_{\mathrm{c}}^*)/T_{\mathrm{n}}$ in degree-2 polynomial evaluations ($d = 2$, $t = 1, 2, 3$ and $m = 200, 400, \ldots, 2000$ in all schemes)

E. Experimental Results: Polynomials of Degree Two

As stated in Section V-D, for degree-2 polynomial evaluations, we choose $t = 1, 2, 3$ and choose q to be either a 129-bit prime or a 253-bit prime. Given q, the scale of a problem instance $(F, \mathbf{x}) \in \mathcal{P}(q, m, 2) \times \mathbb{F}_q^m$ can be measured with m, the number of variables in F. In the experiments, we choose $m = 200, 400, \ldots, 2000$ and analyze how $\mathtt{Rt}, \mathtt{Rp}$ (for all schemes) and $\mathtt{Nt}, \mathtt{Nm}$ (for Π_4 and Π_5) vary as m is increasing. For each (t, m), we run the KeyGen, ProbGen and Verify on VM_c and run the Compute on VM_s, respectively. We also evaluate $F(\mathbf{x})$ on VM_c. We repeat each execution 5 times and record the average CPU time such that the standard deviations are within 5% of the means.

Time cost. Fig. 1 plots the experimental results of $\mathtt{Rt}$ for the five MSVC schemes and for all settings of $(t, m) \in \{1, 2, 3\} \times \{200, 400, \ldots, 2000\}$. For Π_1, Π_2, Π_4 and Π_5, it shows that $\mathtt{Rt} < 1$ starting from $m = 200$ ($m_0 = 200$) and $\mathtt{Rt}$ tends to be < 0.6 when m is large. Thus, in terms of time cost, the client may benefit from using these schemes when $m \geq 200$. Furthermore, the client may bring its time cost in these schemes down to $0.6T_{\mathrm{n}}$ when m is large enough. On the other hand, for Π_3, Fig. 1 shows that $\mathtt{Rt} > 1$ for all m and $\mathtt{Rt}$ tends to be ≈ 1.2 when m is large, which leads to a seemingly surprising conclusion that the client will not benefit from using Π_3. However, we will see shortly that this is not always true and whether one will benefit from Π_3 largely depends on the gap between the computing speed of the client and that of the servers. On one hand, when m is large enough, we always have that $T_{\mathrm{p}}+T_{\mathrm{v}} = o(T_{\mathrm{n}})$ in all schemes. As a result, we will roughly have $\mathtt{Rt} \approx T_{\mathrm{c}}^*/T_{\mathrm{n}}$. On the other hand, each server in Π_1, Π_2, Π_4 and Π_5 needs to evaluate F at a random point of

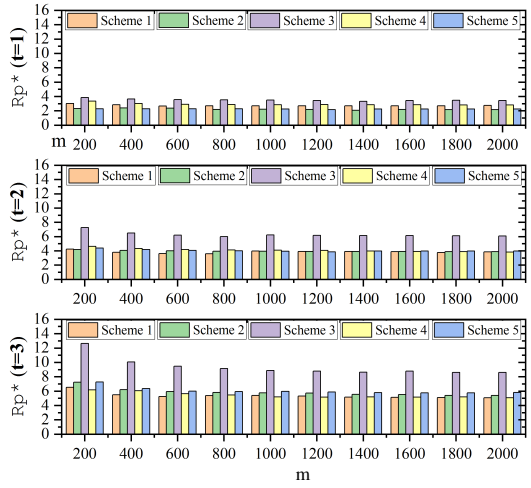

Fig. 2. The value of $\texttt{Rp}^\star = T_\mathrm{c}/(T_\mathrm{n} - T_\mathrm{p} - T_\mathrm{v})$ for degree-2 polynomial evaluations ($d = 2$, $t = 1, 2, 3$ and $m = 200, 400, \ldots, 2000$ in all schemes)

$\mathbb{F}_q^m$ and the workload is roughly equal to that of computing $F(\mathbf{x})$. In our experiments, T_c^* is needed by VM_s evaluating F once and T_n is needed by VM_c evaluating $F(\mathbf{x})$. Note that the ratio of the speed of VM_c to that of VM_s is $1.9/3.6 \approx 0.53$. Therefore, in the four schemes, given that the client and each server have the same workload, we would have that $\texttt{Rt} \to 0.53$ as $m \to \infty$. This fact has been partially reflected in Fig. 1 as $\texttt{Rt} \leq 0.6$ starting from $m = 1600$. In Π_3, the situation is quite different. Each server has to evaluate F once, but at a random point of $\mathbb{F}_{q^2}^m$. Based on the fast integer multiplication of FLINT, the workload of each server is roughly twice of that of the client. With our virtual machines, one would have that $\texttt{Rt} \to 1.06$ as $m \to \infty$, which is partially reflected in Fig. 1. In order for $\texttt{Rt} < 1$ in Π_3, one requires that the ratio of the speed of VM_s to that of VM_c is > 2.

Monetary cost. Fig. 2 plots the experimental results of Rp for the five schemes and for all settings of $(t, m) \in \{1, 2, 3\} \times \{200, 400, \ldots, 2000\}$. For every $t \in \{1, 2, 3\}$, it shows that the $\texttt{Rp}^\star$ value of every scheme is decreasing as m is increasing. In particular, when $m = 2000$, the $\texttt{Rp}^\star$ values are as shown in Table III. Therefore, for every $t \in \{1, 2, 3\}$ and $\Pi \in \{\Pi_1, \Pi_2, \Pi_3, \Pi_4, \Pi_5\}$, as long as the ratio of the price of VM_c's computation to that of VM_s's computation is larger than the (t, Π)-entry of Table III, the client will benefit in terms of monetary cost by using MSVC to evaluate a 2000-variate polynomial of degree 2. When $m \to \infty$, we expect the $\texttt{Rp}^\star$ values of every scheme to converge. When m is large enough, we have that $T_\mathrm{p} + T_\mathrm{v} = o(T_\mathrm{n})$ and thus

$$\texttt{Rp}^\star = T_\mathrm{c}/(T_\mathrm{n} - T_\mathrm{p} - T_\mathrm{v}) \approx T_\mathrm{c}/T_\mathrm{n} \approx kT_\mathrm{c}^*/T_\mathrm{n} \approx k \cdot \texttt{Rt}, \quad (19)$$

where k is the number of required servers in each scheme. Given the estimated Rt limits (≈ 0.53 for $\Pi_1, \Pi_2, \Pi_4, \Pi_5$ and ≈ 1.06 for Π_3), the estimated $\texttt{Rp}^\star$ limits (see Table IV) can be easily computed as well. A simple comparison shows that our experimental results of $\texttt{Rp}^\star$ (as shown in Table II) for $m = 2000$ are both lower bounded by and quite close to the estimated limits of $\texttt{Rp}^\star$ (as shown in Table IV). As per (19),

TABLE III
The experimental results of $\texttt{Rp}^\star$ ($m = 2000$)

$\texttt{Rp}^\star$	Π_1	Π_2	Π_3	Π_4	Π_5
$t = 1$	2.8	2.2	3.5	2.9	2.3
$t = 2$	3.9	3.9	6.1	3.9	4.0
$t = 3$	5.1	5.4	8.7	5.1	5.9

TABLE IV
The estimated limits of $\texttt{Rp}^\star$ ($m \to \infty$)

$\texttt{Rp}^\star$	Π_1	Π_2	Π_3	Π_4	Π_5
$t = 1$	2.6	2.1	3.2	2.6	2.1
$t = 2$	3.7	3.7	5.2	3.7	3.7
$t = 3$	4.8	5.3	7.4	4.8	5.3

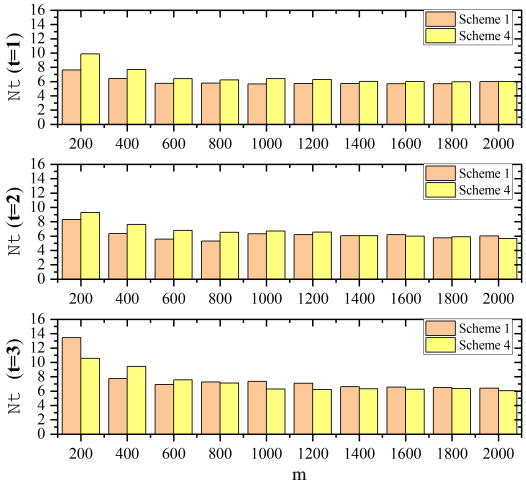

Fig. 3. The value of $\texttt{Nt} = T_\mathrm{k}/(T_\mathrm{n} - T_\mathrm{c}^* - T_\mathrm{p} - T_\mathrm{v})$ for degree-2 polynomial evaluations ($d = 2$, $t = 1, 2, 3$ and $m = 200, 400, \ldots, 2000$ in all schemes)

the $\texttt{Rp}^\star$ limits are linear in k and the Rt limits, respectively. The client will benefit most from MSVC when k is small and the computing gap between client and servers is large.

Break-even points. Only two of the schemes (Π_1 and Π_4) have a non-empty KeyGen algorithm. Fig. 3 plots the experimental results of Nt for these schemes and for all settings of $(t, m) \in \{1, 2, 3\} \times \{200, 400, \ldots, 2000\}$. It shows that $\texttt{Nt} \leq 7$ when m is large enough. That is, if T_k is accounted, then in terms of time cost the client will benefit from evaluating F at least 7 times. The actual values of Nm depend on our choices of the $\texttt{Rp}_0^\star$, an upper bound of the $\texttt{Rp}^\star$ in (16). If we choose $\texttt{Rp}_0^\star = 6$ in both schemes, then for $t = 1, 2, 3$ we will respectively have $\texttt{Nm} \leq 5, 8, 20$ in these schemes. That is, in terms of monetary cost the client will benefit from evaluating F at least $5, 8$, and 20 times, respectively.

F. Experimental Results: Polynomials of Higher Degree

We also consider higher-degree polynomials to have a more complete overview of performance. In these experiments, we choose $d = 4, 8, 16, 32, 64$, choose $t = 1$ and choose q to be either the 129-bit prime or the 253-bit prime from Section V-D. Given (q, d), the scale of a problem instance $(F, \mathbf{x}) \in \mathcal{P}(q, m, d) \times \mathbb{F}_q^m$ can be measured with m. However, for our choice of d, a small m may result in a huge $n = \binom{m+d}{d}$, the number of coefficients in an m-variate polynomial of degree d. For example, for $(m, d) = (32, 10)$, we have $n = 1471442973$, which is too large for a real application. Instead of using m, we

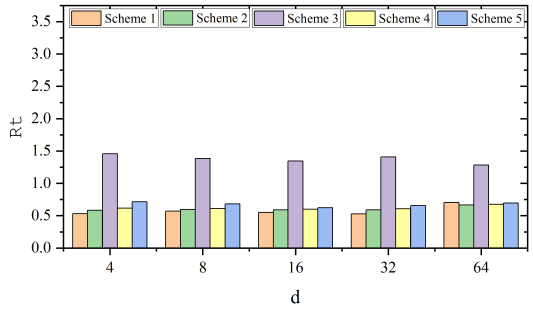

Fig. 4. The value of $\texttt{Rt} = (T_\text{p} + T_\text{v} + T_\text{c}^*)/T_\text{n}$ in higher-degree polynomial evaluations ($d = 4, 8, 16, 32, 64$, $t = 1$ and $n = 10^6$ in all schemes)

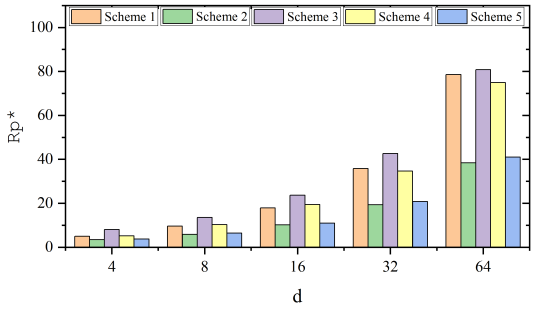

Fig. 5. The value of $\texttt{Rp}^\star = T_\text{c}/(T_\text{n} - T_\text{p} - T_\text{v})$ for higher-degree polynomial evaluations ($d = 4, 8, 16, 32, 64$, $t = 1$ and $n = 10^6$ in all schemes)

will use the number of coefficients in F to measure the scale of $(F, \mathbf{x})$. We try to experiment with homogeneous polynomials that have 10^6 coefficients. In particular, we choose $(d, m) = (4, 69), (8, 18), (16, 10), (32, 7), (64, 6)$ such that it is possible to have a polynomial with 10^6 coefficients. For each setting of (d, m), we run the KeyGen, ProbGen and Verify on VM_c and run the Compute on VM_s. We also execute the native computation of $F(\mathbf{x})$ on VM_c. We repeat each execution 5 times and record the average CPU time such that the standard deviations are within 5% of the means.

Time cost. Fig. 4 plots the experimental results of Rt for the five schemes and for all settings of (d, m). For Π_1, Π_2, Π_4 and Π_5, it shows that $\texttt{Rt} < 0.7$. Thus, in terms of time cost, the client may benefit from using these schemes when the polynomial has $\geq 10^6$ coefficients. For Π_3, it shows that $\texttt{Rt} > 1$. Comparing with the degree-2 setting, these schemes have quite similar performances for every $d \in \{4, 8, 16, 32, 64\}$.

Monetary cost. Fig. 5 plots the experimental results of $\texttt{Rp}^\star$ for the five schemes and for all settings of (d, m). It shows that the $\texttt{Rp}^\star$ value of every scheme is increasing as d is increasing. In particular, these values are lower bounded by and quite close to the estimated limits of $\texttt{Rp}^\star$ in Table V.

VI. Applications

A. Curve Fitting with Private Data Points

Given a set of data points $\{(x_i, y_i)\}_{i=1}^m$ that result from an experiment or a real-life scenario, we may assume there is a function $y = f(x)$ that passes through the data points and perfectly represents the quantity of interest at all non-data points. Curve fitting [5] is the process of constructing a curve or function that has the best fit to the data points. When the data points exhibit a significant degree of error, the strategy is to derive a single curve that represents the general trend of the data and may be realized with *least squares regression* (LSR) [34]. When the data points are very precise, the strategy is to fit a curve or a series of curves that pass directly through each of the points and may be realized with *interpolation* [34].

TABLE V
The estimated limits of $\texttt{Rp}^\star$ ($m \to \infty$)

$\widetilde{\texttt{Rp}}$	Π_1	Π_2	Π_3	Π_4	Π_5
$d = 4$	4.8	3.2	5.3	4.8	3.2
$d = 8$	9.0	5.3	9.5	9.0	5.3
$d = 16$	17.5	9.5	18.0	17.5	9.5
$d = 32$	34.4	18.0	35.0	34.4	18.0
$d = 64$	68.4	35.0	68.9	68.4	35.0

In LSR, to fit a degree-d polynomial $y = \sum_{j=0}^d a_j x^j$, the idea is to minimize $S = \sum_{i=1}^m (y_i - \sum_{j=0}^d a_j x_i^j)^2$, the sum of the squares of the residuals between the measured values $\{y_i\}_{i=1}^m$ and the values calculated with the model. The best fit is obtained by setting $\frac{\partial S}{\partial a_i} = 0$ for $0 \leq i \leq d$ and then solving a linear equation system in $\{a_i\}_{i=0}^d$. In particular, the solution of each a_i may be represented as a rational function of the data points $\{(x_i, y_i)\}_{i=1}^m$. For example, for $d = 1$, both the numerator and the denominator of the rational function are polynomials of degree ≤ 3 in the $2m$ variables $\{(x_i, y_i)\}_{i=1}^m$. In general, fitting with a degree-d polynomial requires one to evaluate polynomials whose degrees are determined by d. In particular, small values of d are usually preferred and approached first [58]. Therefore, low-degree polynomial evaluations will be frequently used. The same situation occurs in multiple linear regression and interpolation.

In real-life scenarios, the data points may contain highly sensitive personal information such as the patients' brain white matter microstructure [77] and inspiratory pressure [65]. Our schemes allow the client to outsource curve fitting with private data points to cloud, without leaking the personal information.

B. PIR with Cheating Detection

PIR [37] is a cryptographic primitive that has important real applications [40]. It allows a client to retrieve an item F_i from a database $F = (F_1, F_2, \ldots, F_n)$ and reveals no information about i to the database server(s). In a t-private k-server PIR [94], each server keeps a copy of F and answers to the client's query, the client reconstructs F_i from all servers' answers, and any t servers cannot learn i. The communication complexity of PIR is the total number of communicated bits for retrieving one bit from the database. The low-degree polynomial interpolation based t-private k-server PIR [37], [94] have nontrivial communication complexity of $O(n^{1/\lfloor (2k-1)/t \rfloor})$. Despite of efficient communication, PIR has not been widely deployed due to high computational complexity [14]. One can offload the PIR servers' computations to cloud [66], which however may be untrusted and respond incorrectly. It is an interesting problem [13] to deal with dishonest PIR servers.

The t-private t-Byzantine robust k-server PIR schemes ((t, t, k)-BRPIR) of [15], [73] allow the client to reconstruct correctly, even if t servers provide wrong answers. By using error-correcting techniques, they provide *information-theoretic* security and enable the *identification* of cheating servers. In

particular, the techniques of [15], [73] may result in (t,t,k)-BRPIR with communication complexity $O(n^{1/(\lfloor(2k-1)/t\rfloor-4)})$, which is the best to date for general t. There are also schemes [44], [55] that are mostly suitable for retrieving $O(n^{1/2})$ bits per query and have communication complexity $O(n^{1/2})$.

Identification of cheating servers may be overly strong in many scenarios such as private media browsing [62], where a database of movies is stored on several non-communicating servers and the client hides its media diet by using PIR to retrieve a movie. In this scenario, it may suffice for the client to detect the existence of cheating and then refuse to pay. If we consider this relaxed security, then it may be possible to obtain meaningful efficiency improvements.

In the literature, t-private k-server PIR schemes [94] with best communication complexity for general t are based on low-degree multivariate polynomial evaluations. Given a prime q, the database F may be regarded as a vector over $\mathbb{F}_q$. Let $m,d>0$ be integers such that $\binom{m}{d}\geq n$. With an injection $E:[n]\to\{0,1\}^m$ that maps $[n]$ to 0-1 vectors of weight d, F may be encoded as $F(x_1,\ldots,x_m)=\sum_{j=1}^{n}F_j\prod_{s:E(j)_s=1}x_s$, a polynomial in $\mathcal{P}(q,m,d)$ such that $F(E(i))=F_i$ for all $i\in[n]$. Then the work of retrieving F_i can be reduced to the work of computing $F(\mathbf{x})$ at $\mathbf{x}=E(i)$. The MSVC schemes in Section III and IV directly translate to t-private k-server PIR schemes that can detect the cheating of t servers. In particular, Π_3 gives a scheme with communication complexity $O(n^{1/\lfloor(k-1)/t\rfloor})$. By using the partial derivative-based technique of [94], this communication complexity can be further reduced to $O(n^{1/\lfloor(2k-1)/t\rfloor})$, which is asymptotically better than the schemes of [15], [73]. If we consider computational verifiability, then Π_4 and Π_5 would yield PIR schemes that allow public detection of cheating.

Theorem 6. *Let $\mu(k,t)=\max\{\lfloor\frac{k-1}{t+1}\rfloor,\lfloor\frac{k-1}{t}\rfloor-1\}$. There is a t-private k-server PIR scheme that allows public detection of cheating by t servers with communication complexity $O(n^{\frac{1}{\mu(k,t)}})$. (see Appendix H)*

By using the partial derivative-based technique of [94], the number of servers in Theorem 6 may be halved.

Theorem 7. *Let $\nu(k,t)=\max\{\lfloor\frac{2k-1}{t+1}\rfloor,\lfloor\frac{2k-1}{t}\rfloor-1\}$. There is a t-private k-server PIR that allows public detection of cheating by t servers with communication complexity $O(n^{\frac{1}{\nu(k,t)}})$.*

Comparison with BRPIR. As $\nu(k,t)>\lfloor(2k-1)/t\rfloor-4$, the PIR in Theorem 7 is more efficient than the (t,t,k)-BRPIR of [15], [73], in terms of communication. As the price, the security is relaxed to cheating detection.

Comparison with PIR for honest servers. Comparing with PIR for honest servers, PIR with cheating detection needs additional cost. We implement the t-private k-server PIR (wyPIR) of [94] and the PIR from Theorem 6 (a Π_4-based PIR4 and a Π_5-based PIR5), using the security parameters, cyclic groups, libraries and platforms from Section V-D. We choose $t=1$, $(d,m)\in\{(2,1415),(3,183),(4,72),(5,44),(6,33),(7,28)\}$, and experiment with a database of $n=10^6$ blocks of 257 bits. Fig. 6 plots the maximum and total running time of servers,

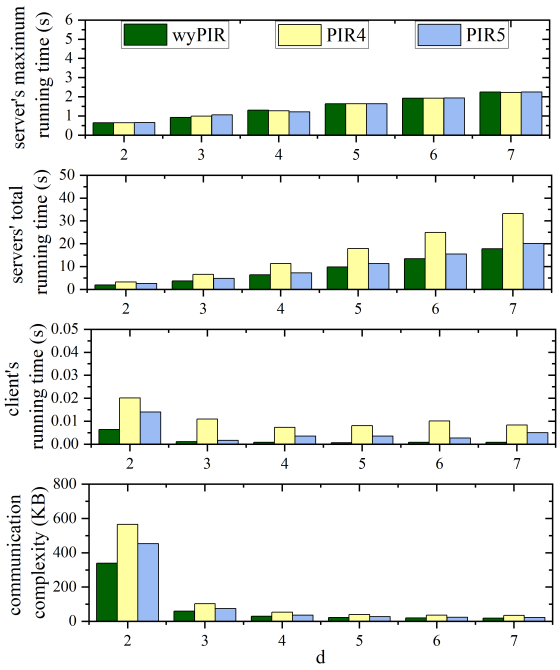

Fig. 6. Cost for retrieving a 257-bit block from a database 10^6 blocks (wyPIR is the baseline protocol [94]; PIR4 and PIR5 are based on Π_4 and Π_5 respectively; all without using the partial derive technique)

the total running time of the client, and the communication complexity in each scheme. It shows that:

- The maximum running time of servers in all all schemes is roughly equal to each other;
- The total running times of servers in PIR4 and PIR5 are within 2 and 1.5 times that of wyPIR, respectively;
- The client's running time in PIR4 and PIR5 are within 15 and 6 times that of wyPIR, respectively; and
- The communication complexities of PIR4 and PIR5 are within 2 and 1.5 times that of wyPIR, respectively.

On the number of servers. In order to use degree-d polynomial evaluations to do t-private PIR with the same communication complexity $O(n^{1/d})$, PIR4 and PIR5 use $d(t+1)+1$ and $(d+1)t+1$ servers, respectively. They are more than the $dt+1$ servers used by wyPIR. Furthermore, PIR4 uses less servers than PIR5 if and only if $d<t$.

VII. Conclusions

In this paper, we define a new MSVC model and construct five MSVC schemes for outsourcing low-degree polynomials over a finite field, of which three are (information-theoretically) privately verifiable and two are publicly verifiable. All schemes are publicly delegatable, information-theoretically private, and outsourceable, and have highly efficient server computations. Our schemes yield multi-server PIR schemes that can detect the existence of cheating.

Acknowledgment. The authors would like to thank the anonymous reviewers for their insightful comments. The work was supported by Natural Science Foundation of Shanghai under grant 21ZR1443000 and Singapore Ministry of Education under grant RG12/19.

REFERENCES

[1] S. Agrawal and D. Boneh, "Homomorphic MACs: MAC-based integrity for network coding," in *ACNS*, 2009.

[2] P. Ananth, N. Chandran, V. Goyal, B. Kanukurthi, and R. Ostrovsky, "Achieving privacy in verifiable computation with multiple servers-without FHE and without pre-processing," in *PKC*, 2014.

[3] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETIhome: an experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, 2002.

[4] B. Applebaum, Y. Ishai, and E. Kushilevitz, "From secrecy to soundness: efficient verification via secure computation," in *ICALP*, 2010.

[5] S. Arlinghaus, *Practical handbook of curve fitting (1st Edition)*. CRC Press, 1994.

[6] L. Babai, "Trading group theory for randomness," in *STOC*, 1985.

[7] L. Babai, L. Fortnow, L.A. Levin, and M. Szegedy, "Checking computations in polylogarithmic time," in *STOC*, 1991.

[8] L. Babai, L. Fortnow, and C. Lund, "Non-deterministic exponential time has two-prover interactive protocols," in *FOCS*, 1990.

[9] M. Backes, D. Fiore, and R. M. Reischuk, "Verifiable delegation of computation on outsourced data," in *CCS*, 2013.

[10] M. Barbosa, D. Catalano, and D. Fiore, "Labeled homomorphic encryption-scalable and privacy-preserving processing of outsourced data," in *ESORICS*, 2017.

[11] O. Barkol and Y. Ishai, "Secure computation of constant-depth circuits with applications to database search problems," in *CRYPTO*, 2005.

[12] O. Barkol, Y. Ishai, and E. Weinreb, "On d-multiplicative secret sharing," *J. Cryptology*, vol. 23, no. 4, 2010.

[13] A. Beimel, "Private information retrieval: a primer," *www.cs.bgu.ac.il /beimel/ Papers/PIRsurvey.ps*, 2008

[14] A. Beimel, Y. Ishai, and T. Malkin, "Reducing the servers computation in private information retrieval: PIR with preprocessing," in *CRYPTO*, 2000.

[15] A. Beimel and Y. Stahl, "Robust information-theoretic private information retrieval," in *SCN*, 2002.

[16] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya, "Incentivizing outsourced computation," in *NetEcon*, 2008.

[17] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable delegation of computation over large datasets," in *CRYPTO*, 2011.

[18] D. J. Bernstein, "Curve25519: New Diffie-Hellman speed records," in *PKC*, 2006.

[19] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *ITCS*, 2012.

[20] A.J. Blumberg, J. Thaler, V. Vu, and M. Walfish, "Verifiable computation using multiple provers," *IACR Cryptol. ePrint Arch.*, 2014.

[21] D. Boneh, D. M. Freeman, J. Katz, and B. Waters, "Signing a linear subspace: signature schemes for network coding," in *PKC*, 2009.

[22] D. Boneh and D. M. Freeman, "Homomorphic signatures for polynomial functions," in *EUROCRYPT*, 2011.

[23] E. Boyle, N. Gilboa, and Y. Ishai, "Breaking the circuit size barrier for secure computation under DDH," in *CRYPTO*, 2016.

[24] G. Brassard, D. Chaum, and C. Crépeau, "Minimum disclosure proofs of knowledge," *J. Comput. Syst. Sci.*, vol. 37, no. 2, 1988.

[25] B. Braun, A. J. Feldman, Z. Ren, S. T. V. Setty, A. J. Blumberg, and M. Walfish, "Verifying computations with state," in *SOSP*, 2013.

[26] Anirudh C, A. Choudhury, and A. Patra, "A survey on perfectly-secure verifiable secret-sharing," *IACR Cryptol. ePrint Arch.*, 2021.

[27] R. Canetti, B. Riva, and G. N. Rothblum, "Practical delegation of computation using multiple servers," in *CCS*, 2011.

[28] R. Canetti, B. Riva, and G. N. Rothblum, "Two protocols for delegation of computation," in *ICITS*, 2012.

[29] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, 2002.

[30] D. Catalano and D. Fiore, "Practical homomorphic MACs for arithmetic circuits," in *EUROCRYPT*, 2013.

[31] D. Catalano, D. Fiore, R. Gennaro, and L. Nizzardo, "Generalizing homomorphic MACs for arithmetic circuits," in *PKC*, 2014.

[32] D. Catalano, D. Fiore, and B. Warinschi, "Efficient network coding signatures in the standard model," in *PKC*, 2012.

[33] D. Catalano, D. Fiore, and B. Warinschi, "Homomorphic signatures with efficient verification for polynomial functions," in *CRYPTO*, 2014.

[34] S.C. Chapra and R.P. Canale, *Numerical methods for engineers (8th edition)*. Mc Graw Hill, 2020.

[35] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *CRYPTO*, 1992.

[36] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. P. Ward, "Marlin: preprocessing zkSNARKs with universal and updatable SRS," in *EUROCRYPT*, 2020.

[37] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *FOCS*, 1995.

[38] K.M. Chung, Y.T. Kalai, and S.P. Vadhan, "Improved delegation of computation using fully homomorphic encryption," in *CRYPTO*, 2010.

[39] G. Cormode, M. Mitzenmacher, and J. Thaler, "Practical verified computation with streaming interactive proofs," in *ITCS*, 2012.

[40] H. Corrigan-Gibbs, D. Boneh, and D. Maziéres, "Riposte: an anonymous messaging system handling millions of users," in *IEEE Symposium on Security and Privacy*, 2015.

[41] C. Cremers, M. Naor, S. Paz, and E. Ronen, "CHIP and CRISP: protecting all parties against compromise through identity-binding PAKEs," *IACR Cryptol. ePrint Arch.*, 2020.

[42] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung, "Secure efficient multiparty computing of multivariate polynomials and applications," in *ACNS*, 2011.

[43] H. de Valence, "The ristretto group," *https://ristretto.group*, 2021.

[44] C. Devet, I. Goldberg, and N. Heninger, "Optimally robust private information retrieval," in *USENIX Security Symposium*, 2012.

[45] G. Di Crescenzo and H. Lipmaa, "Succinct NP proofs from an extractability assumption," in *CiE*, 2008.

[46] K. Elkhiyaoui, M. Önen, M. Azraoui, and R. Molva, "Efficient techniques for publicly verifiable delegation of computation," in *ASIACCS*, 2016.

[47] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *FOCS*, 1987.

[48] D. Fiore and R. Gennaro, "Publicly verifiable delegation of large polynomials and matrix computations, with applications," in *CCS*, 2012.

[49] D. Fiore, R. Gennaro, and V. Pastro, "Efficiently verifiable computation on encrypted data," in *CCS*, 2014.

[50] L. Fortnow and C. Lund, "Interactive proof systems and alternating time-space complexity," *Theor. Comput. Sci.*, vol. 113, no. 1, 1993.

[51] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: outsourcing computation to untrusted workers," in *CRYPTO*, 2010.

[52] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct NIZKs without PCPs," in *EUROCRYPT*, 2013.

[53] R. Gennaro and D. Wichs, "Fully homomorphic message authenticators," in *ASIACRYPT*, 2013.

[54] C. Gentry and D. Wichs, "Separating succinct non-interactive arguments from all falsifiable assumptions," in *STOC*, 2011.

[55] I. Goldberg, "Improving the robustness of private information retrieval," in *IEEE Symposium on Security and Privacy*, 2007.

[56] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *STOC*, 1985.

[57] S. Goldwasser, Y.T. Kalai, and G.N. Rothblum, "Delegating computation: interactive proofs for muggles," in *STOC*, 2008.

[58] S. Gopalakrishnan and N. Bourbakis, "Curve fitting methods: a survey," *Int. J. Monit. Surveillance Technol. Res.*, vol. 4, no. 4, 2016.

[59] S. Gorbunov, V. Vaikuntanathan, and D. Wichs, "Leveled fully homomorphic signatures from standard lattices," in *STOC*, 2015.

[60] J. Groth, "Short pairing-based non-interactive zero-knowledge arguments," in *ASIACRYPT*, 2010.

[61] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT*, 2016.

[62] T. Gupta, N. Crooks, W. Mulhern, S.T. V. Setty, L. Alvisi, and M. Walfish, "Scalable and private media consumption with Popcorn," in *NSDI*, 2016.

[63] A. Haeberlen, P. Kouznetsov, and P. Druschel, "PeerReview: practical accountability for distributed systems," in *SOSP*, 2007.

[64] M. Hamburg, "Decaf: Eliminating cofactors through point compression," in *CRYPTO*, 2015.

[65] W. Heinrichs, E. Quirin, U. Fauth, I. Tzanova, and M. Halmágyi, "Investigation of inspiratory pressure-volume curves on mechanically ventilated patients using least square polynomial fit," in *the IFAC Symposium on Modelling and Control in Biomedical Systems*, 1988.

[66] Y. Huang and I. Goldberg, "Outsourced private information retrieval," in *WPES*, 2013.
[67] Y. Ishai, E. Kushilevitz, and R. Ostrovsky, "Efficient arguments without short PCPs," in *IEEE Conference on Computational Complexity*, 2007.
[68] G. Karame, M. Strasser, and S. Capkun, "Secure remote execution of sequential computations," in *ICICS*, 2009.
[69] J. Kilian, "A note on efficient zero-knowledge proofs and arguments," in *STOC*, 1992.
[70] J. Kilian, "Improved efficient arguments," in *CRYPTO*, 1995.
[71] G. Kol and R. Raz, "Competing-provers protocols for circuit evaluation," *Theory Comput.*, vol. 10, no. 5, 2014.
[72] B. Kreuter, A. Shelat, and C.H. Shen, "Billion-gate secure computation with malicious adversaries," in *USENIX Security Symposium*, 2012.
[73] K. Kurosawa, "How to correct errors in multi-server PIR," in *ASIACRYPT*, 2019.
[74] R.W.F. Lai, G. Malavolta, and D. Schröder, "Homomorphic secret sharing for low degree polynomials," in *ASIACRYPT*, 2018.
[75] C. Lund, L. Fortnow, H.J. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," in *FOCS*, 1990.
[76] S. Micali, "CS proofs," in *FOCS*, 1994.
[77] S. Michielse, N. Coupland, R. Camicioli, R. Carter, P. Seres, J. Sabino, and N. Malykhin, "Selective effects of aging on brain white matter microstructure: A diffusion tensor imaging tractography study," *NeuroImage*, vol. 52, no. 4, 2010.
[78] F. Monrose, P. Wyckoff, and A. D. Rubin, "Distributed execution with remote audit," in *NDSS*, 1999.
[79] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: nearly practical verifiable computation," in *IEEE Symposium on Security and Privacy*, 2013.
[80] B. Parno, M. Raykova, and V. Vaikuntanathan, "How to delegate and verify in public: verifiable computation from attribute-based encryption," in *TCC*, 2012.
[81] B. Schoenmakers, M. Veeningen, and N. de Vreede, "Trinocchio: privacy-preserving outsourcing by distributed verifiable computation," in *ACNS*, 2016.
[82] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P.K. Khosla, "Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems," in *SOSP*, 2005.
[83] S. Setty, "Spartan: efficient and general-purpose zkSNARKs without trusted setup," in *CRYPTO*, 2020.
[84] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish, "Resolving the conflict between generality and plausibility in verified computation, in *EuroSys*, 2013.
[85] S. Setty, R. McPherson, A. J. Blumberg, and M. Walfish, "Making argument systems for outsourced computation practical (sometimes)," in *NDSS*, 2012.
[86] S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish, "Taking proof-based verified computation a few steps closer to practicality," in *USENIX Security Symposium*, 2012.
[87] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, 1979.
[88] A. Shamir, "IP=PSPACE," in *FOCS*, 1990.
[89] Z. Shan, K. Ren, M. Blanton, and C. Wang, "Practical secure computation outsourcing: a survey," *ACM Comput. Surv.* vol. 51, no. 2, 2018.
[90] S. W. Smith and S. H. Weingart, "Building a high-performance, programmable secure coprocessor," *Comput. Networks*, vol. 31, no. 8, 1999.
[91] J. Thaler, "Time-optimal interactive proofs for circuit evaluation," in *CRYPTO*, 2013.
[92] J. Thaler: Proofs, arguments, and zero-knowledge. https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html
[93] J. Thaler, M. Roberts, M. Mitzenmacher, and H. Pfister, "Verifiable computation with massively parallel interactive proofs," in *HotCloud*, 2012.
[94] D.P. Woodruff and S. Yekhanin, "A geometric approach to information-theoretic private information retrieval," in *IEEE Conference on Computational Complexity*, 2005.
[95] B. S. Yee, *Using secure coprocessors*. PhD thesis, Carnegie Mellon University, 1994.
[96] M. Yoshida and S. Obana, "Verifiably multiplicative secret sharing," *IEEE Trans. Inf. Theory*, vol. 65, no. 5, 2019.
[97] L.F. Zhang and R. Safavi-Naini, "Verifiable multi-server private information retrieval," in *ACNS*, 2014.
[98] L. Zhao, X. Wang, and X. Huang, "Verifiable single-server private information retrieval from LWE with binary errors," *Inf. Sci.*, vol. 546, 2021.

APPENDIX A
PROOF FOR THEOREM 1

Proof for input privacy. According to Definition 4, we need to show that for any set $T \subseteq [k]$ of cardinality $\leq t$, any $F \in \mathcal{P}(q,m,d)$, and any $\mathbf{x}^0, \mathbf{x}^1 \in \mathbb{F}_q^m$, $\sigma_{\Pi_1}(T,F,\mathbf{x}^0)$ and $\sigma_{\Pi_1}(T,F,\mathbf{x}^1)$ are identically distributed. It suffices to take $T=[t]$ and show that for any $\mathbf{x} \in \mathbb{F}_q^m$, $\sigma_{\Pi_1}(T,F,\mathbf{x})$ is uniformly distributed over $\mathbb{F}_q^{mt}$. The specifications of Π_1 show that $\mathsf{ProbGen}(\mathsf{pk}_F, \mathbf{x})$ will output $\sigma_i = c(i)$ for every $i \in [k]$. In particular, $\sigma_T = (\sigma_i)_{i\in T}$ will satisfy the equation system

$$\underbrace{\begin{pmatrix} 1-\frac{1}{\alpha^t} & 1-\frac{1}{\alpha^{t-1}} & \cdots & 1-\frac{1}{\alpha} \\ 2-\frac{2^{t+1}}{\alpha^t} & 2^2-\frac{2^{t+1}}{\alpha^{t-1}} & \cdots & 2^t-\frac{2^{t+1}}{\alpha} \\ \vdots & \vdots & \cdots & \vdots \\ t-\frac{t^{t+1}}{\alpha^t} & t^2-\frac{t^{t+1}}{\alpha^{t-1}} & \cdots & t^t-\frac{t^{t+1}}{\alpha} \end{pmatrix}}_{G} \begin{pmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_t \end{pmatrix} = \begin{pmatrix} \sigma_1 - \mathbf{x} - \frac{1}{\alpha^{t+1}}(\mathbf{a}-\mathbf{x}) \\ \sigma_2 - \mathbf{x} - \frac{2^{t+1}}{\alpha^{t+1}}(\mathbf{a}-\mathbf{x}) \\ \vdots \\ \sigma_t - \mathbf{x} - \frac{t^{t+1}}{\alpha^{t+1}}(\mathbf{a}-\mathbf{x}) \end{pmatrix}. \tag{20}$$

Note that G is non-singular as $\alpha \notin [t]$. For any choice of σ_T, there is a unique set of random vectors $\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_t$ such that (20) is satisfied. Hence, σ_T is uniformly distributed over $\mathbb{F}_q^{mt}$.

Proof for security. By Definition 2, it suffices to show that for any set $T \subseteq [k]$ of cardinality t, any $F \in \mathcal{P}(q,m,d)$, any adversary $\mathcal{A}$, $\Pr[\mathbf{Exp}^{\mathrm{PriV}}_{\mathcal{A},\Pi_1}(T,F,\lambda) = 1] \leq \epsilon$. W.l.o.g., we take $T=[t]$ and consider the Experiment 1:

- *The challenger mimics* $\mathsf{KeyGen}(\lambda, F)$ *as follows: choose* $\boldsymbol{\ell}_0, \boldsymbol{\ell}_1 \leftarrow \mathbb{F}_q^m$, *let* $\ell(u) = \boldsymbol{\ell}_0 + \boldsymbol{\ell}_1 u$ *and* $\rho_i = F$ *for every* $i \in [k]$, *compute* $f(u) = F(\ell(u))$, *set* $\mathsf{pk}_F = \ell(u)$ *and* $\mathsf{vk}_F = (\ell(u), f(u))$. *It then invokes the adversary* $\mathcal{A}$ *with* $(F, \mathsf{pk}_F, \mathsf{vk}_F, \rho_T)$.
- *Given* $(F, \mathsf{pk}_F, \mathsf{vk}_F, \rho_T)$, $\mathcal{A}$ *chooses an input* $\mathbf{x} \in \mathbb{F}_q^m$ *and gives it to the challenger.*
- *The challenger mimics* $\mathsf{ProbGen}(\mathsf{pk}_F, \mathbf{x})$ *as follows: choose* $a \leftarrow \mathbb{F}_q^*, \alpha \leftarrow \mathbb{F}_q^* \setminus [k], \mathbf{r}_1, \ldots, \mathbf{r}_t \leftarrow \mathbb{F}_q^m$, *compute* $\mathbf{a} = \ell(a)$, $\mathbf{r}_{t+1} = \alpha^{-t-1}(\mathbf{a} - (\mathbf{x} + \sum_{s=1}^{t} \mathbf{r}_s \alpha^s))$, *define* $c(u) = \mathbf{x} + \sum_{s=1}^{t+1} \mathbf{r}_s u^s$, *set* $\mathsf{vk}_{\mathbf{x}} = (a, \alpha)$ *and* $\sigma_i = c(i)$ *for all* $i \in [k]$. *It then gives* σ_T *to* $\mathcal{A}$.
- $\mathcal{A}$ *chooses* t *partial results* $\hat{\pi}_T = (\hat{\pi}_1, \ldots, \hat{\pi}_t) \in \mathbb{F}_q^t$ *and gives them to the challenger.*
- *For every* $i \in [k] \setminus T$, *the challenger computes* $\hat{\pi}_i \leftarrow \mathsf{Compute}(i, \rho_i, \sigma_i)$.
- *The challenger mimics* $\mathsf{Verify}(\mathsf{vk}_F, \mathsf{vk}_{\mathbf{x}}, \{\hat{\pi}_i\}_{i=1}^k)$ *as follows: interpolate a polynomial* $\hat{\phi}(u)$ *of degree* $< k$ *such that* $\hat{\phi}(i) = \hat{\pi}_i$ *for all* $i \in [k]$. *If* $\hat{\phi}(\alpha) = f(a)$, *set* $\hat{y} = \hat{\phi}(0)$; *otherwise, set* $\hat{y} = \perp$.
- *If* $\hat{y} \notin \{\perp, F(\mathbf{x})\}$, *then outputs 1; otherwise, outputs 0.*

For every $i \in [k]$, let π_i be the output of correctly executing $\mathsf{Compute}(i, \rho_i, \sigma_i)$. Then $\pi_i = \hat{\pi}_i$ for every $i \in [k] \setminus T$; and

$\phi(u) = F(c(u))$ is the polynomial of degree $< k$ such that $\phi(i) = \pi_i$ for all $i \in [k]$. Note that $\hat{y} \neq \perp$ if and only if $\hat{\phi}(\alpha) = f(a)$. As $f(a) = \phi(\alpha)$, $\hat{y} \neq \perp$ if and only if

$$\hat{\phi}(\alpha) = \phi(\alpha). \tag{21}$$

On the other hand, $\phi(0) = F(\mathbf{x})$. When (21) is true, we have that $\hat{y} = \hat{\phi}(0)$ and thus $\hat{y} \neq F(\mathbf{x})$ if and only if

$$\hat{\phi}(0) \neq \phi(0). \tag{22}$$

Equation (22) requires that $\Delta(u) = \hat{\phi}(u) - \phi(u)$ is a nonzero polynomial and (21) requires that the random field element α is a root of the degree $< k$ polynomial $\Delta(u)$. As $\Delta(u)$ has at most $k-1$ roots in $\mathbb{F}_q^* \setminus [k]$ and $\alpha \in \mathbb{F}_q^* \setminus [k]$ is randomly chosen and completely hidden from $\mathcal{A}$ (which can be seen from (20)), we have that

$$\begin{aligned}
&\Pr[\mathbf{Exp}^{\mathrm{PriV}}_{\mathcal{A},\Pi_1}(T,F,\lambda)=1]\\
=\ &\Pr[(\hat{y}\neq\perp)\wedge(\hat{y}\neq F(\mathbf{x}))]\\
=\ &\Pr[(\hat{\phi}(\alpha)=\phi(\alpha))\wedge(\hat{\phi}(0)\neq\phi(0))]\\
\leq\ &\Pr[(\hat{\phi}(\alpha)=\phi(\alpha))\mid(\hat{\phi}(0)\neq\phi(0))]\\
\leq\ &(k-1)/(q-1-k).
\end{aligned}$$

Hence, Π_1 is (t,ϵ)-secure with $\epsilon = \frac{d(t+1)}{q-2-d(t+1)}$.

Appendix B
Proof for Theorem 2

Proof for input privacy. By Definition 4, we need to show that for any set $T \subseteq [k]$ of cardinality t, any $F \in \mathcal{P}(q,m,d)$, and any $\mathbf{x}^0, \mathbf{x}^1 \in \mathbb{F}_q^m$, $\sigma_{\Pi_2}(T,F,\mathbf{x}^0)$ and $\sigma_{\Pi_2}(T,F,\mathbf{x}^1)$ are identically distributed. It suffices to set $T = [t]$ and show for any $\mathbf{x} \in \mathbb{F}_q^m$, $\sigma_{\Pi_2}(T,F,\mathbf{x})$ is uniformly distributed over the set $\mathbb{F}_q^{(m+1)t}$. However, this is obvious because the input shares are all computed with Shamir's threshold scheme [87].

Proof for security. By Definition 2, it suffices to show that for any $T \subseteq [k]$ of cardinality t, any $F \in \mathcal{P}(q,m,d)$, any adversary $\mathcal{A}$, $\Pr[\mathbf{Exp}^{\mathrm{PriV}}_{\mathcal{A},\Pi_2}(T,F,\lambda)=1] \leq \epsilon$. W.l.o.g., we take $T = [t]$ and consider Experiment 1:

- *The challenger mimics* $\mathsf{KeyGen}(\lambda, F)$ *as follows: let* $\rho_i = F$ *for all* $i \in [k]$*, and set* $\mathsf{pk}_F = \perp$ *and* $\mathsf{vk}_F = \perp$*. It then invokes* $\mathcal{A}$ *with* $(F, \mathsf{pk}_F, \mathsf{vk}_F, \rho_T)$.
- *Given* $(F, \mathsf{pk}_F, \mathsf{vk}_F, \rho_T)$*,* $\mathcal{A}$ *chooses an input* $\mathbf{x} \in \mathbb{F}_q^m$ *and gives it to the challenger.*
- *The challenger mimics* $\mathsf{ProbGen}(\mathsf{pk}_F, \mathbf{x})$ *as follows: choose* $\alpha \leftarrow \mathbb{F}_q^*, \mathbf{r}_1, \ldots, \mathbf{r}_t \leftarrow \mathbb{F}_q^m, \gamma_1, \ldots, \gamma_t \leftarrow \mathbb{F}_q$*, let* $c(u) = \mathbf{x} + \sum_{s=1}^t \mathbf{r}_s u^s$ *and* $b(u) = \alpha + \sum_{s=1}^t \gamma_s u^s$*, compute* $\sigma_i = (c(i), b(i))$ *for all* $i \in [k]$*, and set* $\mathsf{vk}_{\mathbf{x}} = \alpha$*. It then gives* σ_T *to* $\mathcal{A}$*.*
- $\mathcal{A}$ *chooses* t *partial results* $\{(\hat{v}_i, \hat{w}_i)\}_{i \in T}$ *and gives them to the challenger.*
- *For every* $i \in [k] \setminus T$*, the challenger computes* $(\hat{v}_i, \hat{w}_i) \leftarrow \mathsf{Compute}(i, \rho_i, \sigma_i)$.
- *The challenger interpolates a polynomial* $\hat{\phi}(u)$ *of degree* $\leq dt$ *such that* $\hat{\phi}(i) = \hat{v}_i$ *for all* $i \in [k]$*; and a degree* $\leq (d+1)t$ *polynomial* $\hat{\psi}(u)$ *such that* $\hat{\psi}(i) = \hat{w}_i$ *for all* $i \in [k]$*. If* $\hat{\psi}(0) = \alpha\hat{\phi}(0)$*, then it sets* $\hat{y} = \hat{\phi}(0)$*; otherwise, it sets* $\hat{y} = \perp$.
- *If* $\hat{y} \notin \{\perp, F(\mathbf{x})\}$*, then outputs 1; otherwise, outputs 0.*

It's easy to see that $\hat{y} \neq \perp$ if and only if $\hat{\psi}(0) = \alpha\hat{\phi}(0)$. For $\phi(u) = F(c(u))$ and $\psi(u) = \phi(u)b(u)$, we always have that $\psi(0) = \alpha\phi(0)$. Thus, the event $\hat{y} \neq \perp$ occurs if and only if

$$\hat{\psi}(0) - \psi(0) = \alpha(\hat{\phi}(0) - \phi(0)). \tag{23}$$

On the other hand, $\phi(0) = F(\mathbf{x})$. When (23) is true, we will have that $\hat{y} = \hat{\phi}(0)$ and thus $\hat{y} \neq F(\mathbf{x})$ if and only if $\hat{\phi}(0) \neq \phi(0)$. Note that $\alpha \in \mathbb{F}_q^*$ is randomly chosen and completely hidden from $\mathcal{A}$ (i.e., $\mathcal{S}_T$) due to the security of Shamir's threshold scheme. Thus, we have that

$$\begin{aligned}
&\Pr[\mathbf{Exp}^{\mathrm{PriV}}_{\mathcal{A},\Pi_2}(T,F,\lambda)=1]\\
=\ &\Pr\left[(\hat{y}\neq\perp)\wedge(\hat{y}\neq F(\mathbf{x}))\right]\\
=\ &\Pr\left[\alpha=\frac{\hat{\psi}(0)-\psi(0)}{\hat{\phi}(0)-\phi(0)}\wedge(\hat{\phi}(0)\neq\phi(0))\right]\\
\leq\ &\Pr\left[\alpha=\frac{\hat{\psi}(0)-\psi(0)}{\hat{\phi}(0)-\phi(0)}\;\middle|\;(\hat{\phi}(0)\neq\phi(0))\right]\\
\leq\ &1/(q-1).
\end{aligned}$$

Hence, Π_2 is (t,ϵ)-secure with $\epsilon = 1/(q-1)$.

Appendix C
Proof for Theorem 3

Proof for input privacy. By Definition 4, we need to show that for any set $T \subseteq [k]$ of cardinality t, any $F \in \mathcal{P}(q,m,d)$, and any $\mathbf{x}^0, \mathbf{x}^1 \in \mathbb{F}_q^m$, $\sigma_{\Pi_3}(T,F,\mathbf{x}^0)$ and $\sigma_{\Pi_3}(T,F,\mathbf{x}^1)$ are identically distributed. It suffices to take $T = [t]$ and show that for any $\mathbf{x} \in \mathbb{F}_q^m$, $\sigma_{\Pi_3}(T,F,\mathbf{x})$ is uniformly distributed over $\mathbb{F}_{q^2}^{mt}$. In an execution of Π_3, $\mathsf{ProbGen}(\mathsf{pk}_F, \mathbf{x})$ will output an input share $\sigma_i = c(i)$ for all $i \in [k]$. In particular, $\sigma_T = (\sigma_i)_{i \in T}$ will satisfy the following equation system

$$\underbrace{\begin{pmatrix} 1-\alpha & 1-\alpha^2 & \cdots & 1-\alpha^t \\ 2-\alpha & 2^2-\alpha^2 & \cdots & 2^t-\alpha^t \\ \vdots & \vdots & \cdots & \vdots \\ t-\alpha & t^2-\alpha^2 & \cdots & t^t-\alpha^t \end{pmatrix}}_{G} \begin{pmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_t \end{pmatrix} = \begin{pmatrix} \sigma_1 - \mathbf{x} \\ \sigma_2 - \mathbf{x} \\ \vdots \\ \sigma_t - \mathbf{x} \end{pmatrix}.$$

Note that the coefficient matrix G is non-singular because $\alpha \notin [t]$. For any value of $\sigma_T \in \mathbb{F}_{q^2}^{mt}$, there is a unique set of vectors $\mathbf{r}_1, \ldots, \mathbf{r}_t$ such that the above equation system is satisfied. Hence, σ_T is uniformly distributed over $\mathbb{F}_{q^2}^{mt}$.

Proof for security. By Definition 2, it suffices to show that for any $T \subseteq [k]$ of cardinality t, any $F \in \mathcal{P}(q,m,d)$, any adversary $\mathcal{A}$, $\Pr[\mathbf{Exp}^{\mathrm{PriV}}_{\mathcal{A},\Pi_3}(T,F,\lambda)=1] \leq \epsilon$. W.l.o.g., we take $T = [t]$ and consider Experiment 1:

- *The challenger mimics* $\mathsf{KeyGen}(\lambda, F)$ *as follows: let* $\rho_i = F$ *for all* $i \in [k]$*,* $\mathsf{pk}_F = \perp$*, and* $\mathsf{vk}_F = \perp$*. It then invokes* $\mathcal{A}$ *with* $(F, \mathsf{pk}_F, \mathsf{vk}_F, \rho_T)$.
- *Given* $(F, \mathsf{pk}_F, \mathsf{vk}_F, \rho_T)$*,* $\mathcal{A}$ *chooses an input* $\mathbf{x} \in \mathbb{F}_q^m$ *and gives it to the challenger.*

- *The challenger mimics* $\mathsf{ProbGen}(\mathsf{pk}_F, \mathbf{x})$ *as follows: choose* $\alpha \leftarrow \mathbb{F}_{q^2}^* \setminus [k]$, $\mathbf{r}_1, \ldots, \mathbf{r}_t \leftarrow \mathbb{F}_{q^2}^m$, *let* $c(u) = \mathbf{x} + \sum_{s=1}^{t} \mathbf{r}_s(u^s - \alpha^s)$, *set* $\mathsf{vk}_{\mathbf{x}} = \alpha$ *and* $\sigma_i = c(i)$ *for all* $i \in [k]$. *It then gives* σ_T *to* $\mathcal{A}$.
- $\mathcal{A}$ *chooses* t *partial results* $\{\hat{\pi}_i\}_{i \in T}$ *and gives them to the challenger.*
- *For every* $i \in [k] \setminus T$, *the challenger computes* $\hat{\pi}_i \leftarrow \mathsf{Compute}(i, \rho_i, \sigma_i)$.
- *The challenger interpolates a polynomial* $\hat{\phi}(u)$ *of degree* $\leq dt$ *such that* $\hat{\phi}(i) = \hat{\pi}_i$ *for all* $i \in [k]$. *If* $\hat{\phi}(\alpha) \in \mathbb{F}_q$, *it sets* $\hat{y} = \hat{\phi}(\alpha)$; *otherwise, it sets* $\hat{y} = \perp$.
- *If* $\hat{y} \notin \{\perp, F(\mathbf{x})\}$, *then outputs 1; otherwise, outputs 0.*

The event $\mathbf{Exp}^{\mathrm{PriV}}_{\mathcal{A},\Pi_3}(T, F, \lambda) = 1$ occurs if and only if $\hat{\phi}(\alpha) \in \mathbb{F}_q \setminus \{F(\mathbf{x})\}$, i.e., if and only if α is a root of at least one of the $q-1$ polynomials in $\{\hat{\phi}(u) - \delta : \delta \in \mathbb{F}_q \setminus \{F(\mathbf{x})\}\}$. Note that $\mathcal{A}$ knows nothing about $\alpha \in \mathbb{F}_{q^2}^* \setminus [k]$, which was randomly chosen. We must have that $\Pr[\hat{\phi}(\alpha) \in \mathbb{F}_q \setminus \{F(\mathbf{x})\}] \leq (q-1)dt/(q^2-1-k)$. Hence, Π_3 is (t, ϵ)-secure for $\epsilon = (q-1)dt/(q^2-2-dt)$.

Appendix D
Cryptographic Assumptions

Definition 5. (DLog Assumption) *Let* $\mathbb{G}$ *be a cyclic group of order* $q > 2^\lambda$. *For a generator* $g \in \mathbb{G}$ *and* $\alpha \leftarrow \mathbb{Z}_q$ *we define the advantage of an adversary* $\mathcal{A}$ *in solving the* DLog problem *as* $\mathbf{Adv}^{\mathrm{DLog}}_{\mathcal{A}}(\lambda) = \Pr[\mathcal{A}(g, g^\alpha) = \alpha]$. *We say that the* DLog assumption holds in $\mathbb{G}$ *if for any PPT adversary* $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{DLog}}_{\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$.

Definition 6. (D-DHI Assumption [30]) *Let* $\mathbb{G}$ *be a cyclic group of order* $q > 2^\lambda$. *For a generator* $g \in \mathbb{G}$ *and* $\alpha \leftarrow \mathbb{Z}_q$ *we define the advantage of an adversary* $\mathcal{A}$ *in solving the* D-DHI problem *as* $\mathbf{Adv}^{\mathrm{DHI}}_{\mathcal{A}}(\lambda) = \Pr[\mathcal{A}(g, g^\alpha, \ldots, g^{\alpha^D}) = g^{1/\alpha}]$ *and we say that the* D-DHI assumption holds in $\mathbb{G}$ *if for any PPT adversary* $\mathcal{A}$ *and for* $D = \mathrm{poly}(\lambda)$, $\mathbf{Adv}^{\mathrm{DHI}}_{\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$.

Appendix E
Proof for Theorem 4

By Definition 3, it suffices to show that for any $T \subseteq [k]$ of cardinality $\leq t$, any $F \in \mathcal{P}(q, m, d)$, any PPT adversary $\mathcal{A}$, $\epsilon := \Pr[\mathbf{Exp}^{\mathrm{PubV}}_{\mathcal{A},\Pi_4}(T, F, \lambda) = 1]$ must be negligible in λ. W.l.o.g., we take $T = [t]$ and construct a PPT algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to solve the $(k-1)$-DHI problem in $\mathbb{G}$:

- *Given* $I = (g, g^\alpha, \ldots, g^{\alpha^{k-1}})$, $\mathcal{B}$ *mimics* $\mathsf{KeyGen}(\lambda, F)$ *as follows: choose* $\boldsymbol{\ell}_0, \boldsymbol{\ell}_1 \leftarrow \mathbb{F}_q^m$, *let* $\ell(u) = \boldsymbol{\ell}_0 + \boldsymbol{\ell}_1 u$, *compute* $f(u) = F(\ell(u))$, *set* $\rho_i = F$ *for every* $i \in [k]$, $\mathsf{pk}_F = \ell(u)$, *and* $\mathsf{vk}_F = (\ell(u), f(u))$. *It then invokes* $\mathcal{A}$ *with* $(F, \mathsf{pk}_F, \mathsf{vk}_F, \rho_T)$.
- $\mathcal{A}$ *chooses an input* $\mathbf{x} \in \mathbb{F}_q^m$ *and gives it to* $\mathcal{B}$.
- $\mathcal{B}$ *mimics* $\mathsf{ProbGen}(\mathsf{pk}_F, \mathbf{x})$ *as follows: choose* $a \leftarrow \mathbb{F}_q^*$, $\sigma_1, \sigma_2, \ldots, \sigma_t \leftarrow \mathbb{F}_q^m$, *set* $\mathsf{vk}_{\mathbf{x}} = (g, g^a, \ldots, g^{a^d}, g^\alpha, \ldots, g^{\alpha^{k-1}})$. *It then gives both* $\mathsf{vk}_{\mathbf{x}}$ *and and* σ_T *to* $\mathcal{A}$.
- $\mathcal{A}$ *chooses* $\hat{\pi}_T = \{\hat{\pi}_i\}_{i \in T} \in \mathbb{F}_q^t$ *and gives it to* $\mathcal{B}$.
- $\mathcal{B}$ *computes* $\pi_i = F(\sigma_i)$ *for all* $i \in [t]$ *and interpolates a polynomial* $\delta(u) = \sum_{i=0}^{k-1} \delta_i u^i$ *of degree* $< k$ *such that* $\delta(i) = \hat{\pi}_i - \pi_i$ *for all* $i \in [t]$ *and* $\delta(i) = 0$ *for all* $t < i \leq k$. *Note that* $\mathcal{B}$ *is able to compute* $g^{\delta(\alpha)}$ *with* I *and the coefficients of* $\delta(u)$. *If* $\delta(0) \neq 0$ *and* $g^{\delta(\alpha)} = 1$, $\mathcal{B}$ *outputs*

$$g^{1/\alpha} = \left(\prod_{i=1}^{k-1} g^{\delta_i \alpha^{i-1}} \right)^{-\frac{1}{\delta_0}}, \tag{24}$$

Note that what $\mathcal{A}$ sees in this procedure is identically distributed to what it should see in $\mathbf{Exp}^{\mathrm{PubV}}_{\mathcal{A},\Pi_4}(T, F, \lambda)$. In particular, the vectors $\mathbf{r}_1, \ldots, \mathbf{r}_t \in \mathbb{F}_q^m$ used to encode $\mathbf{x}$ are implicit and uniquely determined with (20). Furthermore, the $\mathbf{r}_{t+1}$ could be computed as $\mathbf{r}_{t+1} = \alpha^{-t-1}(\mathbf{a} - (\mathbf{x} + \sum_{s=1}^{t} \mathbf{r}_s \alpha^s))$. Although $\mathcal{B}$ is unable to compute the partial result π_i for every $t < i \leq k$, we have that $\delta(u) = \hat{\phi}(u) - \phi(u)$, where $\hat{\phi}(u)$ is interpolated from $(\hat{\pi}_1, \ldots, \hat{\pi}_t, \pi_{t+1}, \ldots, \pi_k)$ such that $\hat{\phi}(i) = \hat{\pi}_i$ for all $i \in [t]$ and $\hat{\phi}(i) = \pi_i$ for all $t < i \leq k$, and $\phi(u)$ is interpolated from $(\pi_1, \ldots, \pi_t, \pi_{t+1}, \ldots, \pi_k)$ such that $\phi(i) = \pi_i$ for all $i \in [k]$. It is clear that $\mathcal{A}$ wins in the security experiment if and only if $\delta(0) \neq 0$ and $g^{\delta(\alpha)} = 1$, where "$g^{\delta(\alpha)} = 1$" signs that the partial results chosen by $\mathcal{A}$ will be accepted and "$\delta(0) \neq 0$" signs that Verify will output a value $\neq F(\mathbf{x})$. When $\mathcal{A}$ wins, (24) will allow $\mathcal{B}$ to learn $g^{1/\alpha}$. Hence, we have that $\Pr[\mathcal{B}(I) = g^{1/\alpha}] = \Pr[\mathbf{Exp}^{\mathrm{PubV}}_{\mathcal{A},\Pi_4}(T, F, \lambda) = 1] = \epsilon$. Under the $(k-1)$-DHI assumption in $\mathbb{G}$, ϵ must be negligible in λ. Hence, Π_4 should be t-secure under the same assumption.

Appendix F
Proof for Theorem 5

By Definition 3, it suffices to show that for any $T \subseteq [k]$ of cardinality $\leq t$, any $F \in \mathcal{P}(q, m, d)$, any PPT adversary $\mathcal{A}$, $\epsilon := \Pr[\mathbf{Exp}^{\mathrm{PubV}}_{\mathcal{A},\Pi_5}(T, F, \lambda) = 1]$ is negligible in λ. Without loss of generality, we take $T = [t]$ and construct a PPT algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to solve the DLog problem in $\mathbb{G}$:

- *Given* $I = (g, g^\alpha)$, $\mathcal{B}$ *mimics* $\mathsf{KeyGen}(\lambda, F)$ *as follows: let* $\rho_i = F$ *for all* $i \in [k]$, $\mathsf{pk}_F = \perp$ *and* $\mathsf{vk}_F = \perp$. *It then invokes* $\mathcal{A}$ *with* $(F, \mathsf{pk}_F, \mathsf{vk}_F, \rho_T)$.
- $\mathcal{A}$ *chooses an input* $\mathbf{x} \in \mathbb{F}_q^m$ *and gives it to* $\mathcal{B}$.
- $\mathcal{B}$ *mimics* $\mathsf{ProbGen}(\mathsf{pk}_F, \mathbf{x})$ *as follows: choose* $\mathbf{c}_1, \ldots, \mathbf{c}_t \leftarrow \mathbb{F}_q^m, b_1, \ldots, b_t \leftarrow \mathbb{F}_q$, *set* $\mathsf{vk}_{\mathbf{x}} = g^\alpha$ *and* $\sigma_i = (\mathbf{c}_i, b_i)$ *for every* $i \in [t]$. *It then gives* $\mathsf{vk}_{\mathbf{x}}$ *and* σ_T *to* $\mathcal{A}$.
- $\mathcal{A}$ *chooses* $\hat{\pi}_i = (\hat{v}_i, \hat{w}_i) \in \mathbb{F}_q^2$ *for all* $i \in [t]$ *and gives* $\hat{\pi}_T$ *to* $\mathcal{B}$.
- *Let* $v_i = F(\mathbf{c}_i)$ *and* $w_i = v_i b_i$ *for* $i \in [t]$. $\mathcal{B}$ *interpolates a polynomial* $\delta(u)$ *of degree* $\leq dt$ *such that* $\delta(i) = \hat{v}_i - v_i$ *for all* $i \in [t]$ *and* $\delta(i) = 0$ *for all* $t < i \leq k$; *and interpolates a polynomial* $\Delta(u)$ *of degree* $\leq (d+1)t$ *such that* $\Delta(i) = \hat{w}_i - w_i$ *for all* $i \in [t]$ *and* $\Delta(i) = 0$ *for all* $t < i \leq k$. *If* $g^{\Delta(0)} = g^{\alpha\delta(0)}$ *and* $\delta(0) \neq 0$, *then* $\mathcal{B}$ *outputs* $\alpha = \Delta(0)/\delta(0)$; *otherwise, it outputs* $\perp$.

Note that what $\mathcal{A}$ sees in the above procedure is identically distributed to what it should see in $\mathbf{Exp}^{\mathrm{PubV}}_{\mathcal{A},\Pi_5}(T, F, \lambda)$. The vectors $\mathbf{r}_1, \ldots, \mathbf{r}_t \in \mathbb{F}_q^m$ and the numbers $\gamma_1, \ldots, \gamma_t \in \mathbb{F}_q$ for encoding $\mathbf{x}$ are implicit but uniquely determined by σ_T, g^α and $\mathbf{x}$. Although $\mathcal{B}$ is unable to compute the partial result $\pi_i =$

(v_i, w_i) for every $t < i \leq k$, it is true $\delta(u) = \hat{\phi}(u) - \phi(u)$, where $\hat{\phi}(u)$ is interpolated from $(\hat{v}_1, \ldots, \hat{v}_t, v_{t+1}, \ldots, v_k)$ such that $\hat{\phi}(i) = \hat{v}_i$ for all $i \in [t]$ and $\hat{\phi}(i) = v_i$ for all $t < i \leq k$, and $\phi(u)$ is interpolated from $(v_1, \ldots, v_k)$ such that $\phi(i) = v_i$ for all $i \in [k]$. Similarly, it is true that $\Delta(u) = \hat{\psi}(u) - \psi(u)$, where $\hat{\psi}(u)$ is interpolated from $(\hat{w}_1, \ldots, \hat{w}_t, w_{t+1}, \ldots, w_k)$ such that $\hat{\psi}(i) = \hat{w}_i$ for all $i \in [t]$ and $\hat{\psi}(i) = w_i$ for all $t < i \leq k$, and $\psi(u)$ is interpolated from $(w_1, \ldots, w_k)$ such that $\psi(i) = w_i$ for all $i \in [k]$. It is clear that $\mathcal{A}$ wins in the security experiment if and only if $\delta(0) \neq 0$ and $g^{\Delta(0)} = g^{\alpha\delta(0)}$, where "$g^{\Delta(0)} = g^{\alpha\delta(0)}$" signs that the partial results chosen by $\mathcal{A}$ will be accepted and "$\delta(0) \neq 0$" signs that Verify will output a value $\neq F(\mathbf{x})$. When $\mathcal{A}$ wins, $\mathcal{B}$ is to learn $\alpha = \Delta(0)/\delta(0)$. Hence, $\Pr[\mathcal{B}(I) = \alpha] = \Pr[\mathbf{Exp}^{\mathrm{PubV}}_{\mathcal{A},\Pi_5}(T, F, \lambda) = 1] = \epsilon$. Under the DLog assumption in $\mathbb{G}$, ϵ must be negligible in λ. Hence, Π_5 is t-secure under the same assumption.

APPENDIX G
OUTSOURCEABILITY

Definition 7. (Outsourceability) *An MSVC scheme is said to be* outsourceable *if for any x and $\{\pi_i\}_{i=1}^k$, the time $T_{\mathrm{p}} + T_{\mathrm{v}}$ required by* $\mathsf{ProbGen}(\mathsf{pk}_F, x)$ *and* $\mathsf{Verify}(\mathsf{vk}_F, \mathsf{vk}_x, \{\pi_i\}_{i=1}^k)$ *is $o(T_{\mathrm{n}})$, where T_{n} is the running time of the native computation $F(x)$.*

Theorem 8. *If $kmt^2 + k^3 + (k+d)\lambda = o(nd)$, where $n = \binom{m+d}{d}$, then the five MSVC schemes are all outsourceable.*

Proof. In Π_1, Π_2 and Π_3, ProbGen and Verify require $O(kmt^2)$ and $O(k^3)$ field operations in $\mathbb{F}_q$ (or $\mathbb{F}_{q^2}$), respectively. The native computation requires $O(nd)$ field operations. Under the given condition, we have that $T_{\mathrm{p}} + T_{\mathrm{v}} = o(T_{\mathrm{n}})$ and thus these schemes are outsourceable. In Π_4 and Π_5, ProbGen requires $O(kmt^2)$ field operations, and Verify requires both $O(k^3)$ field operations and $O(k+d)$ exponentiations in $\mathbb{G}$, where the group operations are equivalent to around $O((k+d)\lambda)$ field operations. Under the given condition, we have that $T_{\mathrm{p}} + T_{\mathrm{v}} = o(T_{\mathrm{n}})$. Thus, Π_4 and Π_5 are outsourceable. □

APPENDIX H
PIR WITH CHEATING DETECTION

Let $d = \mu(k, t)$. By using Π_4 (when $k = d(t+1)+1$) or Π_5 (when $k = (d+1)t + 1$) to publicly and verifiably compute $F(E(i))$, the client has a PIR scheme that can detect the cheating of $\leq t$ servers. Its communication complexity is $O(n^{1/d}) = O(n^{1/\mu(k,t)})$.