Publicly Verifiable Delegation of Computation

Charalampos Papamanthou[*] UC Berkeley

Elaine Shi[†] PARC/UC Berkeley

Roberto Tamassia[‡] Brown University

November 2, 2011

Abstract

We initiate the study of *publicly verifiable computation*, which generalizes authenticated data structures, and verifiable computation in the secret-key setting. In publicly verifiable computation, a trusted source outsources an application (algorithm) to an untrusted server. Any client can ask the server to run the application over some given inputs, and the server can produce a witness vouching for the correctness of the outcome. We propose publicly verifiable computation schemes supporting expressive manipulations over multivariate polynomials, such as polynomial evaluation and differentiation. Our scheme allows the client to verify the outcome in time proportional to the size of the input, and not depending on the degree and the description of the polynomial, i.e., in asymptotically less time than performing the computation locally. Moreover, our scheme allows the source to efficiently update the polynomial coefficients without performing expensive recomputations proportional to the size of the polynomial.

1 Introduction

Given the emergence of the cloud computing paradigm in business and consumer applications, it has become increasingly important to provide integrity guarantees in third-party data management settings. This security goal has prompted the development of new research areas, such as *authenticated data structures* (e.g., [21, 32, 34]) and *outsourced verifiable computation* (e.g., [1, 11, 15]), that aim at developing efficient solutions for verifying the correctness of outsourced computations, a crucial property for the trustworthiness of cloud services. In this context, it is especially important to minimize the computational overhead incurred by the verification processes at the client and server, or otherwise the benefits of outsourcing data and computations are dismissed. Ideally, the computations performed by the server should be verified by the client without having to locally rerun them and by utilizing a small additional amount of cloud storage.

Consider the following scenario. A *trusted source* has developed a cloud application and wants to make it available to its clients. However, due to the large number of clients, this application cannot be executed on the local data center of the source and is outsourced to several *untrusted servers* in the cloud. How can a client (e.g., a smart phone with limited computational resources) *efficiently* verify that the application will return the correct results, under the assumption that she trusts *only* the source that created the application? Here, efficiency means that the running time of the verification algorithm should be asymptotically less than the time needed to execute the application. Note that since we want to support *any* client, we require the verification mechanism not depend on a secret key, i.e., we are in quest of a *publicly verifiable* scheme. This requirement is used in the literature on authenticated data structures [32] but is not captured by the recent body of work on delegation of computation [4, 15].

[*]`cpap@cs.berkeley.edu`
[†]`elaines@cs.berkeley.edu`
[‡]`rt@cs.brown.edu`

This paper introduces the model of *publicly verifiable computation*. Publicly verifiable computation is strictly more general than its secret-key counterpart. In particular, a publicly verifiable computation scheme immediately implies a secret-key verifiable computation scheme supporting the same operations. As a result, designing publicly verifiable computation mechanisms is in general more challenging than in the secret-key setting. Our notion of publicly verifiable computation also generalizes the model of authenticated data structures, since authenticated data structures are essentially a special case of publicly verifiable computation where the computation supported are data structure operations.

Apart from the efficiency requirement mentioned above, another important goal of publicly verifiable computation is the ability to support expressive operations. In this paper, we build a publicly verifiable computation toolkit supporting expressive polynomial manipulations, including *evaluation* and *differentiation* on general (univariate and multivariate) polynomials over $\mathbb{Z}_p$. Our constructions are efficient and non-circuit-based.

Specifically, we consider the following setting. Suppose that a trusted source outsources a degree-d n-variate polynomial $f(\mathbf{x})$ to an untrusted server. The source also publishes a succinct signature of the polynomial which is refreshed whenever the source *updates* the polynomial. Later, a client can request to evaluate the polynomial at a specific point in $\mathbb{Z}_p^n$, or request to evaluate the k-th (partial) derivative at a given point. To facilitate verification, the untrusted server computes a succinct witness vouching for the correctness of the computation result. Our constructions allow the client to verify the outcome in asymptotically less running time than the cost of performing the computation itself. Specifically, if the client were to evaluate the polynomial or the derivative itself, it would have to perform $\Omega(\binom{n+d}{d})$ amount of work in the worst-case, since it takes $\Theta(\binom{n+d}{d})$ bits to describe an n-variate polynomial with degree at most d in the worst-case. However, using our constructions, the client's verification cost is asymptotically less than the above (see Table 1).

Other than the performance numbers shown in Table 1, we emphasize that our construction supports efficient incremental updates—it requires $O(1)$ computation and bandwidth overhead for the trusted source to update $O(1)$ number of coefficients in the polynomial.

Contributions. Our main contributions are summarized below:

1. We initiate the study and give definitions of *publicly verifiable computation*, which is a generalization of authenticated data structures [32] and secret-key verifiable computation [4, 15];

2. To the best of our knowledge, we present the first *publicly verifiable computation* scheme for operations on multivariate polynomials, such as polynomial evaluation and polynomial differentiation. Our construction implies the first verifiable computation scheme in the *secret-key* setting (i.e., in a two-party setting) for verification of polynomial differentiation;

3. We achieve verification costs that are not dependent on the total degree of the polynomial. Namely, for a polynomial with n variables of total degree d, the verification cost is $O(n)$ (and $O(n+k)$ for k-th derivative computation). This is optimal since the size of the input (the values of the n variables) is $O(n)$. With a small increase in server computation, our scheme further reduces the client verification cost in comparison with the best previous result by Benabbas et al. [4] (see Table 1).

1.1 Main techniques

Multivariate polynomial evaluation. The polynomial commitment scheme by Kate et al. [23] (see Table 1) can be employed to achieve public verifiability of univariate polynomial evaluations. However, there does not seem to be any straightforward method to extend it to the multivariate case. Specifically, Kate et al. [23] observe that to vouch for the outcome of a polynomial $f(x)$ in $\mathbb{Z}_p$ evaluated at the point $a \in \mathbb{Z}_p$, one can rely on the property that the polynomial $f(x) - f(a)$ is perfectly divisible by the degree-1 polynomial $x - a$, where $a \in \mathbb{Z}_p$. In other words, one can find a polynomial $w(x)$ such that $f(x) - f(a) = (x - a)w(x)$. Using this property, they construct a witness from the term $w(x)$, and using the pairing operation in bilinear groups, they encode the above test $f(x) - f(a) = (x - a)w(x)$ in the exponents of group elements.

Unfortunately, the test $f(x) - f(a) = (x - a)w(x)$ does not apply to the multivariate case. To allow verifiable computation for multivariate polynomials, we propose a novel technique based on the following key observation. Let $f(\mathbf{x})$ be a multivariate polynomial in $\mathbb{Z}_p$ where $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. Then for $\mathbf{a} = [a_1, a_2, \ldots, a_n] \in \mathbb{Z}_p^n$, the polynomial $f(\mathbf{x}) - f(\mathbf{a})$ can be expressed as $f(\mathbf{x}) - f(\mathbf{a}) = \sum_{i \in [n]} (x_i - a_i) w_i(\mathbf{x})$. The polynomials $w_i(\mathbf{x})$ will be used to construct witnesses in our scheme. Namely, we encode the terms in bilinear groups, as exponents of group elements. The verification is a pairing product equation encoding the above test in the exponent.

Derivative evaluation. A naive method to support verifiable derivative evaluation is for the source to commit to nk polynomials during setup, corresponding to the 1st, 2nd, ..., k-th derivatives of each possible variable. However, as noted in Section 4, this naive scheme results in increased setup and update overhead.

Our techniques for verifying the evaluation of an arbitrary derivative are inspired by the following observation that holds for first derivatives of univariate polynomials: Given a univariate polynomial $f(x)$, then the remainder of dividing the polynomial $f(x) - f'(a)x$ with the polynomial $(x - a)^2$ is always a *constant* polynomial, and not a degree-one polynomial, as would generally happen. In other words, $f(x) - f'(a)x = (x - a)^2 q(x) + b$ for some $q(x) \in \mathbb{Z}_p[x]$, and $b \in \mathbb{Z}_p$. A similar, slightly more involved, observation can be made for higher-order derivatives and multivariate polynomials. More details are provided in Section 4.

We note here that since our techniques are applied for polynomials in $\mathbb{Z}_p$, one might wonder what is the meaning of a derivative in $\mathbb{Z}_p$. It appears that derivatives in $\mathbb{Z}_p$ do have applications. For example, the *Hasse* derivative [12] of polynomials in finite fields has applications in coding theory, and is directly related to classic Newton-Leibniz derivative through a simple formula. Therefore, verification of the classic derivative in $\mathbb{Z}_p$ directly enables the verification of the Hasse derivative.

Note on the security notion. For both polynomial evaluation and derivative evaluation, the test for multivariate polynomials contains a sum of terms, as opposed to a single term in the univariate case. This gives rise to certain technicalities making the proof much trickier than the univariate case [23]. We observe that if the adversary commits to the challenge point upfront, then the simulator can carefully craft an ℓ-SBDH instance into the game with the adversary, such that if the adversary breaks the security of the verifiable computation scheme, the simulator can then break the computational assumption (see Definition 5). Therefore, while we prove *full security* for the univariate case, we use a reasonable relaxation called *selective security* for the multivariate case, since our proof requires the adversary commit to the challenge point upfront. Our selective security notion (Definition 4) is analogous to the selective security notion often adopted in the Identity-Based Encryption (IBE) [6], Attribute-Based Encryption (ABE) [22], Functional Encryption (FE) [33] and Predicate Encryption (PE) [9] literature.

1.2 Related work

Recent works on secret-key verifiable computation [1, 11, 15] achieve operation sensitive verification of general boolean circuits, where "operation sensitive" means that the bandwidth required is asymptotically minimal, i.e., asymptotically the same as the size of the query and the size of the answer. Although such approaches cover polynomial operations as a special case, thus providing verifiability, they are inherently inadequate to meet our goals of 1) *public verifiability*, since a secret key is required for verification; and 2) *dynamic updates*, since the description of the circuit is fixed at the initialization of the scheme—both important properties in real-world scenarios. Moreover, due to the use of primitives such as *fully-homomorphic*

Scheme	**Source Setup**	**Server Query**	**Client Verification**	**Publicly Verifiable**
polynomial evaluation: univariate polynomial of degree d				
Kate et al. [23]	$O(d)$	$O(d)$	$O(1)$	yes
polynomial evaluation: n-variate polynomial of degree d				
Benabbas et al. [4]	$O\left(\binom{n+d}{d}\right)$	$O\left(\binom{n+d}{d}\right)$	$O(n\log d)$	no
This paper	$O\left(\binom{n+d}{d}\right)$	$O\left(n\binom{n+d}{d}\min\{\log n, d\}\right)$	$O(n)$	yes
k-th derivative evaluation: n-variate polynomial of degree d				
This paper	$O\left(\binom{n+d}{d}\right)$	$O\left(n\binom{n+d}{d}\min\{\log n, d\} + d\log d\right)$	$O(n+k)$	yes

Table 1: **Asymptotic performance (running times) of our scheme in comparison with related work.** The numbers shown represent worst-case performance. Appendix B.2 discusses some practical optimizations one can do to achieve much better practical performance for polynomials of smaller sizes. Note that the storage requirement at the server is $O(\binom{n+d}{d})$ since this is the worst-case size of an n-variate polynomial whose total degree is bounded by d. We require no permanent storage at the client, since all necessary cryptographic information required for verification can be signed by the trusted source, stored at the server, and downloaded by the client on the fly.

encryption [16], their practicality is unclear.

Related to this paper is the work on authenticated data structures whose goal is to provide publicly verifiable solutions for data structure problems, such as dictionaries [19, 27], graphs [21, 25], hash tables [31, 35] and sets [32]. The great majority of authenticated data structures involve the use of cryptographic hashing [2, 5, 19, 25, 26, 28] or other primitives [18, 30, 31] to hierarchically compute one or more secure digests over the outsourced data. Most authenticated data structures incur verification costs that are proportional to the time spent to produce the query answer and thus are not operation sensitive. Some bandwidth-optimal and operation-sensitive solutions have been developed for range queries [2, 20] and set operations [32].

The challenge of public verifiability also arises in *memory checking* [5, 14, 29], a fundamental approach to data integrity: Dwork et al. [14] show how to trade off reads and writes for a *secret-memory* checker but state that it is "intriguingly" difficult to achieve the same result for checkers using only *public* reliable memory. Non-membership proofs are also related, both for the RSA accumulator [24] and the bilinear-map accumulator [3, 10, 13].

Perhaps closest related works are those by Kate et al. [23] and Bennabas et al. [4]. Kate et al. [23] give a publicly verifiable commitment scheme for univariate polynomials. However, their scheme does not directly extend to multivariate polynomials. On the other hand, while Bennabas et al. [4] develop methods for operation sensitive verification of multivariate polynomial evaluation, their solution is restricted to the secret-key setting. Moreover, their verification complexities depend on the degree of the polynomial (see Table 1). Note that our construction is the first to support the efficient verification of differentiation queries—even in the secret-key setting.

2 Preliminaries, definitions and assumptions

In this section, we give necessary definitions that are going to be used in the rest of the paper. The security parameter is denoted with λ, PPT stands for *probablistic polynomial-time* and $\mathsf{neg}(\lambda)$ denotes the set of negligible functions, i.e., all the functions less than $1/p(\lambda)$, for all polynomials $p(\lambda)$. We also use bold letters for vector variables, i.e., $\mathbf{x} = [x_1,\ x_2, \ldots, x_n]$ denotes a vector of n entries $x_1, x_2, \ldots, x_n$.

We begin with the definition of a publicly verifiable computation scheme. This definition is based on an analogous definition of an *authenticated data structure scheme* [32], appropriately adjusted for the case of publicly verifiable computation.

Definition 1 (Publicly verifiable computation scheme) *A* publicly verifiable computation scheme *for a function family $\mathcal{F}$ is a tuple of six PPT algorithms* $(\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Compute}, \mathsf{Verify}, \mathsf{Update}, \mathsf{Refresh})$ *with the following specification:*

1. $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(\lambda, \mathcal{F})$*: The* KeyGen *algorithm takes as input the security parameter λ and a function family $\mathcal{F}$. It outputs a public key and a secret key pair* $(\mathsf{PK}, \mathsf{SK})$*;*

2. $\mathsf{VI}(f) \leftarrow \mathsf{Setup}(\mathsf{SK}, \mathsf{PK}, f)$*: The* Setup *algorithm takes as input the secret key* SK*, the public key* PK*, and a function $f \in \mathcal{F}$. It outputs the public verification information* $\mathsf{VI}(f)$ *for the function f;*

3. $(v, w) \leftarrow \mathsf{Compute}(\mathsf{PK}, f, \mathbf{a})$*: The* $\mathsf{Compute}$ *algorithm takes as input the public key* PK*, a function $f \in \mathcal{F}$ and a point $\mathbf{a} \in \mathsf{domain}(f)$. It outputs a pair (v, w), where $v = f(\mathbf{a})$ is the output of the function f at point $\mathbf{a}$, and w is a witness;*

4. $\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{VI}(f), \mathbf{a}, v, w)$*: The* Verify *algorithm takes as input the public key* PK*, public verification information* $\mathsf{VI}(f)$*, data point $\mathbf{a} \in \mathsf{domain}(f)$, the claimed result v and a witness w. It outputs either 0 or 1;*

5. $(\mathsf{VI}(f'), \mathsf{upd}) \leftarrow \mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{VI}(f), f')$*: The* Update *algorithm takes as input the secret key* SK*, the public key* PK*, the public verification information* $\mathsf{VI}(f)$ *for the old function f and the updated function description f'. It outputs update information* upd *and updated public verification information* $\mathsf{VI}(f')$*;*

6. $\mathsf{VI}(f') \leftarrow \mathsf{Refresh}(\mathsf{PK}, \mathsf{VI}(f), \mathsf{upd}, f')$*: The* $\mathsf{Refresh}$ *algorithm takes as input the public key* PK*, the public verification information* $\mathsf{VI}(f)$ *for the old function f, the update information* upd *and the updated function description f'. It outputs the updated public verification information* $\mathsf{VI}(f')$*.*

In is important to note in this definition that algorithm Verify is required *not* have access to the secret key SK, effectively turning the scheme into *publicly-verifiable*. Moreover, note the difference between Update and $\mathsf{Refresh}$: Algorithm $\mathsf{Refresh}$ does not have access to the secret key, which serves the purpose of enabling an untrusted party to perform the update.

2.1 Protocols

In this section, we briefly describe how the above algorithms of a publicly verifiable computation scheme are applied in a three-party protocol, involving a trusted *source*, an untrusted *server* and a *client*. Note that this is exactly the setting that has been used in authenticated data structures literature (e.g., [32]).

Initially, the source wishes to outsource the computation of a function $f \in \mathcal{F}$ to an untrusted server. It runs algorithm KeyGen to output the keys of the system and then algorithm Setup that takes f as input and outputs the verification information $\mathsf{VI}(f)$, which is sent to the untrusted server, along with the original function f. The verification information contains a succinct, usually constant-size, *signed* description of the function (circuit) to be verified. The client issues queries $\mathbf{a}$ on function f. On input a query $\mathbf{a}$, the server computes the answer $f(\mathbf{a})$ and a witness w by using algorithm $\mathsf{Compute}$. The server then sends $f(\mathbf{a})$ and w to the client. Finally, the client executes algorithm Verify, which takes as input $\mathbf{a}$, $f(\mathbf{a})$, w and $\mathsf{VI}(f)$, in order to *publicly* verify (either accept or reject) the correctness of $f(\mathbf{a})$, based only on the freshness and correctness of $\mathsf{VI}(f)$ that is published by the source. How to ensure freshness and correctness of $\mathsf{VI}(f)$ has been studied in classic settings (e.g., time-stamped signatures [34]), and is outside the scope of this paper. However, we provide some suggestions in Appendix D.

The Update algorithm allows the trusted source to update the function f to some new function f'. It outputs update description upd and the new verification information $\mathsf{VI}(f')$ containing the signed digest of the new function f', which will later be used to publicly verify the correctness of computing the updated function f'. A naive way to realize the Update algorithm is to simply run the Setup algorithm again for the new f'. However, in practice, one may wish to allow more efficient incremental update. Finally, the $\mathsf{Refresh}$ algorithm is run by the untrusted server and has the same functionality with Update, with the difference that it does not use the secret key.

2.2 Correctness and security definitions

We describe now in this section the correctness and security definitions for a publicly verifiable computation scheme. Intuitively, a publicly verifiable computation scheme is correct if whenever its algorithms are executed honestly, it never rejects a correct answer. Also, it is secure if the adversary cannot persuade a verifier to accept a wrong computational result, except with negligible probability.

Definition 2 (Correctness of a publicly verifiable computation scheme) *Let λ be the security parameter, and $\mathcal{P} = (\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Compute}, \mathsf{Verify}, \mathsf{Update}, \mathsf{Refresh})$ be a publicly verifiable computation scheme for a function family $\mathcal{F}$. We say that $\mathcal{P}$ is* correct*, if the following holds. Let $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(\lambda, \mathcal{F})$. For all $i = 1, \ldots, \mathsf{poly}(\lambda)$, for any function $f_i \in \mathcal{F}$, suppose $\mathsf{VI}(f_i)$ is the output of either $(\mathsf{VI}(f_i), \mathsf{upd}) \leftarrow \mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{VI}(f_{i-1}), f_i)$ or $\mathsf{VI}(f_i) \leftarrow \mathsf{Refresh}(\mathsf{PK}, \mathsf{VI}(f_{i-1}), \mathsf{upd}, f_i)$, where $\mathsf{VI}(f_0)$ is output by algorithm $\mathsf{Setup}(\mathsf{SK}, \mathsf{PK}, f_0)$ for some $f_0 \in \mathcal{F}$. For any $i = 0, \ldots, \mathsf{poly}(\lambda)$ and for any $\mathbf{a} \in \mathsf{domain}(f_i)$, let $(v, w) \leftarrow \mathsf{Compute}(\mathsf{PK}, f_i, \mathbf{a})$. Then, we have $1 \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{VI}(f_i), \mathbf{a}, v, w)$.*

In the following, we give two security definitions. The first (full security) can be used to prove security for the case of univariate polynomials. The second (selective security) is used for the case of multivariate polynomials and is a reasonable relaxation of the first. This relaxation requires the adversary to commit ahead of time to the challenge point $\mathbf{x}$, which is analogous to the selective security notion often adopted in the Identity-Based Encryption (IBE) [6], Attribute-Based Encryption (ABE) [22], Functional Encryption (FE) [33] and Predicate Encryption (PE) [9] literature.

Definition 3 (Full security of a publicly verifiable computation scheme) *Let λ be the security parameter and $\mathcal{P} = (\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Compute}, \mathsf{Verify}, \mathsf{Update}, \mathsf{Refresh})$ be a publicly verifiable computation scheme for a function family $\mathcal{F}$. We say that $\mathcal{P}$ is* fully-secure *if no PPT adversary $\mathcal{A}$ has more than negligible probability $\mathsf{neg}(\lambda)$ in winning the following security game played between $\mathcal{A}$ and a challenger:*

1. ***Initialization.*** *The challenger runs the KeyGen algorithm which outputs $(\mathsf{PK}, \mathsf{SK})$. The challenger gives PK to the adversary $\mathcal{A}$ but maintains SK secret;*

2. ***Update queries.*** *The adversary initially makes an oracle query to the $\mathsf{Setup}(\mathsf{SK}, \mathsf{PK}, f_0)$ algorithm, specifying an initial function $f_0 \in \mathcal{F}$, outputting $\mathsf{VI}(f_0)$. Then, for $i = 1, \ldots, \mathsf{poly}(\lambda)$, he adaptively makes a polynomial number of oracle queries to the $\mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{VI}(f_{i-1}), f_i)$ algorithm, each time specifying $f_i \in \mathcal{F}$. The challenger answers the queries by returning the resulting $\mathsf{VI}(f_i)$;*

3. ***Challenge.*** *The adversary $\mathcal{A}$ outputs a tuple $(\mathsf{VI}, \mathbf{a}, v, w)$. The adversary $\mathcal{A}$ wins the game if $\mathsf{Verify}(\mathsf{PK}, \mathsf{VI}, \mathbf{a}, v, w) = 1$ and if one of the following is true:*
   (a) Either VI was never returned in a Setup or Update query;
   (b) Or VI was previously returned in a Setup or Update query upon input $f_j \in \mathcal{F}$ and $f_j(\mathbf{a}) \neq v$.

We note that in the above definition, Case (a) in the challenge phase refers to the event that the adversary $\mathcal{A}$ succeeds in forging verification information VI, whereas Case (b) refers to the event that the adversary claims a wrong computational result that however passes the verification test.

Definition 4 (Selective security of a publicly verifiable computation scheme) *Let λ be the security parameter and $\mathcal{P} = (\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Compute}, \mathsf{Verify}, \mathsf{Update}, \mathsf{Refresh})$ be a publicly verifiable computation scheme for a function family $\mathcal{F}$. We say that $\mathcal{P}$ is* selectively-secure *if no PPT adversary $\mathcal{A}$ has more than negligible probability $\mathsf{neg}(\lambda)$ in winning the following security game played between $\mathcal{A}$ and a challenger:*

1. ***Initialization.*** *The challenger runs the KeyGen algorithm which outputs $(\mathsf{PK}, \mathsf{SK})$. The challenger gives PK to the adversary $\mathcal{A}$ but maintains SK secret. At this point, the adversary commits to a challenge point $\mathbf{a}^*$;*

2. ***Update queries.*** *The adversary initially makes an oracle query to the $\mathsf{Setup}(\mathsf{SK}, \mathsf{PK}, f_0)$ algorithm, specifying an initial function $f_0 \in \mathcal{F}$, outputting $\mathsf{VI}(f_0)$. Then, for $i = 1, \ldots, \mathsf{poly}(\lambda)$, he adaptively makes a polynomial number of oracle queries to the $\mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{VI}(f_{i-1}), f_i)$ algorithm, each time specifying $f_i \in \mathcal{F}$. The challenger answers the queries by returning the resulting $\mathsf{VI}(f_i)$;*

3. ***Challenge.*** *The adversary $\mathcal{A}$ outputs a tuple $(\mathsf{VI}, \mathbf{a}^*, v, w)$, where $\mathbf{a}^*$ is the challenge point that he committed to in the initialization phase. The adversary $\mathcal{A}$ wins the game if $\mathsf{Verify}(\mathsf{PK}, \mathsf{VI}, \mathbf{a}^*, v, w) = 1$ and if one of the following is true:*
   - *(a) Either VI was never returned in a Setup or Update query;*
   - *(b) Or VI was previously returned in a Setup or Update query upon input $f_j \in \mathcal{F}$ and $f_j(\mathbf{a}^*) \neq v$.*

2.3 Multivariate polynomials notation

This paper will build a cryptographic toolkit supporting publicly verifiable computation for expressive manipulations over multivariate polynomials. We now define some notations for multivariate polynomials.

Multiset definitions. A multiset over some universe $\mathcal{U}$ is a generalized set comprising elements from the universe $\mathcal{U}$, where each element can appear more than once; for example, $\{1, 1, 2, 3, 3, 3\}$ is a multiset. In this paper, we use the following notation to denote multisets. Formally, a multiset $S : \mathcal{U} \to \mathbb{Z}^{\geq 0}$ is a function mapping each element in a universe $\mathcal{U}$ to its *multiplicity*. For any element x not in the multiset S, we define $S(x)$ to be 0. For example, for the multiset $\{a, a, b, c, c, c\}$, we have $S(a) = 2$, $S(b) = 1$, $S(c) = 3$; however, $S(e) = 0$ since e is not contained in the above multiset.

Let now S, T denote two multisets over some universe $\mathcal{U}$. We say that $S \subseteq T$, if $\forall a \in \mathcal{U}$, $S(a) \leq T(a)$. Finally, the *size* of a multiset S over some universe $\mathcal{U}$, denoted $|S|$, is defined as the sum of the multiplicity of all elements in the multiset S, i.e., $|S| = \sum_{a \in \mathcal{U}} S(a)$. To make the notation more compact, we denote with $\mathcal{S}_{d,n}$ the set of multisets of size at most d over the universe $\{1, 2, \ldots, n\}$.

Multivariate polynomials with multiset notation. In this paper, we consider verifiable computation for multivariate polynomials over the filed $\mathbb{Z}_p$. Let $f \in \mathbb{Z}_p[x_1, x_2, \ldots, x_n]$ be an n-variate polynomial over $\mathbb{Z}_p$ with maximum degree d. We can use the following generic notation to represent f, i.e.,

$$f(x_1, x_2, \ldots, x_n) = \sum_{S \in \mathcal{S}_{d,n}} c_S \cdot \prod_{i \in S} x_i^{S(i)} . \tag{2.1}$$

For example, the multiset $\{1, 1, 2, 2, 2, 5\}$ corresponds to the term for $x_1^2 x_2^3 x_5$ in the expanded form of the polynomial. The empty multiset $\emptyset$ corresponds to the constant term in the polynomial.

The degree of a multivariate polynomial is the maximum total degree of any monomial contained in the polynomial. For example, the degree of the polynomial $3x_1x_2 + x_3^3 x_4 x_5$ is 5.

2.4 Bilinear groups and computational assumptions

We now review some background on bilinear groups of prime order. Let $\mathbb{G}$ be a cyclic multiplicative group of prime order p, generated by g. Let also $\mathbb{G}_T$ be a cyclic multiplicative group with the same order p and $\mathsf{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear pairing with the following properties: (1) Bilinearity: $\mathsf{e}(P^a, Q^b) = \mathsf{e}(P, Q)^{ab}$ for all $P, Q \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$; (2) Non-degeneracy: $\mathsf{e}(g, g) \neq 1$; (3) Computability: There is an efficient algorithm to compute $e(P, Q)$ for all $P, Q \in \mathbb{G}$. We denote with $(p, \mathbb{G}, \mathbb{G}_T, \mathsf{e}, g)$ the bilinear pairings parameters, output by a PPT algorithm on input 1^λ.

The assumption that we are going to use in order to prove security of our construction is the following:

Definition 5 (Bilinear ℓ-strong Diffie-Hellman assumption) *Suppose λ is the security parameter and let $(p, \mathbb{G}, \mathbb{G}_T, \mathsf{e}, g)$ be a uniformly randomly generated tuple of bilinear pairings parameters. Given the elements $g, g^t, \ldots, g^{t^\ell} \in \mathbb{G}$ for some t chosen at random from $\mathbb{Z}_p^*$, where $\ell = \mathsf{poly}(\lambda)$, there is no polynomial-time algorithm that can output the pair $(c, \mathsf{e}(g, g)^{1/(t+c)}) \in \mathbb{Z}_p^* \backslash \{-t\} \times \mathbb{G}_T$ except with negligible probability* $\mathsf{neg}(\lambda)$.

3 Construction for multivariate polynomial evaluation

In this section we present the construction of a publicly-verifiable computation scheme for multivariate polynomial evaluation. Before we start the presentation of the algorithms of the scheme in Definition 1, we give some preliminary results. Our construction relies on the following key observation stated in Lemma 1.

Lemma 1 *Let $f(\mathbf{x}) \in \mathbb{Z}_p[\mathbf{x}]$ denote an n-variate polynomial over $\mathbb{Z}_p$. Then, $f(\mathbf{x})$ evaluates to 0 at $\mathbf{a} \in \mathbb{Z}_p^n$, i.e., $f(\mathbf{a}) = 0$, if and only if $f(\mathbf{x})$ can be expressed in the form $f(\mathbf{x}) = \sum_{i \in [n]} (x_i - a_i) q_i(\mathbf{x})$, where for all $i \in [n]$, $q_i(\mathbf{x}) \in \mathbb{Z}_p[\mathbf{x}]$ is some polynomial over $\mathbb{Z}_p$.*

Proof: The direct of the above claim is straightforward: If a polynomial $f(\mathbf{x})$ can be expressed as $\sum_{i \in [n]} (x_i - a_i) q_i(\mathbf{x})$, it evaluates to 0 at $\mathbf{a}$. For the inverse, the proof is by explicit construction. Given a polynomial $f(\mathbf{x}) \in \mathbb{Z}_p[\mathbf{x}]$, we use polynomial division to first divide $f(\mathbf{x})$ by $(x_1 - a_1)$. Specifically,

$$f(x_1, x_2, \ldots, x_n) = (x_1 - a_1) \cdot q_1(x_1, x_2, \ldots, x_n) + r_1(x_2, x_3, \ldots, x_n) \,,$$

where $r_1(x_2, x_3, \ldots, x_n)$ is the remainder term, and $r_1(x_2, x_3, \ldots, x_n)$ should no longer contain the variable x_1. Next, divide $r_1(x_2, x_3, \ldots, x_n)$ by $(x_2 - a_2)$, and divide the remainder by $(x_3 - a_3)$, and so on. In this way, we can write $f(\mathbf{x})$ as

$$f(\mathbf{x}) = \sum_{i \in [n]} (x_i - a_i) q_i(\mathbf{x}) + r_n \,,$$

where $r_n \in \mathbb{Z}_p$. Now since $f(\mathbf{a}) = 0$, r_n has to be 0, since otherwise, $f(\mathbf{a})$ would not evaluate to 0. ■

We now give a useful corollary to be used in the construction of our scheme:

Corollary 1 *Let $f(\mathbf{x}) \in \mathbb{Z}_p[\mathbf{x}]$ denote an n-variate polynomial over $\mathbb{Z}_p$. For $\mathbf{a} \in \mathbb{Z}_p^n$, $f(\mathbf{x}) - f(\mathbf{a})$ evaluates to 0 at $\mathbf{a}$. Therefore, there exist polynomials $q_i(\mathbf{x}) \in \mathbb{Z}_p[\mathbf{x}]$ for $i \in [n]$ such that $f(\mathbf{x}) - f(\mathbf{a}) = \sum_{i \in [n]} (x_i - a_i) q_i(\mathbf{x})$.*

Intuition. Given an n-variate polynomial $f(\mathbf{x})$, the trusted source first creates a digest of the polynomial by picking a random point $\mathbf{t} \in \mathbb{Z}_p^n$ (referred to as the *commitment point*), and evaluating the polynomial at $\mathbf{t}$. The resulting value $f(\mathbf{t})$ will be used to compute the digest $g^{f(\mathbf{t})}$, which the trusted source signs and publishes. The digest $g^{f(\mathbf{t})}$ together with its signature comprise $\mathsf{VI}(f)$. Later in the computation stage, when a server wishes to prove that v is indeed the value $f(\mathbf{a})$, it will rely on the key observation stated in Corollary 1. Basically, it will find n polynomials $q_1(\mathbf{x}), q_2(\mathbf{x}), \ldots, q_n(\mathbf{x})$ such that the relation of Corollary 1 holds, and the values $g^{q_i(\mathbf{t})}$ ($i = 1, \ldots, n$) will be provided as witnesses. To allow the server to evaluate the polynomials $q_i(\mathbf{x})$ at the commitment point $\mathbf{t}$ in the exponent, the public key must contain appropriate helper terms. If the claimed computation result v is correct, then the following must be true, where both sides of the equation are evaluated at the commitment point $\mathbf{t}$.

$$f(\mathbf{t}) - v = \sum_{i \in [n]} (t_i - a_i) q_i(\mathbf{t})\,. \tag{3.2}$$

Since in the real construction, the terms in the above equation are encoded in the exponents of group elements, the verifier cannot directly check the above equation. However, since bilinear groups allow us to express polynomials of degree-2 in the exponent, the verifier can check the above condition using operations in the bilinear group, including the pairing operation which allows one to express one multiplication. The bilinear group operations directly translate to checking the above condition (Equation 3.2) in the exponent.

3.1 Algorithms of the scheme

As we will see, the security of our construction relies on the ℓ-SBDH assumption (see Definition 5) and the security of a standard signature scheme satisfying existential unforgeability under adaptive chosen message attacks (e.g., [17]). Let Σ be such a signature scheme. We use the notation $(\Sigma.\mathsf{sk}, \Sigma.\mathsf{pk}) \leftarrow \Sigma.\mathsf{Key}(\lambda)$, $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$, $\{0,1\} \leftarrow \Sigma.\mathsf{Ver}(\Sigma.\mathsf{pk}, \sigma, \mathsf{msg})$ to denote the key generation, signing, and verification algorithms of the signature scheme respectively. This signature scheme will be used by the trusted source to sign some "digest" of the polynomial f. The digest and the signature will be included in the verification information $\mathsf{VI}(f)$.

Algorithm $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(\lambda, \mathcal{F})$**:** Suppose that the function family $\mathcal{F} \subseteq \mathbb{Z}_p[x_1, x_2, \ldots, x_n]$ represents all polynomials over $\mathbb{Z}_p$ with at most n variables and degree bounded by d. Namely, family $\mathcal{F}$ contains the polynomials represented by multisets in set $\mathcal{S}_{n,d}$ (see Relation 2.1) The KeyGen algorithm does the following operations: (a) It invokes the key generation algorithm of the signature scheme $(\Sigma.\mathsf{sk}, \Sigma.\mathsf{pk}) \leftarrow \Sigma.\mathsf{Key}(\lambda)$; (b) Next, it invokes the bilinear group generation algorithm to generate a bilinear group instance of prime order p (of λ bits), with a bilinear map function $\mathsf{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$; (c) Then it chooses a random generator $g \in \mathbb{G}$; (d) Then it chooses a random point $\mathbf{t} = [t_1, t_2, \ldots, t_n] \in \mathbb{Z}_p^n$ and computes the *witness generation set* $\mathcal{W}_{n,d}$ as below:

$$\mathcal{W}_{n,d} = \left\{ g^{\prod_{i \in S} t_i^{S(i)}} : \forall S \in \mathcal{S}_{n,d} \right\}\,. \tag{3.3}$$

For example, the witness generation set $\mathcal{W}_{2,2}$ contains the elements $g, g^{t_1}, g^{t_2}, g^{t_1^2}, g^{t_2^2}, g^{t_1 t_2^2}, g^{t_1^2 t_2}, g^{t_1^2 t_2^2}$. The algorithm finally outputs the public key PK that contains $\Sigma.\mathsf{pk}$, g, $\mathcal{W}_{n,d}$ and the description of $\mathbb{G}, \mathbb{G}_T, \mathsf{e}$. The secret key SK contains the signature signing key $\Sigma.\mathsf{sk}$ and the commitment point $\mathbf{t}$. We describe an optimization referring to the size of the size of $\mathcal{W}_{n,d}$ in Section B.2 in the Appendix.

Algorithm $\mathsf{VI}(f) \leftarrow \mathsf{Setup}(\mathsf{SK}, \mathsf{PK}, f)$: Let $f(\mathbf{x}) \in \mathbb{Z}_p[x_1, \ldots, x_n]$ denote an n-variate polynomial of maximum degree d over $\mathbb{Z}_p$ that is represented by the multisets $S_1, S_2, \ldots, S_k \in \mathcal{S}_{n,d}$ and the respective coefficients $c_1, c_2, \ldots, c_k \in \mathbb{Z}_p$ (the polynomial has k terms), as defined in Relation 2.1. The setup algorithm, by using the witness generation set $\mathcal{W}_{n,d}$ contained in PK, computes the *digest* of the polynomial, i.e.,

$$\mathsf{digest}(f) = g^{f(\mathbf{t})} = \left(g^{\prod_{i \in S_1} t_i^{S_1(i)}} \right)^{c_1} \times \left(g^{\prod_{i \in S_2} t_i^{S_2(i)}} \right)^{c_2} \times \ldots \times \left(g^{\prod_{i \in S_k} t_i^{S_k(i)}} \right)^{c_k}\,. \tag{3.4}$$

The algorithm outputs the public verification information $\mathsf{VI}(f)$ to contain $\mathsf{digest}(f)$ and its signature $\Sigma.\mathsf{Sign}(\mathsf{sk}, \mathsf{digest}(f))$.

Algorithm $(v, w) \leftarrow \mathsf{Compute}(\mathsf{PK}, f, \mathbf{a})$**:** The $\mathsf{Compute}$ algorithm first computes $v = f(\mathbf{a})$. Then, and due to Corollary 1, it finds an appropriate set of polynomials $q_1(\mathbf{x}), q_2(\mathbf{x}), \ldots, q_n(\mathbf{x})$ to express the polynomial $f(\mathbf{x}) - v$ as

$$f(\mathbf{x}) - v = \sum_{i \in [n]} (x_i - a_i) q_i(\mathbf{x}) .$$

The witness w is a vector of n witnesses $w_1, w_2, \ldots, w_n$, such that $w_i = g^{q_i(\mathbf{t})}$ for all $i \in [n]$. Note that w_i can easily be computed using the witness generation set $\mathcal{W}_{n,d}$, as is achieved for the digest in Relation 3.4. It finally outputs the pair (v, w) denoting the outcome of the polynomial evaluated at $\mathbf{a}$, and a witness to vouch for the correctness of the computation.

Algorithm $\mathsf{Verify}(\mathsf{PK}, \mathsf{VI}(f), \mathbf{a}, v, w)$**:** Parse PK as the public key of the signature scheme $\Sigma.\mathsf{pk}$ and the witness generation set $\mathcal{W}_{n,d}$ and $\mathsf{VI}(f)$ as the digest digest and the respective signature σ. To verify that v is indeed the outcome of the correct polynomial evaluated at point $\mathbf{a} \in \mathbb{Z}_p^n$, given a witness $w = [w_1, w_2, \ldots, w_n]$, algorithm Verify checks if the following equations hold:

$$\Sigma.\mathsf{Ver}(\Sigma.\mathsf{pk}, \sigma, \mathsf{digest}) \overset{?}{=} 1 \quad \text{and} \quad \prod_{i=1}^{n} \mathsf{e}\left(g^{t_i} \cdot g^{-a_i}, w_i\right) \overset{?}{=} \mathsf{e}\left(\mathsf{digest} \cdot g^{-v}, g\right) .$$

In the above, the terms g^{t_i} are contained in PK (specifically in $\mathcal{W}_{n,d}$) and digest equals $g^{f(\mathbf{t})}$ (with all but negligible probability) if the verification of the signature is successful. The algorithm accepts the computation result v, and outputs 1 if the above equations hold; otherwise, it rejects and outputs 0.

Algorithm $(\mathsf{VI}(f'), \mathsf{upd}) \leftarrow \mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{VI}(f), f')$**:** Let f denote the current polynomial and f' be the new polynomial that corresponds to the update. Assume that f' and f differ in only one coefficient. Specifically, let S denote the multiset corresponding to that coefficient.[1]

Parse the secret key as $\mathsf{SK} = (\Sigma.\mathsf{sk})$. Suppose now that the current verification information is $\mathsf{VI}(f) = (\mathsf{digest}(f), \sigma)$. The algorithm computes

$$\mathsf{digest}(f') \leftarrow \mathsf{digest}(f) \cdot g^{(c'_S - c_S) \prod_{i \in S} t_i^{S(i)}} \quad \text{and} \quad \sigma' \leftarrow \Sigma.\mathsf{Sign}(\Sigma.\mathsf{sk}, \mathsf{digest}(f')) ,$$

updating in this way $\mathsf{VI}(f)$ to $\mathsf{VI}(f')$. Notice that the term $g^{\prod_{i \in S} t_i^{S(i)}}$ is included in the public key PK. Finally, the update information upd output by the algorithm is simply $\mathsf{VI}(f')$.

Algorithm $\mathsf{VI}(f') \leftarrow \mathsf{Refresh}(\mathsf{PK}, \mathsf{VI}(f), \mathsf{upd}, f')$**:** The $\mathsf{Refresh}$ algorithm outputs upd as $\mathsf{VI}(f')$ and also updates the description of the function to f'.

We now give the final result of this section:

Theorem 1 (Publicly verifiable computation scheme for polynomial evaluation) *There exists a publicly verifiable computation scheme* $\{\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Compute}, \mathsf{Verify}, \mathsf{Update}, \mathsf{Refresh}\}$ *for the evaluation of n-variate polynomials of total degree d, such that (1) It is correct according to Definition 2; (2) For $n = 1$, it is fully-secure according to Definition 3 and under the the ℓ-SBDH assumption; (3) For $n > 1$, it is selectively-secure according to Definition 4 and under the the ℓ-SBDH assumption; (4) Algorithm* Setup *runs in $O(\binom{n+d}{d})$ time; (5) Algorithm* $\mathsf{Compute}$ *runs in $O(n\binom{n+d}{d} \min\{\log n, d\})$ time, outputting a witness of $O(n)$ size; (6) Algorithm* Verify *runs in $O(n)$ time; and (7) Algorithms* Update *and* $\mathsf{Refresh}$ *run in $O(1)$ time for updating $O(1)$ number of coefficients.*

The correctness of our construction follows in a straightforward manner from Corollary 1, and the bilinear property of the pairing operation e. We present the security proofs for the multivariate case (Part (3) of Theorem 1) in Appendix C.1. We present the security proofs for the univariate case (Part (2) of Theorem 1) in Appendix C.3. The asymptotic performance analysis is provided in Appendix A.1.

[1]Namely, the only difference between f and f' is that the coefficient c_S corresponding to the term $\prod_{i \in S} x_i^{S(i)}$ is updated to c'_S in f'.

4 Construction for multivariate polynomial differentiation

In this section, we construct a publicly verifiable computation scheme for the verification of differentiation queries. Namely, given a polynomial $f(\mathbf{x})$, we derive efficient verification methods for the k-th partial derivative at some point $\mathbf{a}$, i.e., the computation $\partial^k f(\mathbf{x})/\partial x_j^k(\mathbf{a})$.

One naive method to support derivative computation is to commit to all nk polynomials corresponding to all the possible derivatives (k in total) of each possible variable. This would incur a setup cost of $O(nk\binom{n+d}{d})$, while our construction requires only $O(\binom{n+d}{d})$ setup cost, the same with the simple polynomial evaluation. Another drawback of the naive method is increased update cost, since an update operation would now involve updating all nk polynomials.[2]

We now explain our novel construction supporting differentiation queries on polynomials. We will rely on a useful property of the k-th derivative for polynomials. To provide the necessary intuition, we first state the property for the univariate case as in Lemma 2.

Lemma 2 (Property of k-th derivative of a univariate polynomial) *Let $f(x)$ denote a univariate polynomial over $\mathbb{Z}_p$. For $a \in \mathbb{Z}_p$, $f(x)$ can be expressed as $f(x) = (x-a)^{k+1}q(x) + c_k x^k + c_{k-1}x^{k-1} + \ldots + c_1 x + c_0$, where $c_1, c_2, \ldots, c_k \in \mathbb{Z}_p$. Then, the k-th derivative of $f(x)$ evaluates to $k! \cdot c_k$ at point a, i.e., $\frac{\partial^k f(x)}{\partial x^k}(a) = k! \cdot c_k$.*

Proof: Let $g(x) = (x-a)^{k+1}$. First, observe that for all $0 \le j \le k$, $\frac{\partial^j g(x)}{\partial x^j}(a) = 0$. In the above the 0-th derivative of a polynomial is the polynomial itself. The k-th derivative of a product of terms $g(x)q(x)$ can be expressed as below:

$$\frac{\partial^k g(x)q(x)}{\partial x^k} = g(x)\frac{\partial^k q(x)}{\partial x^k} + \binom{k}{1}\frac{\partial q(x)}{\partial x}\frac{\partial^{k-1}q(x)}{\partial x^{k-1}} + \binom{k}{2}\frac{\partial^2 q(x)}{\partial x^2}\frac{\partial^{k-2}q(x)}{\partial x^{k-2}} + \ldots + \frac{\partial^k g(x)}{\partial x^k}q(x)\,.$$

Due to the fact that $\frac{\partial^j g(x)}{\partial x^j}(a) = 0$ for all $0 \le j < k$, we have $\frac{\partial^k g(x)q(x)}{\partial x^k}(a) = 0$. Suppose now $r(x) = c_k x^k + c_{k-1}x^{k-1} + \ldots + c_1 x + c_0$. Then, we have $\frac{\partial^k f(x)}{\partial x^k}(a) = \frac{\partial^k (g(x)\cdot q(x))}{\partial x^k}(a) + \frac{\partial^k r(x)}{\partial x^k}(a) = \frac{\partial^k r(x)}{\partial x^k}(a) = k! \cdot c_k$. ■

We generalize now this property to the multivariate case.

Lemma 3 (Property of partial k-th derivative of a multivariate polynomial) *Let $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, and let $f(\mathbf{x}) \in \mathbb{Z}_p[\mathbf{x}]$ denote a multivariate polynomial over $\mathbb{Z}_p$. Then, for $\mathbf{a} = [a_1, a_2, \ldots, a_n] \in \mathbb{Z}_p^n$, $f(\mathbf{x})$ can be expressed as*

$$f(\mathbf{x}) = \sum_{i=2}^{n}(x_i - a_i)u_i(\mathbf{x}) + (x_1 - a_1)^{k+1}q(x_1) + c_k x_1^k + c_{k-1}x_1^{k-1} + \ldots + c_1 x_1 + c_0\,. \qquad (4.5)$$

Then, the k-th derivative of $f(\mathbf{x})$ with respect to x_1 evaluates to $k! \cdot c_k$ at point $\mathbf{a}$, i.e., $\frac{\partial^k f(\mathbf{x})}{\partial x_1^k}(\mathbf{a}) = k! \cdot c_k$. A similar result holds for derivatives with respect to any variable x_i by variable renaming.

Proof: The proof is by explicit construction. The polynomial $u_2(\mathbf{x})$ is the quotient when dividing $f(\mathbf{x})$ by $(x_2 - a_2)$, the remainder is then divided by $(x_3 - a_3)$, resulting in the quotient $u_3(\mathbf{x})$, and the remainder is then divided by $(x_4 - a_4)$, and so on. This goes on until we divide the remainder with $(x_n - a_n)$, at which point, we are left with a remainder $r(x_1)$ containing only the variable x_1. At this point, we divide $r(x_1)$ by $(x_1 - a_1)^{k+1}$ resulting in the quotient $q(x_1)$, and the remainder is expressed as $c_k x_1^k + c_{k-1}x_1^{k-1} + \ldots +$

[2]Such a solution would also increase the communication cost during an update between the source and the untrusted server. Namely, every time the source would update a coefficient, it would need to communicate $O(nk)$ signatures, whereas in our scheme there is a need only for one signature.

$c_1x_1 + c_0$. We now show that $\partial^k f(\mathbf{x})/\partial x_1^k(\mathbf{a}) = k! \cdot c_k$. To do this, we analyze the k-th derivative with respect to x_1 for each additive term of $f(\mathbf{x})$ expressed in the above form. Notice that for $2 \le i \le n$, we have

$$\frac{\partial^k (x_i - a_i)u_i(\mathbf{x})}{\partial x_1^k}(\mathbf{a}) = (x_i - a_i)\frac{\partial^k u_i(\mathbf{x})}{\partial x_1^k}(\mathbf{a}) = 0\,.$$

Let $g(x_1) = (x_1 - a_1)^{k+1}$. As proven in Lemma 2, we have $\frac{\partial^k g(x_1)q(x_1)}{\partial x_1^k}(a_1) = 0$. Also notice that for all polynomials whose degree in x_1 is smaller than k, its k-th derivative with respect to x_1 is 0. As a result, $\frac{\partial^k f(\mathbf{x})}{\partial x_1^k}(\mathbf{a}) = \frac{\partial^k c_k x_1^k}{\partial x_1^k}(\mathbf{a}) = k! \cdot c_k$. ■

Intuition. Similar to the construction for polynomial evaluation, the trusted source computes a digest $g^{f(\mathbf{t})}$ for the polynomial at a random commitment point $\mathbf{t} \in \mathbb{Z}_p^n$. The digest is signed and published. In the computation stage, for the server to prove that the claimed result v is indeed the k-th derivative with respect to some variable x_1, evaluated at $\mathbf{a} \in \mathbb{Z}_p^n$, i.e., $v = \partial^k f(\mathbf{x})/\partial x_1^k(\mathbf{a})$, the server first expresses the polynomial as in Equation 4.5 and then provides the following witnesses

$$\left\{g^{u_i(\mathbf{t})}, i = 2, \ldots, n\right\} \in \mathbb{G}^{n-1} \quad \text{and} \quad g^{q(t_1)} \in \mathbb{G} \quad \text{and} \quad c_0, c_1, \ldots, c_{k-1} \in \mathbb{Z}_p\,.$$

The client can then check that Equation 4.5 holds, when both sides are evaluated at the commitment point $\mathbf{t}$. Since terms in the relation of Equation 4.5 are encoded as exponents of bilinear group elements, the verification is through bilinear group operations, which translates to algebra in the exponents.

4.1 Algorithms of the scheme

In this section we provide the detailed construction for the verification of derivatives evaluation. Algorithms KeyGen, Setup, Update and Refresh are exactly the same with the case of multivariate polynomial evaluation in Section 3. We describe in detail algorithms Compute and Verify.

Algorithm $(v, w) \leftarrow \mathsf{Compute}(\mathsf{PK}, f, \mathbf{a}, k, i)$**:** The Compute algorithm here takes in two additional parameters k and i, indicating the evaluation of the k-th derivative of the polynomial with respect to variable x_i at $\mathbf{a}$. Without loss of generality, below we assume $i = 1$. In other words, the algorithm should evaluate $\frac{\partial^k f(\mathbf{x})}{\partial x_1^k}(\mathbf{a})$. Due to Lemma 3, $f(\mathbf{x})$ can be expressed as

$$f(\mathbf{x}) = \sum_{i=2}^{n}(x_i - a_i)u_i(\mathbf{x}) + (x_1 - a_1)^{k+1}q(x_1) + c_kx_1^k + c_{k-1}x_1^{k-1} + \ldots + c_1x_1 + c_0\,.$$

The witness w for the derivative computation is the following tuple:

$$\left(g^{u_2(\mathbf{t})}, g^{u_3(\mathbf{t})}, \ldots, g^{u_n(\mathbf{t})}, g^{q(t_1)}, c_{k-1}, \ldots, c_1, c_0\right) \in \mathbb{G}^n \times \mathbb{Z}_p^k\,.$$

Finally, the result of the computation is given by $v = k! \cdot c_k$.

Algorithm $\mathsf{Verify}(\mathsf{PK}, \mathsf{VI}(f), \mathbf{a}, v, w, k, i)$**:** Let $c_k = \frac{v}{k!}$. Parse PK as the public key of the signature scheme $\Sigma.\mathsf{pk}$ and the witness generation set $\mathcal{W}_{n,d}$ and $\mathsf{VI}(f)$ as the digest digest and the respective signature σ. To verify that v is indeed the outcome of the k-th partial derivative on varaible x_1 evaluated at point $\mathbf{a} \in \mathbb{Z}_p^n$, perform the following steps. Parse w as $(w_2, \ldots, w_n, \omega, c_{k-1}, \ldots, c_0)$. Given the witness w, algorithm Verify checks if the following equations hold:

$$\Sigma.\mathsf{Ver}(\Sigma.\mathsf{pk}, \sigma, \mathsf{digest}) \overset{?}{=} 1 \quad \text{and} \quad \mathsf{e}\left(\mathsf{digest}, g\right) \overset{?}{=} \prod_{i=2}^{n} \mathsf{e}\left(g^{t_i}g^{-a_i}, w_i\right) \cdot \mathsf{e}\left(g^{(t_1-a_1)^{k+1}}, \omega\right) \cdot \prod_{i=0}^{k} \mathsf{e}\left(g^{t_1^i}, g\right)^{c_i}\,.$$

In the above, all the used expressions are computable by using the elements in the witness generation set $\mathcal{W}_{n,d}$. Also, digest equals $g^{f(\mathbf{t})}$ (with all but negligible probability) if the verification of the signature is successful. The algorithm accepts the computation result v, and outputs 1 if the above equations hold; otherwise, it rejects and outputs 0.

Theorem 2 (Publicly verifiable computation scheme for polynomial differentiation) *There exists a publicly verifiable computation scheme* {KeyGen, Setup, Compute, Verify, Update, Refresh} *for the differentiation (k-th partial derivative) of n-variate polynomials of total degree d, such that (1) It is correct according to Definition 2; (2) For $n = 1$, it is fully-secure according to Definition 3 and under the the ℓ-SBDH assumption; (3) For $n > 1$, it is selectively-secure according to Definition 4 and under the the ℓ-SBDH assumption; (4) Algorithm* Setup *runs in $O(\binom{n+d}{d})$ time; (5) Algorithm* Compute *runs in $O(n\binom{n+d}{d} \min\{\log n, d\} + d \log d)$ time, outputting a witness of $O(n + k)$ size; (6) Algorithm* Verify *runs in $O(n + k)$ time; and (7) Algorithms* Update *and* Refresh *run in $O(1)$ time for updating $O(1)$ number of coefficients.*

The correctness of our construction follows in a straightforward manner from Lemma 3, and the bilinear property of the pairing operation e. We present the security proofs for the multivariate case (Part (3) of Theorem 2) in Appendix C.2. We present the security proofs for the univariate case (Part (2) of Theorem 2) in Appendix C.3. The asymptotic performance analysis is provided in Appendix A.2.

5 Conclusions and open problems

Motivated mainly by cloud computing applications, in this paper we initiate the study of *publicly verifiable computation*, extending and generalizing previous work on authenticated data structures and outsourced verifiable computation in the secret key setting. Some interesting and challenging open problems that stem from this work include proving the full security for the multivariate constructions and also constructing efficient publicly verifiable computation schemes for other expressive operations, such as graph and data mining algorithms. Also, it would be very challenging to construct efficient mechanisms for the verification of general boolean circuits in the public key setting.

Acknowledgments

This work was supported by the National Science Foundation under grants CCF-0424422, 0842695, 0808617 and CNS-1012060, by the Air Force Office of Scientific Research (AFOSR) under MURI award FA9550-09-1-0539, by the MURI program under AFOSR grant FA9550-08-1-0352, by the Center for Geometric Computing at Brown University and NetApp. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors. The authors thank Basilis Gidas for verifying the validity of Lemma 1.

References

[1] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 152–163. Springer, 2010.

[2] M. J. Atallah, Y. Cho, and A. Kundu. Efficient data authentication in an environment of untrusted third-party distributors. In *Proceedings of International Conference on Data Engineering (ICDE)*, 2008.

[3] M. H. Au, P. P. Tsang, W. Susilo, and Y. Mu. Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. In *Proc. Cryptographers' Track at the RSA Conference (CT-RSA)*, pages 295–308. Springer, 2009.

[4] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, pages 111–131, 2011.

[5] M. Blum, W. S. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, 12(2/3):225–244, 1994.

[6] D. Boneh and X. Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, 2004.

[7] D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, pages 149–168, 2011.

[8] D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *Public Key Cryptography*, pages 1–16, 2011.

[9] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

[10] J. Camenisch, M. Kohlweiss, and C. Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *Public Key Cryptography (PKC)*, pages 481–500, 2009.

[11] K.-M. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, 2010.

[12] V. C. da Rocha Jr. Digital sequences and the hasse derivative. *Communications Coding and Signal Processing*, 3:256–268, 1997.

[13] I. Damgård and N. Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538, 2008. `http://eprint.iacr.org/`.

[14] C. Dwork, M. Naor, G. N. Rothblum, and V. Vaikuntanathan. How efficient can memory checking be? In *Theoretical Cryptography Conference (TCC)*, pages 503–520, 2009.

[15] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[16] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[17] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17:281–308, April 1988.

[18] M. T. Goodrich, R. Tamassia, and J. Hasic. An efficient dynamic and distributed cryptographic accumulator. In *Proc. Information Security Conference (ISC)*, volume 2433 of *LNCS*, pages 372–388. Springer, 2002.

[19] M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *Proc. DARPA Information Survivability Conference and Exposition II (DISCEX II)*, pages 68–82, 2001.

[20] M. T. Goodrich, R. Tamassia, and N. Triandopoulos. Super-efficient verification of dynamic outsourced databases. In *Proc. RSA Conference, Cryptographers' Track (CT-RSA)*, volume 4964 of *LNCS*, pages 407–424. Springer, 2008.

[21] M. T. Goodrich, R. Tamassia, and N. Triandopoulos. Efficient authenticated data structures for graph connectivity and geometric search problems. *Algorithmica*, 60(3):505–552, 2011.

[22] V. Goyal, A. Jain, O. Pandey, and A. Sahai. Bounded ciphertext policy attribute based encryption. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part II*, ICALP '08, pages 579–591, 2008.

[23] A. Kate, G. Zaverucha, and I. Goldberg. Polynomial commitments. In *Asiacrypt*, 2010.

[24] J. Li, N. Li, and R. Xue. Universal accumulators with efficient nonmembership proofs. In *Proc. Applied Cryptography and Network Security (ACNS)*, pages 253–269, 2007.

[25] K. M. Man Lung Yiu, Yimin Lin. Efficient verification of shortest path search via authenticated hints. In *ICDE*, pages 237–248, 2010.

[26] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, 2004.

[27] R. C. Merkle. A certified digital signature. In G. Brassard, editor, *Proc. CRYPTO '89*, volume 435 of *LNCS*, pages 218–238. Springer-Verlag, 1989.

[28] M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proc. 7th USENIX Security Symposium*, pages 217–228, Berkeley, 1998.

[29] M. Naor and G. N. Rothblum. The complexity of online memory checking. *J. ACM*, 56(1), 2009.

[30] C. Papamanthou and R. Tamassia. Cryptography for efficiency: Authenticated data structures based on lattices and parallel online memory checking. Cryptology ePrint Archive, Report 2011/102, 2011. `http://eprint.iacr.org/`.

[31] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Authenticated hash tables. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, pages 437–448. ACM, October 2008.

[32] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Optimal verification of operations on dynamic sets. In *CRYPTO*, pages 91–110, 2011.

[33] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[34] R. Tamassia. Authenticated data structures. In *Proc. European Symp. on Algorithms*, volume 2832 of *LNCS*, pages 2–5. Springer-Verlag, 2003.

[35] R. Tamassia and N. Triandopoulos. Efficient content authentication in peer-to-peer networks. In *Proc. Int. Conf. on Applied Cryptography and Network Security (ACNS)*, volume 4521 of *LNCS*, pages 354–372. Springer, 2007.

[36] D. Wendlandt, D. Andersen, and A. Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *Proc. USENIX Annual Technical Conference*, June 2008.

Appendices

A Asymptotic performance

A.1 Construction for multivariate polynomial evaluation

Setup. In algorithm Setup, the trusted data source performs $O(\binom{n+d}{d})$ amount of computation, and transfer $O(\binom{n+d}{d})$ amount of information to the server. In the above, n is the number of variables and d is an upper bound on the degree. To see why, observe that $\binom{n+d}{d}$ is the number of n-variate monomials with degree at most d.

While this is the worst-case performance, in Section B.2, we discuss how to significantly reduce the amount of computation and bandwidth needed for the trusted source in practice for polynomials of smaller sizes.

Also, note that the verification information has size $O(n)$.

Computation and verification. In the computation phase, it is not hard to see that the communication between the client and the server is $O(n)$, and that the client performs $O(n)$ amount of work to verify the correctness of the computation result.

As for server-side computation, the server needs to decompose the polynomial according to Equation 3.2. This polynomial decomposition dominates the asymptotic performance.

To perform the polynomial decomposition, the server performs n polynomial divisions. If one uses the naive polynomial division algorithm, since each variable can have degree up to d, each polynomial division involves d steps, and each step takes time proportional to the number of terms in the polynomial, namely, $O(\binom{n+d}{d})$. Therefore, the polynomial decomposition (Equation 3.2) can be done in $O(nd\binom{n+d}{d})$ time using the naive algorithm.

In cases where $d > \log n$, one can use the FFT method to perform polynomial division, resulting in $O(n \log n\binom{n+d}{d})$ computation time.

A.2 Construction for derivative evaluation

Setup. Same as above.

Computation and Verification. It is not hard to see that the client's verification cost and the witness size are both $O(n+k)$. In particular, notice that the client needs to expand polynomials of the form $(x-a)^{k+1}$, and this can be done in $O(k)$ time. The remainder of the computation clearly can be done in $O(n+k)$ time.

As for server computation, similar to the polynomial evaluation case, the server performs a polynomial decomposition according to Equation 4.5. This decomposition can take two steps. First, decompose as below:

$$f(\mathbf{x}) = \sum_{i=2}^{n}(x_i - a_i)u_i(\mathbf{x}) + r(x_1)\,, \tag{A.6}$$

where $r(x_1)$ is a polynomial only in x_1. This step takes the same amount of time as in the polynomial evaluation case, i.e., $O(n\binom{n+d}{d}\min\{\log n, d\})$.

Next, decompose $r(x_1)$ as below:

$$r(x_1) = (x_1 - a_1)^{k+1} q(x_1) c_k x_1^k + c_{k-1}x_1^{k-1} + \ldots c_1 x_1 + c_0\,. \tag{A.7}$$

This step takes $O(d \log d)$ time, since it involves a polynomial division—we can use the FFT method for that.

B Practical optimizations and extensions

B.1 Computing derivatives in a batch

The construction for evaluating derivatives in Section 4 can be used to verifiably compute the values of the $k, k-1, k-2, \ldots, 0$-th derivatives with respect to some variable x_1 (evaluated at $\mathbf{a}$) in a single run. Specifically, observe that in Lemma 3, for $0 \le \kappa \le k$, the κ-th derivative of the function f at $\mathbf{a}$ is the following:

$$\frac{\partial^\kappa f}{\partial x_1^\kappa}(\mathbf{a}) = \sum_{i=\kappa}^{k} \frac{i!}{(i-\kappa)!} c_i a_1^{i-\kappa} \,. \tag{B.8}$$

Since the witness returned from the server contains all of $c_0, c_1, \ldots, c_{k-1}$ as well, the client can easily compute the $k-1, k-2, \ldots, 0$-th derivatives. In Appendix C.2, we show that if the ℓ-SBDH assumption holds, then the server must return all the correct $c_0, c_1, \ldots, c_{k-1}$ values.

As a result, if a client wishes to evaluate multiple derivatives with respect to the same variable, it suffices for the server to run *Compute* only once and return the corresponding witness – in other words, the additional derivatives "get a free ride".

B.2 Smaller witness generation set for smaller polynomials

In our basic constructions in Sections 3 and 4, the witness generation set $\mathcal{W}_{n,d}$ contains one term corresponding to every possible monomial with at most n variables and degree d. Therefore, the witness generation set is of size $\binom{n+d}{d}$—the number of multisets of d elements chosen among $n+1$ things (the n variables and the constant 1).

In practice, if a polynomial is smaller in size when expressed as a sum of product terms, one can potentially reduce the size of $\mathcal{W}_{n,d}$. Specifically, for every monomial $\prod_{i\in S} x_i^{S(i)}$ contained in the polynomial, represented by multiset S, the server just needs the following terms (monomials) to appear in $\mathcal{W}_{n,d}$:

$$\forall T \subseteq S : g^{\prod_{i\in T} t_i^{T(i)}} \,, \tag{B.9}$$

where $T \subseteq S$ if and only if $T(i) \le S(i)\ \forall i \in [n]$. Therefore, if each variable has $O(1)$ degree, and there are at most m terms in the polynomial, then the total size of $\mathcal{W}_{n,d}$ will be $O(m)$.

Notice that if this optimization is used, the Update algorithm needs to be modified accordingly. Specifically, if a new monomial corresponding to the multiset S is added during an update, then the trusted source also has to additionally *expand* the public key PK and compute and transfer the terms contained in Equation B.9 to the server during an update.

C Security proofs

C.1 Construction for multivariate polynomial evaluation

We now build a simulator $\mathcal{S}$ which obtains an instance of the ℓ-SBDH assumption from a challenger $\mathcal{C}$. The simulator $\mathcal{S}$ then embeds this ℓ-SBDH assumption into an instance of the verifiable computation scheme—such that if an adversary $\mathcal{A}$ can break the selective security of the verifiable computation scheme with more than negligible probability, the simulator $\mathcal{S}$ can then leverage the adversary $\mathcal{A}$ to break the ℓ-SBDH instance it obtained from from the challenger, also with non-negligible probability.

C.1.1 Proof of Part (3) of Theorem 1

Assume that the simulator $\mathcal{S}$ obtains the following ℓ-SBDH assumption from the challenger $\mathcal{C}$, where $\ell = d$ (the maximum degree of the polynomial):

$$\left(g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^d}\right) .$$

Initialization. The challenger runs the key generation algorithm, and gives the public key PK to the adversary $\mathcal{A}$. The adversary then commits to a challenge point

$$\mathbf{a}^* = [a_1^*, a_2^*, \ldots, a_n^*] .$$

Setup. When the adversary first queries the Setup algorithm, the simulator $\mathcal{S}$ performs the following. The simulator needs to choose a random point $\mathbf{t}$, evaluate the polynomial at $\mathbf{t}$, and create the corresponding witness generation set $\mathcal{W}_{n,d}$ and verification information $\mathsf{VI}(f)$ the chosen point $\mathbf{t}$. The simulator $\mathcal{S}$ will implicitly choose the point $\mathbf{t}$ as below. The simulator implicitly lets

$$t_1 = \tau . \tag{C.10}$$

For $i \in \{2, 3, \ldots n\}$, the simulator first picks random (r_i, s_i) such that

$$a_i^* = r_i \cdot a_1^* + s_i . \tag{C.11}$$

The simulator then implicitly lets

$$t_i = r_i \cdot \tau + s_i . \tag{C.12}$$

The simulator remembers the values of all r_i's for later usage.

Now the simulator needs to compute $\mathcal{W}_{n,d}$ and $\mathsf{VI}(f)$. The problem is that the simulator does not actually know the t_i's, since these values are inherited and transformed from the ℓ-SBDH assumption it obtained from the challenger. Fortunately, observe that the challenger can still compute all terms in $\mathcal{W}_{n,d}$ and $\mathsf{VI}(f)$, since when one plugs in Equations C.10 and C.12, it is not hard to see that all terms in $\mathcal{W}_{n,d}$ and $\mathsf{VI}(f)$ are essentially of the form $g^{q(\tau)}$ where $q(\tau)$ is some polynomial of degree at most d. Since the challenger knows the values of $\left(g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^d}\right)$, it can easily compute all of these terms. Notice that in the above simulation the terms generated by the simulator are identically distributed as running the real Setup algorithm.

Updates. The adversary can request update the polynomial either by calling the Setup algorithm again, or by calling the incremental update algorithm Update. As mentioned in Section 3, updates change the verification information $\mathsf{VI}(f)$. It is not hard to see that the simulator does not need to know the t_i's to update the digest in the $\mathsf{VI}(f)$. It can then use the signing key Σ.sk to sign the updated digest.

Challenge. The adversary outputs a forgery for the committed point $\mathbf{a}^*$. The forgery consists of some $\mathsf{VI}(f)$, a claimed outcome v of the polynomial at $\mathbf{a}^*$, and a witness $w = (w_1, w_2, \ldots, w_n)$. Due to the security of the signature scheme, $\mathsf{VI}(f)$ must be an output of one of the oracle queries to either Setup or Update—since otherwise, one can leverage this adversary and build a straightforward reduction to break the security of the signature scheme. Let f denote the corresponding input of that oracle query which resulted in $\mathsf{VI}(f)$. If the forgery is successful, the following must be true: $v \neq f(\mathbf{a}^*)$ and $\mathsf{Verify}(\mathsf{VI}, \mathbf{x}, v, w) = 1$. The simulator will now leverage this forgery to break the ℓ-SBDH instance it got from the challenger.

Specifically, let $c = v - f(\mathbf{a}^*) \neq 0 \in \mathbb{Z}_p$, i.e., the difference between the true outcome and the claimed outcome. Since the verification succeeds, we have

$$\mathsf{e}\,(g, g)^{f(\mathbf{t})-v} = \prod_{i \in [n]} \mathsf{e}\left(g^{t_i - a_i^*}, w_i\right) .$$

Or equivalently,

$$\mathsf{e}\,(g,g)^c = \mathsf{e}\,(g,g)^{f(\mathbf{t})-f(\mathbf{a}^*)} \cdot \prod_{i\in[n]} \mathsf{e}\left(g^{a_i^*-t_i}, w_i\right) . \tag{C.13}$$

Due to Lemma 1, the simulator can find polynomials $q_1(\mathbf{t}), q_2(\mathbf{t}), \ldots, q_n(\mathbf{t})$ such that

$$f(\mathbf{t}) - f(\mathbf{a}^*) = \sum_{i\in[n]} (t_i - a_i^*) q_i(\mathbf{t}) .$$

Therefore, we can re-write the above Equation C.13 as below:

$$\mathsf{e}\,(g,g)^c = \prod_{i\in[n]} \mathsf{e}\left(g^{t_i-a_i^*}, g^{q_i(\mathbf{t})}\right) \cdot \prod_{i\in[n]} \mathsf{e}\left(g^{a_i^*-t_i}, w_i\right) . \tag{C.14}$$

Or equivalently,

$$\mathsf{e}(g,g)^c = \prod_{i\in[n]} \mathsf{e}\left(g^{t_i-a_i^*}, \quad g^{q_i(\mathbf{t})} \cdot w_i\right) . \tag{C.15}$$

Now, the simulator will try to raise both sides of the above Equation C.15 to $\frac{1}{\tau-a_1^*}$, i.e., divide the exponent by $\tau - a_1^*$. If the simulator can successfully do this for the right-hand side, then clearly, the simulator would be able to obtain the value $\mathsf{e}(g,g)^{\frac{c}{\tau-a_1^*}}$—thereby breaking the ℓ-SBDH assumption.

The problem is that it is not possible to directly raise the right-hand side of Equation C.15 to $\frac{1}{\tau-a_1^*}$. Fortunately, recall that the simulator has carefully crafted the t_i values earlier (implicitly without actually learning the t_i values). Specifically, due to Equations C.10, C.11, and C.12, it is not hard to see that

$$t_i = a_i^* = r_i(\tau - a_1^*) .$$

As a result,

$$\left(\prod_{i\in[n]} \mathsf{e}\left(g^{t_i-a_i^*}, \quad g^{q_i(\mathbf{t})} \cdot w_i\right)\right)^{\frac{1}{\tau-a_1^*}} = \prod_{i\in[n]} \mathsf{e}\left(g^{r_i}, \quad g^{q_i(\mathbf{t})} \cdot w_i\right) .$$

It is not hard to see that the right-hand side of the above equation provides an efficient method for the simulator to raise the right-hand side of Equation C.15 to $\frac{1}{\tau-a_1^*}$. Specifically, the simulator can now compute

$$e\,(g,g)^{\frac{1}{\tau-a_1^*}} = \left(\prod_{i\in[n]} \mathsf{e}\left(g^{r_i}, \quad g^{q_i(\mathbf{t})} \cdot w_i\right)\right)^{c^{-1}} ,$$

breaking in this way the ℓ-SBDH assumption. This completes the proof.

C.2 Construction for derivative evaluation

We now build a simulator $\mathcal{S}$ which obtains an instance of the ℓ-SBDH assumption from a challenger $\mathcal{C}$. The simulator $\mathcal{S}$ then embeds this ℓ-SBDH assumption into an instance of the verifiable computation scheme for the derivative evaluation—such that if an adversary $\mathcal{A}$ can break the selective security of the verifiable computation scheme with more than negligible probability, the simulator $\mathcal{S}$ can then leverage the adversary $\mathcal{A}$ to break the ℓ-SBDH instance it obtained from from the challenger, also with non-negligible probability.

C.2.1 Proof of Part (3) of Theorem 2

Assume that the simulator $\mathcal{S}$ obtains the following $(k+1)d$-SBDH instance from the challenger $\mathcal{C}$,

$$\left(g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^{(k+1)d}}\right) ,$$

where $0 \leq k \leq d$ is the highest order derivative one needs to support, and d is an upper bound on the degree of the polynomial.

Initialization. The challenger runs the key generation algorithm, and gives the public key PK to the adversary $\mathcal{A}$. The adversary then commits to a challenge point

$$\mathbf{a}^* = [a_1^*, a_2^*, \ldots, a_n^*] .$$

Setup. When the adversary first queries the Setup algorithm, the simulator $\mathcal{S}$ performs the following. The simulator needs to choose a random point $\mathbf{t}$, evaluate the polynomial at $\mathbf{t}$, and create the corresponding witness generation set $\mathcal{W}_{n,d}$ and verification information $\mathsf{VI}(f)$ the chosen point $\mathbf{t}$. The simulator $\mathcal{S}$ will implicitly choose the point $\mathbf{t}$ as below. The simulator guesses at random an index $j \in [n]$, and an order $0 \leq k \leq d$—such that in the challenge stage, the adversary will output a forgery for the k-th derivative

$$\frac{\partial^k f(\mathbf{x})}{\partial x_j^k}$$

at point $\mathbf{a}^*$. If this guess turns out to be wrong later, the simulation simply aborts. Notice that the simulator can guess right with probability $\frac{1}{nd}$.

For the chosen coordinate j, the simulator implicitly lets

$$t_j = \tau , \tag{C.16}$$

without actually computing t_j.

For $i \neq j$, the simulator picks random $r_i \in \mathbb{Z}_p$, and implicitly chooses

$$t_i = r_i(\tau - a_j^*)^{k+1} + a_i^* , \tag{C.17}$$

without actually computing the values. The simulator remembers the r_i values for later use.

Now the simulator needs to compute the witness generation set $\mathcal{W}_{n,d}$ and $\mathsf{VI}(f)$. The problem is that the simulator does not actually know the t_i's, since these values are inherited and transformed from the $(k+1)d$-SBDH instance it obtained from the challenger. Fortunately, observe that the challenger can still compute all terms in $\mathcal{W}_{n,d}$ and $\mathsf{VI}(f)$, since when one plugs in Equations C.16 and C.17, it is not hard to see that all terms in WK and VI are essentially of the form $g^{q(\tau)}$ where $q(\tau)$ is some polynomial of degree at most $(k+1)d$.

Since the challenger knows the values of

$$g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^{(k+1)d}}$$

from the $(k+1)d$-SBDH instance, it can easily compute all of these terms. Notice that in the above simulation the terms generated by the simulator are identically distributed as as running the real Setup algorithm.

Updates. The adversary can request update the polynomial either by calling the Setup algorithm again, or by calling the incremental update algorithm Update. As mentioned in Section 3, updates change the verification information $\mathsf{VI}(f)$. It is not hard to see that the simulator does not need to know the t_i's to update the digest in the $\mathsf{VI}(f)$. It can then use the signing key $\Sigma.\mathsf{sk}$ to sign the updated digest.

Challenge. The adversary outputs a forgery for the committed point $\mathbf{a}^*$, evaluating the derivative $\partial^k f(\mathbf{x})/\partial x_j^k$. If the values of k and j turn out to be different that what the simulator guessed, simply abort the simulation. As mentioned earlier, the simulator can guess correctly with probability $\frac{1}{nd}$.

The forgery consists of some $\mathsf{VI}(f)$, a claimed derivative v for $\partial^k f(\mathbf{x})/\partial x_j^k(\mathbf{a}^*)$, and a witness parsed as:

$$w = (w_2, w_3, \ldots, w_n, \omega, c_{k-1}, c_{k-2} \ldots, c_0)\,.$$

Due to the security of the signature scheme, $\mathsf{VI}(f)$ must be an output of one of the oracle queries to either Setup or Update—since otherwise, one can leverage this adversary and build a straightforward reduction to break the security of the signature scheme. Let f denote the corresponding input of that oracle query which resulted in $\mathsf{VI}(f)$. If the forgery is successful, the following must be true:

$$v \neq \frac{\partial^k f(\mathbf{x})}{\partial x_j^k}(\mathbf{a}^*) \quad \text{and} \quad \mathsf{Verify}(\mathsf{VI}, \mathbf{x}, v, w, k, i) = 1\,.$$

The simulator will now leverage this forgery to break the $(k+1)d$-SBDH instance it got from the challenger. Since the verification succeeds, the following holds:

$$\mathsf{e}\left(g^{f(\mathbf{t})}, g\right) = \prod_{i \neq j} \mathsf{e}\left(g^{t_i} g^{-a_i^*}, w_i\right) \cdot \mathsf{e}\left(g^{(t_j - a_j^*)^{k+1}}, \omega\right) \cdot \prod_{i=0}^{k} \mathsf{e}\left(g^{t_j^i}, g\right)^{c_i}, \tag{C.18}$$

where $c_k = \frac{v}{k!}$. The simulator now decomposes $f(\mathbf{t})$ as in Lemma 3.

$$f(\mathbf{t}) = \sum_{i \neq j} (t_i - a_i^*)\hat{u}_i(\mathbf{t}) + (x_j - a_j^*)^{k+1}\hat{q}(x_j) + \hat{c}_k x_j^k + \hat{c}_{k-1} x_j^{k-1} + \ldots + \hat{c}_1 x_j + \hat{c}_0\,.$$

Notice that we use the convention that the hatted values correspond to the correct decomposition of the polynomial which is performed by the simulator. The unhatted versions of the same variables are those returned by the adversary. They may not be from the correct the decomposition, however, the verification equation (Equation C.18) still holds. Rewrite Equation C.18 as:

$$\prod_{i=0}^{k} \mathsf{e}\left(g^{t_j^i}, g\right)^{\hat{c}_i - c_i} = \mathsf{e}\left(g^{(t_j - a_j^*)^{k+1}}, \omega g^{-\hat{q}(\mathbf{t})}\right) \prod_{i \neq j} \mathsf{e}\left(g^{t_i - a_i^*}, w_i g^{-\hat{u}(\mathbf{t})}\right)\,. \tag{C.19}$$

Due to Equation C.17, for $i \neq j$, we have

$$t_i - a_i^* = r_i \left(t_j - a_j^*\right)^{k+1} = r_i \left(\tau - a_j^*\right)^{k+1}\,.$$

The simulator can raise the right-hand side of Equation C.19 to $\frac{1}{(\tau - a_j^*)^{k+1}}$, by computing the following:

$$\mathsf{e}\left(g, \omega g^{-\hat{q}(\mathbf{t})}\right) \prod_{i \neq j} \mathsf{e}\left(g^{r_i}, w_i g^{-\hat{u}(\mathbf{t})}\right)\,.$$

Notice that the simulator is able to compute the values $g^{\hat{q}(\mathbf{t})}$ and $g^{\hat{u}(\mathbf{t})}$ (evaluated at $\mathbf{t}$) simply by using terms contained in $\mathcal{W}_{n,d}$, even though the simulator does not know the value of $\mathbf{t}$ in the clear.

Let $\Delta_i = \hat{c}_i - c_i$. The simulator now has the following:

$$\mathsf{e}\left(g, g\right)^{\frac{\sum_{i=0}^{k} \Delta_i \tau^i}{(\tau - a_j^*)^{k+1}}} = \mathsf{e}\left(g, \omega g^{-\hat{q}(\mathbf{t})}\right) \prod_{i \neq j} \mathsf{e}\left(g^{r_i}, w_i g^{-\hat{u}(\mathbf{t})}\right)\,. \tag{C.20}$$

We now prove the following lemma.

Lemma 4 *Given* $\mathsf{e}(g,g)^{\frac{\sum_{i=0}^{k}\Delta_i\tau^i}{(\tau-a_j^*)^{k+1}}}$ *from Equation C.20, the simulator can break the* $(k+1)d$*-SBDH assumption.*

Suppose, without loss of generality that $\sum_{i=0}^{k}\Delta_i\tau^i$ does not divide $(\tau-a_j^*)^{k+1}$. If not, one can cancel out the $(\tau-a_j^*)$ factors appearing in polynomial $\sum_{i=0}^{k}\Delta_i\tau^i$, which yields prime polynomials. Then, by using the extended Euclidean algorithm, the simulator can compute polynomials $g(\tau)$ and $f(\tau)$ such that

$$g(\tau)\sum_{i=0}^{k}\Delta_i\tau^i + f(\tau)(\tau-a_j^*)^{k+1} = 1 \Rightarrow g(\tau)\sum_{i=0}^{k}\Delta_i\tau^i = 1 - f(\tau)(\tau-a_j^*)^{k+1}\,.$$

Therefore Equation C.20 is equivalent to

$$\mathsf{e}(g,g)^{\frac{1-f(\tau)(\tau-a_j^*)^{k+1}}{(\tau-a_j^*)^{k+1}}} = \mathsf{e}\left(g^{g(\tau)}, \omega g^{-\hat{q}(\mathbf{t})}\right)\prod_{i\neq j}\mathsf{e}\left(g^{r_i g(\tau)}, w_i g^{-\hat{u}(\mathbf{t})}\right)\,,$$

yielding

$$\mathsf{e}(g,g)^{\frac{1}{(\tau-a_j^*)^{k+1}}} = \mathsf{e}(g,g)^{f(\tau)}\,\mathsf{e}\left(g^{g(\tau)}, \omega g^{-\hat{q}(\mathbf{t})}\right)\prod_{i\neq j}\mathsf{e}(g^{r_i g(\tau)}, w_i g^{-\hat{u}(\mathbf{t})})\,,$$

which eventually gives

$$\mathsf{e}(g,g)^{\frac{1}{\tau-a_j^*}} = \mathsf{e}(g,g)^{f(\tau)(\tau-a_j^*)^k}\,\mathsf{e}\left(g^{g(\tau)(\tau-a_j^*)^k}, \omega g^{-\hat{q}(\mathbf{t})}\right)\prod_{i\neq j}\mathsf{e}\left(g^{r_i g(\tau)(\tau-a_j^*)^k}, w_i g^{-\hat{u}(\mathbf{t})}\right)\,.$$

In other words, unless the adversary honestly follows the protocol, the simulator will be able to break the $(k+1)d$-SBDH assumption.

C.3 Full Security for Univariate Polynomials

For the special case of univariate polynomials, our constructions (both for polynomial evaluation and for derivative evaluation) satisfy the full security definition.

C.3.1 Proof of Part (2) of Theorems 1 and 2

It is not hard to modify the proofs in Sections C.1 and C.2 to the univariate case, and achieve full security. Since there is only a single variable in this case, the simulator can simply embed the τ obtained from the challenger into the only variable by Equation C.10 or Equation C.16. Since the other variables disappear from the proof, Equations C.12 and C.17 are no longer needed. As a result, there is no longer a need for the adversary to commit to the challenge point ahead of time. We can therefore prove full security for the special case of univariate polynomials.

D Discussion

Note on freshness. If the verification information $\mathsf{VI}(f)$ is stored at an untrusted server, the client needs to download the necessary verification information (or a subset of the terms) during the verification step. A malicious server can cheat by providing an old version of $\mathsf{VI}(f)$, thereby violating freshness guarantees.

Freshness is outside the scope of this paper, and essentially the same problem has been studied in classic settings, such as in authenticated data structures [34], or the distribution of certificates in PKI [36]. We suggest one potential method to achieve freshness below. Basically, the trusted source can build a Merkle hash tree over all terms in the $\mathsf{VI}(f)$, sign the root hash, and publish the signed root hash. The trusted source must have some channel to distribute the up-to-date root hash to all clients, e.g., publish it on a website. For higher resilience, the trusted source can also distribute the up-to-date root hash to multiple parties, such that one can use a majority vote mechanism to determine the most up-to-date hash. If clients and the trusted source have weak clock synchronization, the trusted source can also update the root hash each day, and attach a timestamp during signing. This way, a client can be sure that it has the latest root hash by checking the timestamp.

Once a client can obtain the most up-to-date root hash, it can download a subset or all of $\mathsf{VI}(f)$, and the using standard authenticated Merkle hash tree techniques, it can verify that the $\mathsf{VI}(f)$ (or a subset of it) is the most up-to-date.