Signatures of Correct Computation

Charalampos Papamanthou
UC Berkeley
cpap@cs.berkeley.edu

Elaine Shi
University of Maryland
elaine@cs.umd.edu

Roberto Tamassia
Brown University
rt@cs.brown.edu

January 22, 2013

Abstract

We introduce *Signatures of Correct Computation* (SCC), a new model for verifying dynamic computations in cloud settings. In the SCC model, a trusted *source* outsources a function f to an untrusted *server*, along with a public key for that function (to be used during verification). The server can then produce a succinct signature σ vouching for the correctness of the computation of f, i.e., that some result v is indeed the correct outcome of the function f evaluated on some point $\mathbf{a}$. There are two crucial performance properties that we want to guarantee in an SCC construction: (1) verifying the signature should take asymptotically less time than evaluating the function f; and (2) the public key should be efficiently updated whenever the function changes.

We construct SCC schemes (satisfying the above two properties) supporting expressive manipulations over multivariate polynomials, such as polynomial evaluation and differentiation. Our constructions are adaptively secure in the random oracle model and achieve *optimal* updates, i.e., the function's public key can be updated in time proportional to the number of updated coefficients, without performing a linear-time computation (in the size of the polynomial).

We also show that signatures of correct computation imply *Publicly Verifiable Computation* (PVC), a model recently introduced in several concurrent and independent works. Roughly speaking, in the SCC model, *any client* can verify the signature σ and be convinced of some computation result, whereas in the PVC model only the client that issued a query (or anyone who trusts this client) can verify that the server returned a valid signature (proof) for the answer to the query. Our techniques can be readily adapted to construct PVC schemes with adaptive security, efficient updates and *without the random oracle model*.

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. Consider for example the following scenario: A company has developed some novel algorithm, e.g., for personalized medicine, or for stock trend prediction. To avoid investing in expensive IT infrastructure in-house, the company chooses to outsource the execution of this algorithm to an external, untrusted cloud provider (e.g., Amazon, Google). How could a user verify the correctness of the computation under the assumption that she *only* trusts the company that developed the algorithm, but not the cloud provider? The above question poses two crucial requirements: (1) *efficiency*, meaning that the running time of the verification algorithms executed by the client should be asymptotically less than the time needed to execute the algorithm in the cloud; and (2) *public verifiability*, meaning that our verification mechanism should not be tied to a specific verifier's secret key so that any user can verify the computation. In addition, another desirable property is to *efficiently handle updates* to the outsourced algorithm, without computing public parameters from scratch.

In this paper, we propose a new paradigm for verifying dynamic computation in the cloud called *signatures of correct computation* (*SCC*). SCC allows an untrusted worker to produce a signature vouching for the correctness of some computation over some input; any user can verify the signature using a public key (produced by an one-time preprocessing) published by a trusted source who outsourced the function in the cloud.

Signatures of correct computation are closely related to publicly verifiable computation (PVC), proposed by Parno *et al*. [34], Canetti *et al*. [9] and Fiore and Gennaro [13, 14], in *concurrent and independent* works to ours. Specifically, *signatures of correct computation are stronger than publicly verifiable computation: given an SCC scheme, one can directly construct a PVC scheme*; while the other way around does not seem to be true. More specifically, in PVC, a "proof of correct computation" is tied to a specific challenge (generated by an algorithm ProbGen in [34]), and can only be verified by the client who has generated that challenge (or anyone who trusts this client). By contrast, a signature of correct computation is not tied to any challenge, and can be verified by anyone in the world, in much the same way as a traditional signature on a message. We provide a detailed comparison of PVC and SCC in Section 1.2.

1.1 Results and contributions

We design SCC schemes for multivariate polynomial manipulations, including polynomial evaluation and differentiation. One of our technical highlights is a new method in this setting that allows us to *slightly modify our selectively secure schemes to achieve adaptive security*. Our SCC schemes achieve adaptive security under the random oracle model. We also show that under the weaker PVC model, our techniques can achieve adaptive security under the standard model without random oracles.

Our main results and contributions are summarized below:

Definition of new paradigm. We are the first ones to formally define signatures of correct computation (SCC) and its security and to study its relation to PVC.

Novel constructions for polynomial manipulations. We focus on deriving efficient and optimized constructions for *specific* functionalities rather than *generic* constructions, as the approach taken by Parno *et al*. [34] and Canetti *et al*. [9]. We present efficient SCC constructions for expressive polynomial manipulations, including multivariate polynomial evaluation and differentiation. Operations on polynomials represent a common building block in a wide range of applications, such as in statistical analysis, scientific computing, and machine learning. Fiore and Gennaro [14] point out many interesting applications of publicly verifiable computation on polynomials, including its use in proofs of retrievability, verifiable keyword search, discrete Fourier tranform, and linear transformations. Our constructions are based on bilinear groups. We prove the adaptive security of our constructions under the random oracle model.

Efficient incremental updates. Our constructions allow a trusted source to make *incremental updates* in time proportional to the number of the updated polynomial coefficients, and without performing a computation from scratch that would take linear time in the size of the polynomial.

Novel proof techniques for adaptive security. Our constructions and proofs introduce several novel techniques. First, we observe key polynomial decomposition properties (Lemmas 1 and 3) that become the central idea underlying our constructions. Second, while achieving adaptive security appears relatively easy for univariate polynomial evaluation [25], achieving adaptive security in the multivariate case appears to be fundamentally more difficult. To this end, we present novel techniques that involve embedding randomness in the polynomial decomposition properties (Lemmas 2 and 4), such that our simulator can later manipulate these random numbers in the proof. We give a high-level technical overview in Section 1.3.

Contributions to publicly verifiable computation. Our results also bring advances in the area of publicly verifiable computation. Specifically, our techniques can be readily applied to yield publicly verifiable computation schemes (for the same operations) with adaptive security (without the random oracle model) and with efficient updates. In comparison, existing PVC works [9, 14, 34], achieve adaptive security but do not support efficient updates. We give a more detailed comparison in Section 1.2.

Table 1: **Asymptotic cost on the client side.** In the table below, n is the number of variables in the polynomial and d is the maximum degree. With **SVC** we denote a "secretly delegatable and verifiable scheme", with **PVC** we denote a "publicly delegatable and verifiable scheme", with **PVC*** we denote a "publicly verifiable but not publicly delegatable scheme" (see Section 1.2, Paragraph 5) and with **SCC** we denote a "signatures of correct computation scheme". Notice that an n-variate polynomial of degree d can have up to $\binom{n+d}{d}$ terms, requiring up to $\binom{n+d}{d}$ time to evaluate. Therefore, the verification costs here are smaller than the cost of evaluating the polynomial. For PVC schemes, the client cost includes both delegation and verification costs.

scheme	**polynomial evaluation**	**polynomial differentiation**	**efficient updates**	**security**	**model**
Benabbas *et al.* [3]	$n \log d$	N/A	no	adaptive	**SVC**
Parno *et al.* [34]	n	$n + \log d$	no	adaptive	**PVC**
Canetti *et al.* [9]	$\text{polylog}\left(\binom{n+d}{d}\right)$	$\text{polylog}\left(\binom{n+d}{d}\right)$	no	adaptive	**PVC**
Fiore and Gennaro [13, 14]	$n \log d$	N/A	no	adaptive	**PVC***
This paper	n	$n + d$	yes	selective	**SCC**
This paper	$n + d$	$n + d^2$	yes	adaptive	**PVC**
This paper	$n + d$	$n + d^2$	yes	adaptive (RO)	**SCC**

1.2 Related work

Authenticated data structures. The SCC model is directly related to the model of *authenticated data structures* (ADS) [36, 38]. In some sense, SCC and ADS are dual problems to each other, sharing exactly the same security properties. In SCC, a trusted source outsources a function, and a client wishes to verify the outcome of the function at a given point. In ADS, a trusted source outsources the data or a data structure, and the client wishes to verify the correctness of the result of a data structure query, e.g., dictionaries [19, 28], graphs [21, 27] and hash tables [32, 37]. Most authenticated data structures schemes incur logarithmic or linear overheads for verification costs, with some exceptions being authenticated range queries [2, 20] and set operations [33], where verification takes time proportional to the size of the answer.

Verifiable computation in the secret key setting (SVC). Recent works on verifiable computation [1, 10, 15] achieve efficient verification of general boolean circuits, but in the secret key model. Therefore they are inherently inadequate for the setting of signatures, which are required to be publicly verifiable.

Verifiable computation for polynomials. Benabbas *et al.* [3] developed methods for efficient verification of multivariate polynomial evaluation by using algebraic one-way functions—however, in the SVC model. This work does not achieve efficient updates of polynomial coefficients (specifically, in order to update a coefficient, one has to re-randomize all the existing coefficients).[1] Kate *et al.* [25] give a publicly verifiable commitment scheme for univariate polynomials, which is essentially an SCC scheme for univariate polynomial evaluation. However, their scheme does not directly extend to multivariate polynomials. Also note that our construction is the first to support efficient verification of differentiation queries—even in the SVC setting.

Relation to CS proofs and SNARGs. Our SCC model is strongly related to the model of *computationally-sound proofs*, introduced by Micali in 1994 [29], and to the subsequent works on *succinct non-interactive arguments* (SNARGs) by Groth [23], Bitansky *et al.* [4, 5] and Gennaro *et al.* [16]. The main connection

[1] However, apart from verification of polynomial evaluation, their techniques can be applied to support very efficient dynamic verifiable databases (constant query and update complexity).

is that both SCC and SNARGs models are non-interactive and publicly verifiable (CS proofs can also be non-interactive in the random oracle model), i.e., a publicly verifiable proof can be computed independently from (and with no communication with) the verifier. We note here that all CS proofs and SNARGs constructions that have been presented in the literature are *generalized*, in that they can handle all of NP by using powerful tools such as the PCP theorem (with an exception of [16] that uses a different characterization of NP). Moreover, all of them (except for the work of Micali [29] that is secure in the random oracle model) are proved secure based on non-falsifiable assumptions [18], e.g., the works of Groth [23] and Gennaro *et al*. [16] use variants of the knowledge-based assumption introduced by Damgard [12]. Non-falsifiable assumptions are considered to be a lot stronger than all common assumptions used in cryptography (one-way functions, trapdoor permutations, DDH, RSA, LWE etc.). We note that the assumptions that we are using in our construction *do not belong* in this category—however, for verifying multivariate polynomials (not for univariate ones) we do use the random oracle, as the construction of Micali [29] does. The main difference (with [29]) however is that we do not use the PCP theorem, hence achieving more practical schemes.

Concurrent and independent works. Two closely related schemes are the ones by Parno *et al*. [34] and Cannetti *et al*. [9], which were developed concurrently with and independently from our work.

In the PVC formulation proposed by Parno *et al*. [34], any client can verify that an untrusted server correctly computes a function f on a specific input $\mathbf{a}$. Their definition however requires an *input preparation* randomized algorithm ($\mathsf{ProbGen}$), mapping user inputs $\mathbf{a}$ to server inputs $\sigma_{\mathbf{a}}$ and preparing an object $\mathsf{VK}_{\mathbf{a}}$ to be used for verification, specific for $\sigma_{\mathbf{a}}$. Therefore, as opposed to the SCC setting, only the client that issued a query for $\mathbf{a}$ (or anyone who trusts this client) can verify that the server returned a valid signature (proof) for $f(\mathbf{a})$. For otherwise, a client running the $\mathsf{ProbGen}$ algorithm can potentially collude with the server to forge a proof, convincing another party to accept the proof. Apart from defining PVC, Parno *et al*. [34] give a construction for generalized boolean functions (closed under complement) from attribute-based encryption (ABE). Their construction is asymptotically efficient—the proof size is proportional to the size of the answer. Moreover, due to recent advances in ABE schemes by Lewko and Waters [26], the PVC constructions of Parno *et al*. [34] can be proved adaptively secure, since they directly inherit the security of the underlying ABE scheme.

A PVC scheme having similar properties with the scheme of Parno *et al*. [34] was presented by Canetti *et al*. [9], where client verification is polylogarithmic in the size of the evaluated circuit. Canetti *et al.* achieve adaptive security under a slightly weaker model (as Parno *et al.* point out [34]), in which the client needs to keep certain secret state. Their scheme shares the same limitation with the scheme of Parno *et al*. [34] in that a client can verify only his queries unless extra assumptions are put into place.

The most closely related works are the recent works by Fiore and Gennaro [14], who presented a PVC scheme tailored for multivariate polynomials that is based on algebraic one-way functions. An improved version [13] uses less complex assumptions such as RSA to achieve the same goal. The works by Fiore and Gennaro differ from ours in the following sense. First, they consider a model (denoted with PVC* in Table 1) that is more *restrictive* than the PVC model proposed by Parno *et al*. [34]—and hence more restrictive than the SCC model. Specifically, there is an explicit delegation phase where a problem instance is generated based on an input (as in the PVC definition by Parno *et al*. [34]). However, in their constructions (and unlike the original PVC definition), only the party who ran the setup algorithm for a specific function can run the problem generation algorithm. Therefore, their schemes are *publicly verifiable, but not publicly delegatable*. As a result, their schemes would not work for the application scenario where a pharmaceutical company outsources a genomic algorithm, and each user submits their own genomic data for computation. Moreover, they do not consider efficient updates of the polynomial coefficients. In comparison, their scheme has more efficient verification and a delegation step of $O(n \log d)$ cost. A detailed comparison of our scheme against several related works in terms of verification cost and security model is presented in Table 1.

1.3 Highlights of techniques

Multivariate polynomial evaluation. The polynomial commitment scheme by Kate *et al.* [25] can be employed to construct an SCC scheme of univariate polynomial evaluations. Specifically, Kate *et al.* [25] 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 above test does not apply to the multivariate case. We therefore propose a novel technique based on the following 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 $f(\mathbf{x}) - f(\mathbf{a}) = \sum_{i=1}^{n}(x_i - a_i)w_i(\mathbf{x})$. The polynomials $w_i(\mathbf{x})$ will be used to construct witnesses in our scheme. Specifically, we encode their terms as exponents of bilinear group elements. The verification is a pairing product equation encoding the above test in the exponent.

From selective to adaptive security. The test that holds for the polynomial evaluation contains a sum of terms, as opposed to a single term in the univariate case [25]. This gives rise to certain technicalities in the proof, allowing us to prove only the weaker notion of *selective security* (see Definition 6 in Section A of the Appendix).

Going from selective security to adaptive security turns out to be non-trivial. To achieve this, we devise a novel technique where we build randomness into the polynomial decompositions (Lemmas 2 and 4) which are central to our constructions. As an immediate corollary of our adaptively secure SCC construction with random oracles, we construct an adaptively secure PVC scheme in the plain model.

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 5, this 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 $f(x)$, 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 5.

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 λ, PPT stands for *probabilistic 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 $x_1, x_2, \ldots, x_n$.

2.1 Problem definition

We now formally define signatures of correct computation (SCC).

Definition 1 (SCC scheme) *An* SCC scheme *(signatures of correct computation) for a function family $\mathcal{F}$ is a tuple* $(\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Compute}, \mathsf{Verify}, \mathsf{Update})$ *of five PPT algorithms with the following specification:*

1. $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(\lambda, \mathcal{F})$*: Algorithm* KeyGen *takes as input the security parameter λ and a function family $\mathcal{F}$. It outputs a public/secret key pair* $(\mathsf{PK}, \mathsf{SK})$. KeyGen *is run only once at system initialization by a trusted source;*

2. $\mathsf{FK}(f) \leftarrow \mathsf{Setup}(\mathsf{SK}, \mathsf{PK}, f)$*: Algorithm* Setup *(run by a trusted source) takes as input the secret key* SK*, the public key* PK*, and a function $f \in \mathcal{F}$. It outputs the function public key* $\mathsf{FK}(f)$ *for the function f;*
3. $(v, w) \leftarrow \mathsf{Compute}(\mathsf{PK}, f, \mathbf{a})$*: Algorithm* Compute *(run by an untrusted server) takes as input the public key* PK*, a function $f \in \mathcal{F}$ and a value $\mathbf{a} \in \mathsf{domain}(f)$. It outputs a pair (v, w), where $v = f(\mathbf{a})$ and w is a signature;*
4. $\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f), \mathbf{a}, v, w)$*: Algorithm* Verify *(run by any verifier) takes as input the public key* PK*, the function public key* $\mathsf{FK}(f)$*, value $\mathbf{a} \in \mathsf{domain}(f)$, a claimed result v and a signature w. It outputs* 0 *or* 1*;*
5. $\mathsf{FK}(f') \leftarrow \mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{FK}(f), f')$*: Algorithm* Update *(run by a trusted source) takes as input the secret key* SK*, the public key* PK*, the function public key* $\mathsf{FK}(f)$ *for the old function f and the updated function description f'. It outputs the updated function public key* $\mathsf{FK}(f')$*.*

The Update algorithm allows the source to update the function f to a new function f'. A naive way to implement Update is to simply run the Setup algorithm again for the new f'. However, in practice, one may wish to allow more efficient incremental updates (and this is what is achieved by our constructions).

2.2 Correctness and security definitions

We describe now the correctness and adaptive security definitions for SCC. Intuitively, an SCC scheme is correct if whenever its algorithms are executed honestly, it never rejects a correct signature. Also, it is secure if, after the setup/update algorithms have been executed, an adversary cannot convince a verifier to accept a wrong result on an input of his choice, except with negligible probability.

Definition 2 (Correctness of an SCC scheme) *Let λ be the security parameter and let $\mathcal{P}$ be an SCC scheme* (KeyGen, Setup, Compute, Verify, Update) *for a function family $\mathcal{F}$. 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* $\mathsf{FK}(f_i)$ *is the output of* $\mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{FK}(f_{i-1}), f_i)$*, where* $\mathsf{FK}(f_0)$ *is output by algorithm* $\mathsf{Setup}(\mathsf{SK}, \mathsf{PK}, f_0)$ *for some $f_0 \in \mathcal{F}$. We say that $\mathcal{P}$ is* correct*, if for any $i = 0, \ldots, \mathsf{poly}(\lambda)$, for any $\mathbf{a} \in \mathsf{domain}(f_i)$, it is $1 \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f_i), \mathbf{a}, v, w)$, where $(v, w) \leftarrow \mathsf{Compute}(\mathsf{PK}, f_i, \mathbf{a})$.*

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

1. ***Initialization.*** *The challenger runs algorithm* KeyGen *which outputs* (PK, SK) *and then gives* PK *to the adversary but maintains* SK *secret;*
2. ***Setup and update.*** *The adversary 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* $\mathsf{FK}(f_0)$*. Then, for $i = 1, \ldots, k$, where $k = \mathsf{poly}(\lambda)$, he makes a polynomial number of oracle queries to the* $\mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{FK}(f_{i-1}), f_i)$ *algorithm, each time specifying $f_i \in \mathcal{F}$. The challenger answers the queries by returning the resulting* $\mathsf{FK}(f_i)$*;*
3. ***Forgery.*** *The adversary $\mathcal{A}$ outputs a point $\mathbf{b} \in \mathsf{domain}(f_i)$ for some $0 \leq i \leq k$, and the forgery $(\mathbf{b}, v, w)$.*

The adversary $\mathcal{A}$ wins the game if $1 \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f_i), \mathbf{b}, v, w)$ and $f_i(\mathbf{b}) \neq v$.

2.3 SCC implies PVC

As we highlighted in the introduction, signatures of correct computation (SCC) are stronger than the publicly verifiable computation (PVC) notions studied in concurrent but independent papers [9, 13, 14, 34]. Specifically, a correct and secure SCC scheme implies a correct and secure PVC scheme, but not the other way around. To see that, one can implement algorithm $\sigma_{\mathbf{a}} \leftarrow \mathsf{ProbGen}(\mathsf{PK}, \mathbf{a})$ of the PVC scheme (e.g., [34]) to simply output $\mathbf{a}$ and all the other algorithms remain the same.

For completeness, in Definition 7 in Section A of the Appendix, we also provide the definition of publicly verifiable computation (PVC) along with its correctness (Definition 8) and adaptive security (Definition 9) definitions. Our PVC definition is essentially equivalent to those proposed by Parno *et al*. [34] and Canetti *et al*. [9], with the exception that we augment it with an Update algorithm which a trusted source can employ to incrementally update the outsourced function (also, our ProbGen algorithm is called Challenge).

2.4 Multivariate polynomials notation

We now give some notation for multivariate polynomials. We use the notion of a *multiset* over some universe $\mathcal{U}$, 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 $x \notin S$, $S(x) = 0$. E.g., 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 universe $\mathcal{U}$. It is $S \subseteq T$, if $\forall a \in \mathcal{U}$, $S(a) \leq T(a)$. The *size* of S over universe $\mathcal{U}$, denoted $|S|$, is defined as the sum of the multiplicity of all elements in S, i.e., $|S| = \sum_{a \in \mathcal{U}} S(a)$. Finally, $\mathcal{S}_{d,n}$ denotes the set of multisets of size at most d over the universe $\{1, 2, \ldots, n\}$. Let now $f \in \mathbb{Z}_p[x_1, x_2, \ldots, x_n] = \mathbb{Z}_p[\mathbf{x}]$ 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.,

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

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. Finally, 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.

2.5 Bilinear groups and computational assumption

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. We use the following computational assumption [6]:

Definition 4 (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 $\mathbb{Z}_p^*$, for $\ell = \mathsf{poly}(\lambda)$, there is no PPT 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 Selectively secure multivariate polynomial evaluation

As a warm-up exercise, in this section we first present an SCC scheme for multivariate polynomial evaluation that is secure under a relaxed security model, namely, the *selective* security model. Then, in Section 4, we explain how to augment this selectively secure scheme and achieve adaptive security in the random oracle model.

Selective security is weaker than adaptive security, requiring the adversary to *commit ahead of time* to the challenge point $\mathbf{a}$, which is analogous to the selective security notion often adopted in Identity-Based Encryption (IBE) [7], Attribute-Based Encryption (ABE) [22], Functional Encryption (FE) [35] and Predicate Encryption (PE) [8]. The detailed selective security definition is described in Definition 6 in the Appendix.

3.1 Intuition

Our construction relies on the following key observation.

Lemma 1 (Polynomial decomposition) *Let $f(\mathbf{x}) \in \mathbb{Z}_p[\mathbf{x}]$ be an n-variate polynomial. For all $\mathbf{a} \in \mathbb{Z}_p^n$, there exist polynomials $q_i(\mathbf{x}) \in \mathbb{Z}_p[\mathbf{x}]$ such that the polynomial $f(\mathbf{x}) - f(\mathbf{a})$ can be expressed as $f(\mathbf{x}) - f(\mathbf{a}) = \sum_{i=1}^{n}(x_i - a_i)q_i(\mathbf{x})$. Moreover, there exists a polynomial-time algorithm to find the above polynomials $q_i(\mathbf{x})$.*

Proof: The direct of the claim is straightforward: If a polynomial $f(\mathbf{x}) - f(\mathbf{a})$ 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 $f(\mathbf{x}) - f(\mathbf{a}) \in \mathbb{Z}_p[\mathbf{x}]$, we use polynomial division to first divide $f(\mathbf{x}) - f(\mathbf{a})$ by $(x_1 - a_1)$. Specifically,

$$f(x_1, x_2, \ldots, x_n) = (x_1 - a_1) \cdot q_1(x_1, x_2, \ldots, x_n) + s_1(x_2, x_3, \ldots, x_n)\,,$$

where $s_1(x_2, x_3, \ldots, x_n)$ is the remainder term, and $s_1(x_2, x_3, \ldots, x_n)$ should no longer contain the variable x_1. Next, divide $s_1(x_2, x_3, \ldots, x_n)$ by $(x_2 - a_2)$, and divide the remainder by $(x_3 - a_3)$, and so on. In this way, we can write $f(\mathbf{x}) - f(\mathbf{a})$ as

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

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

Given now an n-variate polynomial $f(\mathbf{x})$, the trusted source runs algorithms KeyGen and Setup to create the function public key $\mathsf{FK}(f) = g^{f(\mathbf{t})}$ of the polynomial f evaluated over a randomly chosen point $\mathbf{t}$. 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 Lemma 1: It will compute n polynomials $q_1(\mathbf{x}), q_2(\mathbf{x}), \ldots, q_n(\mathbf{x})$ such that the relation of Lemma 1 holds, and the values $g^{q_i(\mathbf{t})}$ $(i = 1, \ldots, n)$ will be provided as the signature. 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 $\mathbf{t}$, i.e., it should be $f(\mathbf{t}) - v = \sum_{i\in[n]}(t_i - a_i)q_i(\mathbf{t})$. We note here that in the real construction, the terms in the above equation are encoded in the exponents of group elements, and therefore the verifier cannot directly check the above equation. However, the verifier can check the above condition using operations in the bilinear group, including the pairing operation which allows one to express one multiplication in the exponent. The bilinear group operations directly translate to checking the above condition in the exponent.

3.2 Detailed construction

We now present our *selectively* secure SCC scheme supporting multivariate polynomial evaluation.

Algorithm $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(\lambda, \mathcal{F})$**:** Suppose that the function family $\mathcal{F} \subseteq \mathbb{Z}_p[\mathbf{x}]$ 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 Equation 2.1). The KeyGen algorithm invokes the bilinear group generation algorithm to generate a bilinear group instance of prime order p (of λ bits), with a bilinear map function $\mathsf{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Then it chooses a random generator $g \in \mathbb{G}$ and a random point $\mathbf{t} = [t_1, t_2, \ldots, t_n] \in \mathbb{Z}_p^n$ and computes the *signature generation set* $\mathcal{W}_{n,d}$

$$\mathcal{W}_{n,d} = \left\{ g^{\prod_{i \in S} t_i^{S(i)}} : \forall S \in \mathcal{S}_{n,d} \right\} . \tag{3.2}$$

For example, $\mathcal{W}_{2,2}$ 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 $g, \mathcal{W}_{n,d}$ and the description of $\mathbb{G}, \mathbb{G}_T, \mathsf{e}$. The secret key SK contains the commitment point $\mathbf{t}$. We describe an optimization referring to reducing the number of group elements of $\mathcal{W}_{n,d}$ in the full version of the paper [31].

Algorithm $\mathsf{FK}(f) \leftarrow \mathsf{Setup}(\mathsf{SK}, \mathsf{PK}, f)$**:** Let $f(\mathbf{x}) \in \mathbb{Z}_p[\mathbf{x}]$ 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 Equation 2.1. The setup algorithm, by using the signature generation set $\mathcal{W}_{n,d}$ contained in PK, computes the polynomial public key, i.e.,

$$\mathsf{FK}(f) = g^{f(\mathbf{t})} = \left(g^{\prod_{i \in S_1} t_i^{S_1(i)}} \right)^{c_1} \times \left(g^{\prod_{i \in S_2} t_i^{S_2(i)}} \right)^{c_2} \times \ldots \times \left(g^{\prod_{i \in S_k} t_i^{S_k(i)}} \right)^{c_k} . \tag{3.3}$$

The algorithm outputs the function public key $\mathsf{FK}(f)$.

Algorithm $(v, w) \leftarrow \mathsf{Compute}(\mathsf{PK}, f, \mathbf{a})$**:** This algorithm first computes $v = f(\mathbf{a})$. Using Lemma 1, it finds an appropriate set of polynomials $q_1(\mathbf{x}), q_2(\mathbf{x}), \ldots, q_n(\mathbf{x})$ to express polynomial $f(\mathbf{x}) - v$ as $f(\mathbf{x}) - v = \sum_{i=1}^{n}(x_i - a_i)q_i(\mathbf{x})$. The signature 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 signature generation set $\mathcal{W}_{n,d}$, as is achieved for the function public key in Equation 3.3. It finally outputs the pair (v, w), where v is the outcome of the polynomial evaluated at $\mathbf{a}$, and w is the signature of correctness.

Algorithm $\mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f), \mathbf{a}, v, w)$**:** Parse PK as the signature generation set $\mathcal{W}_{n,d}$. To verify that v is indeed $f(\mathbf{a})$, given a signature $w = [w_1, w_2, \ldots, w_n]$, algorithm Verify checks if the following equation holds:

$$\mathsf{e}\left(\mathsf{FK}(f) g^{-v}, g\right) \stackrel{?}{=} \prod_{i=1}^{n} \mathsf{e}\left(g^{t_i - a_i}, w_i\right) . \tag{3.4}$$

In the above, the terms g^{t_i} are contained in PK (specifically in $\mathcal{W}_{n,d}$) and the function public key $\mathsf{FK}(f)$ equals $g^{f(\mathbf{t})}$. The algorithm accepts the result v, and outputs 1 if the above equations hold; otherwise, it rejects and outputs 0.

Algorithm $\mathsf{FK}(f') \leftarrow \mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{FK}(f), f')$**:** Let f denote the current polynomial and f' be the new polynomial that corresponds to the update. Assume f' and f differ in only one coefficient. Specifically, let S denote the multiset corresponding to that coefficient.[2] Suppose the current function public key is $\mathsf{FK}(f)$. The algorithm sets

$$\mathsf{FK}(f') = \mathsf{FK}(f) \cdot g^{(c'_S - c_S) \prod_{i \in S} t_i^{S(i)}} ,$$

updating $\mathsf{FK}(f)$ to $\mathsf{FK}(f')$, the new function public key. We now state our first theorem.

Theorem 1 *There exists an SCC scheme for polynomial evaluation such that (1) It is correct according to Definition 2; (2) For univariate polynomials, it is adaptively secure according to Definition 3 and under the ℓ-SBDH assumption; (3) For multivariate polynomials, it is selectively secure according to Definition 6 and under the ℓ-SBDH assumption.*

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

The correctness of our construction follows in a straightforward manner from Lemma 1, and the bilinear property of the pairing operation e. The asymptotic cost analysis of the scheme's algorithms are presented in Section 6. The security proofs in Section C.1 of the Appendix. However, we give a proof sketch in the following.

3.3 Selective security proof sketch

We briefly explain the selective security proof intuition of our scheme. The simulator obtains an ℓ-SBDH instance, $g, g^{\tau}, \ldots, g^{\tau^{\ell}} \in \mathbb{G}$ and it will construct a simulation such that if an adversary can break the selective security of the SCC scheme, the simulator can leverage it to break the ℓ-SBDH instance. Specifically, with knowledge of the challenge point $\mathbf{a} = [a_1, a_2, \ldots, a_n]$ that the adversary commits to at the beginning of the selective security game, the simulator can carefully craft the simulation such that $t_i - a_i = \lambda_i(\tau + c)$, where $t = [t_1, t_2, \ldots, t_n]$ represents the committed point used to compute the polynomial digest, and λ_i and c are constants known to the simulator.

If an adversary can forge a signature for a wrong outcome of a polynomial, then the simulator is able to raise terms in Equation 3.4 to the $(\tau + c)^{-1}$ power and output $\mathsf{e}(g, g)^{(\tau+c)^{-1}}$, breaking in this way the ℓ-SBDH assumption. Notice that in the selective security proof, the simulator's ability to take appropriate terms in Equation 3.4 to the $(\tau + c)^{-1}$ power relies on knowing the challenge point $\mathbf{a}$ in advance, and the ability to craft the simulation such that $t_i - a_i = \lambda_i(\tau + c)$.

4 Adaptively secure multivariate polynomial evaluation

In this section, we augment the above selectively secure SCC scheme to achieve adaptive security in the random oracle model. We also show that the same techniques can be applied to construct an adaptively secure PVC scheme under the formulation of Parno *et al.* [34] *without the random oracle model.*

4.1 Intuition

The intuition of the new construction is similar to the selectively secure construction. For technical reasons explained later, instead of relying on the polynomial decomposition method described in Lemma 1, we use a new decomposition that is *randomized*, so that it can later be manipulated by a simulator in the proof to achieve adaptive security. The decomposition we are using is the following:

Lemma 2 (Randomized decomposition) *Let $f(\mathbf{x}) \in \mathbb{Z}_p[\mathbf{x}]$ be an n-variate polynomial of degree at most d. For all $\mathbf{a} \in \mathbb{Z}_p^n$ and for all $r_1, \ldots, r_{n-1} \in \mathbb{Z}_p$ such that $r_1 r_2 \ldots r_{n-1} \neq 0$, there exist polynomials $q_i(\mathbf{x}) \in \mathbb{Z}_p[\mathbf{x}]$ such that the polynomial $f(\mathbf{x}) - f(\mathbf{a})$ can be expressed as*

$$f(\mathbf{x}) - f(\mathbf{a}) = \sum_{i=1}^{n-1} \left[r_i(x_i - a_i) + x_{i+1} - a_{i+1}\right] q_i(\mathbf{x}) + (x_n - a_n) q_n(x_n)\,,$$

where $q_n(x_n)$ is a polynomial of degree at most d that contains only variable x_n. Moreover, there exists a polynomial-time algorithm to find the above polynomials $q_i(\mathbf{x})$.

Proof: The direct of the claim is straightforward: If a polynomial $f(\mathbf{x}) - f(\mathbf{a})$ can be expressed as

$$\sum_{i \in [n-1]} \left[r_i(x_i - a_i) + x_{i+1} - a_{i+1}\right] q_i(\mathbf{x}) + (x_n - a_n) q_n(x_n)\,,$$

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 $f(\mathbf{x}) - f(\mathbf{a})$ by $r_1(x_1 - a_1) + x_2 - a_2$. Specifically,

$$f(x_1, x_2, \ldots, x_n) = \left[r_1(x_1 - a_1) + x_2 - a_2\right] \cdot q_1(x_1, x_2, \ldots, x_n) + s_1(x_2, x_3, \ldots, x_n)\,,$$

where $s_1(x_2, x_3, \ldots, x_n)$ is the remainder term, and $s_1(x_2, x_3, \ldots, x_n)$ should no longer contain the variable x_1. Next, divide $s_1(x_2, x_3, \ldots, x_n)$ by $r_2(x_2 - a_2) + x_3 - a_3$, and divide the remainder by $r3(x_3 - a_3) + x_4 - a_4$, and so on. For the last division we divide with $x_n - a_n$. In this way, we can write $f(\mathbf{x}) - f(\mathbf{a})$ as

$$\sum_{i \in [n-1]} [r_i(x_i - a_i) + x_{i+1} - a_{i+1}]\, q_i(\mathbf{x}) + (x_n - a_n) q_n(x_n) + s_n \,,$$

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

We note here that in our construction explained below, the numbers $r_1, r_2, \ldots, r_{n-1}$ mentioned in Lemma 2 will be chosen "at random" by calling a hash function modelled as a random oracle (see Equation 4.5).

4.2 Detailed construction

We now continue with the algorithms of our adaptively secure SCC scheme.

Algorithm $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(\lambda, \mathcal{F})$**:** Same as in Section 3.

Algorithm $\mathsf{FK}(f) \leftarrow \mathsf{Setup}(\mathsf{SK}, \mathsf{PK}, f)$**:** Same as in Section 3.

Algorithm $(v, w) \leftarrow \mathsf{Compute}(\mathsf{PK}, f, \mathbf{a})$**:** Parse $\mathbf{a}$ as $[a_1, a_2, \ldots, a_n]$. The algorithm first computes the outcome of the polynomial $v = f(\mathbf{a})$. Next, compute the following, where $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_p$ is a hash function (later modelled as a random oracle):

$$\forall 1 \le i \le n-1 : \quad r_i = \mathsf{H}(\mathbf{a}||i)\,. \tag{4.5}$$

Now, using Lemma 2, find an appropriate set of polynomials $q_1(\mathbf{t}), q_2(\mathbf{t}), \ldots, q_n(t_n)$ to express polynomial $f(\mathbf{x}) - f(\mathbf{a})$ as $\sum_{i=1}^{n-1} [r_i(x_i - a_i) + x_{i+1} - a_{i+1}]\, q_i(\mathbf{x}) + (x_n - a_n) q_n(x_n)$. Next, leverage the signature generation set $\mathcal{W}_{n,d}$ (see Equation 3.2) to compute $w_i = g^{q_i(\mathbf{t})}$ for $1 \le i \le n-1$. It is not hard to see that all w_i's can be computed from $\mathcal{W}_{n,d}$. The signature w is composed as

$$w = [w_1, w_2, \ldots, w_n,\ \text{polynomial } q_n(x_n)]\ ,$$

where the polynomial $q_n(x_n)$ contains the description of the polynomial, i.e., up to d coefficients $\beta_d, \ldots, \beta_0$, since it is a univariate polynomial in x_n of degree at most d.

The algorithm outputs the pair (v, w) denoting the outcome of the polynomial evaluated at $\mathbf{a}$, and a signature to vouch for the correctness of the computation.

Algorithm $\{0,1\} \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f), \mathbf{a}, v, w)$**:** Parse $\mathbf{a}$ as $[a_1, a_2, \ldots, a_n] \in \mathbb{Z}_p^n$; then parse the signature w as $[w_1, w_2, \ldots, w_{n-1},\ \text{polynomial } q_n(x_n)]$. To verify that v is indeed the outcome of the correct polynomial evaluated at point $\mathbf{a} \in \mathbb{Z}_p^n$, algorithm Verify first computes $g^{q_n(t_n)}$ using the signature generation set $\mathcal{W}_{n,d}$ (Equation 3.2) which is part of the public key PK.

Next, it computes the r_i values in the same way as in Equation 4.5, namely, $r_i = \mathsf{H}(\mathbf{a}||i)$ for $1 \le i \le n-1$. Finally, it checks if the following equation holds:

$$\mathsf{e}\left(\mathsf{FK}(f) \cdot g^{-v}, g\right) \overset{?}{=} \prod_{i=1}^{n-1} \mathsf{e}\left(g^{r_i(t_i - a_i) + t_{i+1} - a_{i+1}}, w_i\right) \mathsf{e}\left(g^{t_n - a_n}, g^{q_n(t_n)}\right)\ , \tag{4.6}$$

In the above, the terms g^{t_i} are contained in PK (specifically in $\mathcal{W}_{n,d}$) and $\mathsf{FK}(f)$ equals $g^{f(\mathbf{t})}$. The algorithm accepts if the above equation holds; otherwise, it rejects.

Algorithm $\mathsf{FK}(f') \leftarrow \mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{FK}(f), f')$**:** Same as in Section 3.

4.3 Adaptive security proof sketch

The simulator obtains an ℓ-SBDH instance, $g, g^{\tau}, \ldots, g^{\tau^{\ell}} \in \mathbb{G}$ and it will construct a simulation such that if an adversary can break the adaptive security of the SCC scheme, the simulator can leverage it to break the ℓ-SBDH instance. Unlike in the selective security proof of Section 3.3, without the adversary committing to the challenge point in advance, the simulator cannot craft terms to satisfy conditions such as $t_i - a_i = \lambda_i(t + c)$—but this condition is crucial later for the simulator to compute $\mathsf{e}(g, g)^{(\tau+c)^{-1}}$ and break the hardness assumption.

To circumvent this barrier in the proof, we embed "randomness" into the verification equation, such that the simulator can manipulate these random numbers to satisfy a condition described below, without having to know the challenge point ahead of time:

$$r_i(t_i - a_i) + t_{i+1} - a_{i+1} = \lambda_i(\tau + c) \quad \text{for } i = 1, \ldots, n-1\,, \tag{4.7}$$

where λ_i and c are constants known to the simulator.

Specifically, since these random numbers are outputs from a "random" hash function, under the random oracle model, the simulator can manipulate the answers to the random oracle queries in the simulation to achieve the above goal. Note that our SCC signature with adaptive security has size $O(n + d)$, as opposed to $O(n)$, which was the size of the signature in the selectively secure scheme (see Section 6). This is because it is essential the signature contain the polynomial $q_n(x_n)$ for the adaptive security proof to work, so that the simulator can divide both sides of Equation 4.6 with $\tau + c$. We can now state our main theorem (see detailed security proof in the Section C.2 of the Appendix).

Theorem 2 *There exists an SCC scheme for the evaluation of multivariate polynomials such that (1) It is correct according to Definition 2; (2) It is adaptively secure according to Definition 3, under the ℓ-SBDH assumption and in the random oracle model.*

4.4 An adaptively secure PVC scheme without random oracles

Our techniques can be readily adapted to construct an adaptively secure PVC scheme for multivariate polynomial evaluation—see Section 2.3. Nevertheless, if we were to use the observations of Section 2.3 as a black box, we would construct a PVC scheme that has the random oracle. However, we are able to remove the random oracle by taking advantage of the fact that PVC is weaker than SCC.

The resulting PVC scheme is very similar to our construction in this section—except that in the PVC scheme, the random numbers r_i's are directly chosen at random (as a challenge) by a client issuing a query to the untrusted server, instead of being the outputs of a hash function modeled as a random oracle. We provide the detailed PVC scheme construction in Section B.1 of the Appendix. Its full security proof can be found in Section C.3 of the Appendix.

Theorem 3 *There exists a PVC scheme for the evaluation of multivariate polynomials of total such that (1) It is correct according to Definition 8; (2) It is adaptively secure according to Definition 9 and under the ℓ-SBDH assumption.*

5 SCC schemes for polynomial differentiation

In this section, we construct an SCC scheme for the verification of differentiation queries. Given a multivariate polynomial $f(\mathbf{x})$, we show how to construct signatures of correct computation for derivatives $\partial^k f(\mathbf{x})/\partial x_j^k(\mathbf{a})$ evaluated at a chosen point $\mathbf{a}$.

One naive method to support verification of 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 $O(nk\binom{n+d}{d})$. In contrast, our construction requires only $O(\binom{n+d}{d})$ setup cost (see Section 6), the same with the polynomial evaluation scheme. Another drawback of the naive method is increased update cost, since an update operation would now involve updating all nk polynomials. In contrast, our construction allows for efficient incremental updates.

5.1 Intuition

The intuition of supporting polynomial differentiation is similar to the evaluation case. In place of the decomposition lemmas (Lemmas 1 and 2) for polynomial evaluation, we have the following counterparts (Lemmas 3 and 4) for derivative computation:

Lemma 3 (Decomposition for derivatives) *For $\mathbf{a} \in \mathbb{Z}_p^n$, the n-variate polynomial $f(\mathbf{x}) \in \mathbb{Z}_p[\mathbf{x}]$ can be expressed as*

$$f(\mathbf{x}) = \sum_{i=1}^{n-1}(x_i - a_i)u_i(\mathbf{x}) + (x_n - a_n)^{k+1}q(x_n) + c_k x_n^k + \ldots + c_1 x_n + c_0 \, .$$

Then, the k-th derivative of $f(\mathbf{x})$ wrt x_n equals $k! \cdot c_k$ at point $\mathbf{a}$, i.e., $\partial^k f(\mathbf{x})/\partial x_n^k(\mathbf{a}) = k! \cdot c_k$. A similar result holds for other variables x_i by variable renaming.

Proof: The proof is by explicit construction. The polynomial $u_1(\mathbf{x})$ is the quotient when dividing $f(\mathbf{x})$ by $(r_1 x_1 - a_1)$, the remainder is then divided by $(x_2 - a_2)$, resulting in the quotient $u_2(\mathbf{x})$, and the remainder is then divided by $(x_3 - a_3)$, and so on. This goes on until we divide the remainder with $(x_{n-1} - a_{n-1})$, at which point, we are left with a remainder $s(x_n)$ containing only the variable x_n. At this point, we divide $s(x_n)$ by $(x_n - a_n)^{k+1}$ resulting in the quotient $q(x_n)$, and the remainder is expressed as $c_k x_n^k + c_{k-1}x_n^{k-1} + \ldots + c_1 x_n + c_0$. We now show that $\partial^k f(\mathbf{x})/\partial x_n^k(\mathbf{a}) = k! \cdot c_k$. To do this, we analyze the k-th derivative with respect to x_n for each additive term of $f(\mathbf{x})$ expressed in the above form. Notice that for $1 \leq i \leq n-1$, we have

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

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

Lemma 4 (Randomized decomposition for derivatives) *For $\mathbf{a} \in \mathbb{Z}_p^n$ and for all $r_1, \ldots, r_{n-2} \in \mathbb{Z}_p$ such that $r_1 r_2 \ldots r_{n-2} \neq 0$, the n-variate polynomial $f(\mathbf{x}) \in \mathbb{Z}_p[\mathbf{x}]$ can be expressed as*

$$\begin{aligned} f(\mathbf{x}) &= \sum_{i=1}^{n-2}[r_i(x_i - a_i) + x_{i+1} - a_{i+1}]u_i(\mathbf{x}) + (x_{n-1} - a_{n-1})u_{n-1}(\mathbf{x}) \\ &+ (x_n - a_n)^{k+1}q(x_n) + c_k x_n^k + \sum_{i=0}^{k} c_i x_n^i \, , \end{aligned}$$

where $u_{n-1}(\mathbf{x})$ is a polynomial containing only variables x_{n-1} and x_n and $q(x_n)$ is a polynomial containing only variable x_n. Then, the k-th derivative of $f(\mathbf{x})$ wrt x_n equals $k! \cdot c_k$ at point $\mathbf{a}$, i.e., $\partial^k f(\mathbf{x})/\partial x_n^k(\mathbf{a}) = k! \cdot c_k$. A similar result holds for other variables x_i by variable renaming.

Proof: Same as proof of Lemma 3, with the difference that each time we are dividing with $r_i(x_i - a_i) + x_{i+1} - a_{i+1}$ (for $i = 1, \ldots, n-2$), after that we divide with $x_{n-1} - a_{n-1}$ and for $i = n$ we devide by $(x_n - a_n)$. By taking the derivatives as above, one can see that the derivative is again equal to $k! \cdot c_k$. ■

Similar to the multivariate polynomial evaluation case, Lemmas 3 and 4 allow us to construct respectively: 1) an SCC scheme for polynomial differentiation with *selective security*; and 2) an SCC scheme for polynomial differentiation with *adaptive security in the random oracle model* and a PVC scheme for polynomial differentiation with *adaptive security without the random oracle model* .

5.2 Detailed construction

We now present the *adaptively secure* SCC scheme for polynomial differentiation (based on Lemma 4). For completeness, we also present in the Appendix a selectively secure scheme for polynomial differentiation—see Sections B.2 and C.4 of the Appendix for the detailed construction and its proof of security respectively.

Algorithm $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(\lambda, \mathcal{F})$**:** Same as in Section 3.

Algorithm $\mathsf{FK}(f) \leftarrow \mathsf{Setup}(\mathsf{SK}, \mathsf{PK}, f)$**:** Same as in Section 3.

Algorithm $(v, w) \leftarrow \mathsf{Compute}(\mathsf{PK}, f, \mathbf{a}, k, \mathsf{ind})$**:** In addition to the point $\mathbf{a} \in \mathbb{Z}_p^n$, the $\mathsf{Compute}$ algorithm here takes in two additional parameters k and ind, indicating the evaluation of the k-th derivative of the polynomial with respect to variable x_{ind} at $\mathbf{a}$. Without loss of generality, below we assume $\mathsf{ind} = n$. In other words, the algorithm should evaluate the k-th partial derivative with respect to x_n at point $\mathbf{a}$. First, the algorithm computes randomness r_i as

$$r_i = \mathsf{H}(\mathbf{a}||\mathsf{ind}||k||i) \;\; \forall 1 \leq i \leq n-2\,, \tag{5.8}$$

where $\mathsf{H} : \{0,1\}^* \rightarrow \mathbb{Z}_p$ is a hash function (later modeled as a random oracle). Due to Lemma 4, $f(\mathbf{x})$ can be expressed as $f(\mathbf{x}) = \sum_{i=1}^{n-2}[r_i(x_i - a_i) + x_{i+1} - a_{i+1}]u_i(\mathbf{x}) + (x_{n-1} - a_{n-1})u_{n-1}(\mathbf{x}) + (x_n - a_n)^{k+1}q(x_n) + \sum_{i=0}^{k} c_i x_n^i$. The signature w for correct derivative computation is the following tuple:

$$w = \left(g^{u_1(\mathbf{t})}, \ldots, g^{u_{n-2}(\mathbf{t})},\; g^{q(t_n)},\;\; c_{k-1}, \ldots, c_1, c_0,\;\; \text{polynomial } u_{n-1}(\mathbf{x})\right)\,,$$

where polynomial $u_{n-1}(\mathbf{x})$ is a description of the polynomial containing the corresponding coefficients. Note that by Lemma 4, polynomial $u_{n-1}(\mathbf{x})$ contains up to d^2 terms. Also, the signature does not contain the term c_k—this can be implicitly retrieved by the result v since $c_k = v/k!$. Finally, the result of the computation v is returned.

Algorithm $\mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f), \mathbf{a}, k, \mathsf{ind}, v, w)$**:** Let $c_k = \frac{v}{k!}$. To verify that v is indeed the outcome of the k-th partial derivative on variable x_{ind} ($\mathsf{ind} = n$) evaluated at point $\mathbf{a} \in \mathbb{Z}_p^n$, perform the following steps.

Parse w as $(w_1, \ldots, w_{n-2}, w_n, c_{k-1}, \ldots, c_1, c_0,\;\; \text{polynomial } u_{n-1}(\mathbf{x}))$.

Compute the r_i values in the same way as in Equation 5.8, i.e., $r_i = \mathsf{H}(\mathbf{a}||\mathsf{ind}||k||i)$ for $1 \leq i \leq n-2$.

Check if $\mathsf{e}\left(\mathsf{FK}(f), g\right)$ equals the following quantity (where $\mathsf{L} = \prod_{i=0}^{k} \mathsf{e}\left(g^{t_n^i}, g\right)^{c_i}$):

$$\prod_{i=1}^{n-2} \mathsf{e}\left(g^{r_i(t_i - a_i) + t_{i+1} - a_{i+1}}, w_i\right) \cdot \mathsf{e}\left(g^{t_{n-1} - a_{n-1}}, g^{u_{n-1}(\mathbf{t})}\right) \cdot \mathsf{e}\left(g^{(t_n - a_n)^{k+1}}, w_n\right) \cdot \mathsf{L}\,,$$

The above quantity can be easily computed with the public keys in $O(n + d^2)$ time, since $u_{n-1}(\mathbf{x})$ is a polynomial containing d^2 terms and $k \leq d$ (see Section 6). The algorithm accepts v and outputs 1 if the above equation holds; otherwise, it rejects.

Algorithm $\mathsf{FK}(f') \leftarrow \mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{FK}(f), f')$**:** Same as in Section 3.

Theorem 4 *There exists an SCC scheme for the differentiation of multivariate polynomials such that (1) It is correct according to Definition 2; (2) It is adaptively secure according to Definition 3, under the ℓ-SBDH assumption and in the random oracle model.*

The proof of security for the above theorem can be found in Section C.5 of the Appendix. By following the same techniques that we used in Theorem 3, we can construct a PVC scheme without random oracles for polynomial differentiation as well:

Corollary 1 *There exists a PVC scheme for the differentiation of multivariate polynomials such that (1) It is correct according to Definition 8; (2) It is adaptively secure according to Definition 9 and under the ℓ-SBDH assumption.*

6 Asymptotic cost analysis

In this section, we analyze the asymptotic cost of our schemes. Clearly, the worst-case complexity of KeyGen is $O(\binom{n+d}{d})$, since the set $\mathcal{W}_{n,d}$ should contain one term for every possible term of the polynomial in n variables and total degree d. Similarly algorithm Setup takes $O(\binom{n+d}{d})$ time to execute in the worst case. In practice, both these complexities can be $O(m)$, where m is the number of the terms contained in the polynomial—see Section D of the Appendix for minimizing the size of $\mathcal{W}_{n,d}$.

Also for our adaptive security schemes, the size of the signature is $O(n)$, and the client performs $O(n)$ amount of work to verify it using algorithm Verify (these costs are $O(n+d)$ for derivative computation). For our adaptive security schemes, the size of the signature increases to $O(n+d)$, and the client performs $O(n+d)$ amount of work to verify it (again, these costs $O(n+d^2)$ for derivative computation).

As for algorithm Compute, it needs to decompose the polynomial according to Lemmata 1, 2, 3, 4 (depending on which scheme we are using). This polynomial decomposition dominates the asymptotic performance. To perform the polynomial decomposition, the server performs n polynomial divisions. If we use 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 (Lemma 1) can be achieved in $O(nd\binom{n+d}{d})$ time using the naive algorithm. However, 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. Finally, algorithm Update takes constant time to update a constant number of coefficients.

7 Extensions and observations

7.1 I/O privacy

In our constructions, the client's sensitive input is in plaintext, directly readable by the untrusted server. To offer input and output privacy, we could potentially use a *fully-homomorphic* public-key encryption scheme [17] (FHE scheme) so that algorithm Compute executed by the untrusted server could operate on encrypted points. In this way, everybody that knows pk could send queries to the server. After Compute executes on the encryption of some point $\bar{\mathbf{a}}$, it outputs the encrypted signature w of the value $\bar{v} = f(\bar{\mathbf{a}})$ under the public key pk, allowing only the owner of the secret key to decrypt and retrieve (and verify) the output of the computation. This could have various applications which we highlight in Section E of the Appendix.

7.2 Removing the random oracles through stronger assumptions

We now observe that if we are willing to (i) use subexponential assumptions and (ii) restrict the size of the domain of the inputs of our polynomials to be subexponential (now it is exponential), we can remove the random oracle from our adaptively secure constructions. The subexponential assumption we use can be stated as follows:

Definition 5 (δ-subexponential bilinear ℓ-strong Diffie-Hellman assumption) *Suppose k is the security parameter, let $0 < \delta < \frac{\log k - 1}{\log k}$ 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 $\mathbb{Z}_p^*$, for $\ell = \mathsf{poly}(k)$, there is no algorithm running in time less than 2^{2k^δ} 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 $\mathsf{neg}(k)$.*

Note that in the above definition, we require $\delta < \frac{\log k - 1}{\log k}$ so that $2k^\delta < k$.

Theorem 5 (Adaptive security in the standard model) *Let $\mathbf{x}$ be the input to our polynomial. For $\mathbf{x}$ belonging to a domain of subexponential size, our selectively secure scheme (Section 3) is adaptively secure in the standard model and assuming the δ-subexponential bilinear ℓ-strong Diffie-Hellman assumption.*

Namely, for all PPT adversaries, we can build a simulator running in subexponential time that breaks the δ-subexponential bilinear ℓ-strong Diffie-Hellman assumption (see Definition 5).

Proof: Suppose we have n variables $x_1, x_2, \ldots, x_n$, and each one of which can take values in $[0, 1, \ldots, m-1]$. Assume that $m^n = 2^{k^\delta}$, yielding $n \log m = k^\delta$. To build the desired simulator, we modify the initialization phase of our selective security proof in Section 3.3: We do not require the adversary to commit to an initial point $\mathbf{a}$. Instead the simulator guesses the point $\mathbf{a}$ that the adversary is going to output later as a forgery—and the simulator aborts if the guess is wrong. Clearly, the guess is successful with probability 2^{-k^δ}. Therefore the simulation, in expectation, takes 2^{k^δ} time to succeed. Since the adversary runs in at most polynomial time (see our adaptive security definition), it follows that we have derived an algorithm that runs in $\mathsf{poly}(k)2^{k^\delta}$ time and breaks the assumption. Note that this is a contradiction since the function $\mathsf{poly}(k)2^{k^\delta} = o(2^{2k^\delta})$. This completes our proof. ■

The same technique was also described by Boneh and Boyen [7] to achieve adaptive security in their IBE scheme.

Acknowledgments

This work was supported by Intel through the ISTC for Secure Computing, by the National Science Foundation under grants CCF-0424422, 0842695, 0808617 and CNS-1228485, 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 by a NetApp Faculty Fellowship. The authors thank Xavier Boyen, Basilis Gidas, Matthew Green, Jonathan Katz, Dawn Song, Aishwarya Thiruvengadam, Nikos Triandopoulos and Brent Waters for useful discussions, and the TCC 2013 reviewers for their feedback. 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.

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

[4] 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*, pages 326–349, 2012.

[5] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. *IACR Cryptology ePrint Archive*, 2012:95, 2012.

[6] D. Boneh and X. Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptology*, 21(2):149–177, 2008.

[7] D. Boneh and X. Boyen. Efficient selective identity-based encryption without random oracles. *J. Cryptology*, 24(4):659–693, 2011.

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

[9] R. Canetti, B. Riva, and G. N. Rothblum. Two 1-round protocols for delegation of computation. *IACR Cryptology ePrint Archive*, 2011:518, 2011.

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

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

[12] I. Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *CRYPTO*, pages 445–456, 1991.

[13] D. Fiore and R. Gennaro. Improved publicly verifiable delegation of large polynomials and matrix computations. Cryptology ePrint Archive, Report 2012/434, 2012. `http://eprint.iacr.org/`.

[14] D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *ACM Conference on Computer and Communications Security*, 2012.

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

[16] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. Cryptology ePrint Archive, Report 2012/215, 2012. `http://eprint.iacr.org/`.

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

[18] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.

[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] J. Groth. Short non-interactive zero-knowledge proofs. In *ASIACRYPT*, pages 341–358, 2010.

[24] V. Guruswami and C. Wang. Optimal rate list decoding via derivative codes. In *APPROX-RANDOM*, pages 593–604, 2011.

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

[26] A. B. Lewko and B. Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In *CRYPTO*, pages 180–198, 2012.

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

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

[29] S. Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[30] L. Nguyen. Accumulators from bilinear pairings and applications. In *Proc. RSA Conference, Cryptographers' Track (CT-RSA) , LNCS 3376, pp. 275-292, Springer.*, 2005.

[31] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. Cryptology ePrint Archive, Report 2011/587, 2011. `http://eprint.iacr.org/`.

[32] 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.

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

[34] B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, 2012.

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

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

[37] 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.

[38] R. Tamassia and N. Triandopoulos. Certification and authentication of data structures. In *Proc. Alberto Mendelzon Workshop on Foundations of Data Management*, 2010.

A Definitions

Definition 6 (Selective security of an SCC scheme) *Let λ be the security parameter and let $\mathcal{P}$ be an SCC scheme* $(\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Compute}, \mathsf{Verify}, \mathsf{Update})$ *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 game between $\mathcal{A}$ and a challenger:*

1. ***Initialization.*** *The adversary $\mathcal{A}$ commits to a point* $\mathbf{b}$. *The challenger runs algorithm* KeyGen *which outputs* $(\mathsf{PK}, \mathsf{SK})$ *and gives* PK *to $\mathcal{A}$ but maintains* SK *secret;*
2. ***Setup and Update.*** *The adversary $\mathcal{A}$ initially makes an oracle query to algorithm* $\mathsf{Setup}(\mathsf{SK}, \mathsf{PK}, f_0)$*, specifying an initial function $f_0 \in \mathcal{F}$, outputting* $\mathsf{FK}(f_0)$*. Then, for $i = 1, \ldots, k$, where $k = \mathsf{poly}(\lambda)$, he makes a polynomial number of oracle queries to the* $\mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{FK}(f_{i-1}), f_i)$ *algorithm, each time specifying $f_i \in \mathcal{F}$. The challenger answers the queries by returning the resulting* $\mathsf{FK}(f_i)$*;*
3. ***Forgery.*** *The adversary $\mathcal{A}$ outputs a forgery $(\mathbf{b}, v, w)$ for point $\mathbf{b}$ that he committed in the initialization phase, for some function f_i previously queried where $0 \leq i \leq k$.*

The adversary $\mathcal{A}$ wins if $1 \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f_i), \mathbf{b}, v, w)$ *and* $f_i(\mathbf{b}) \neq v$.

Definition 7 (PVC scheme) *We define a* PVC scheme *for a function family $\mathcal{F}$ to be a tuple of six PPT algorithms* $(\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Challenge}, \mathsf{Compute}, \mathsf{Verify}, \mathsf{Update})$ *with the following specification:*

1. $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(\lambda, \mathcal{F})$*: Algorithm* KeyGen *takes as input the security parameter λ and a function family $\mathcal{F}$. It outputs a public/secret key pair* $(\mathsf{PK}, \mathsf{SK})$. KeyGen *is run only once at system initialization by a trusted source;*
2. $\mathsf{FK}(f) \leftarrow \mathsf{Setup}(\mathsf{SK}, \mathsf{PK}, f)$*: Algorithm* Setup *(run by a trusted source) takes as input the secret key* SK*, the public key* PK*, and a function $f \in \mathcal{F}$. It outputs the function public key* $\mathsf{FK}(f)$ *for the function f;*
3. $\mathsf{chal}(\mathbf{a}) \leftarrow \mathsf{Challenge}(\mathsf{PK}, \mathbf{a})$*: Algorithm* $\mathsf{Challenge}$ *(run by the verifier) takes as input a value* $\mathbf{a} \in \mathsf{domain}(f)$. *It outputs a challenge* $\mathsf{chal}(\mathbf{a})$ *corresponding to* $\mathbf{a}$*;*
4. $(v, w) \leftarrow \mathsf{Compute}(\mathsf{PK}, f, \mathbf{a}, \mathsf{chal}(\mathbf{a}))$*: Algorithm* $\mathsf{Compute}$ *(run by an untrusted server) takes as input the public key* PK*, a function $f \in \mathcal{F}$ and a value* $\mathbf{a} \in \mathsf{domain}(f)$. *It outputs a pair (v, w), where $v = f(\mathbf{a})$ and w is a signature;*
5. $\{0,1\} \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f), \mathbf{a}, \mathsf{chal}(\mathbf{a}), v, w)$*: Algorithm* Verify *(run by the verifier) takes as input the public key* PK*, function public key* $\mathsf{FK}(f)$*, value* $\mathbf{a} \in \mathsf{domain}(f)$*, a claimed result v and a signature w. It outputs 0 or 1;*
6. $\mathsf{FK}(f') \leftarrow \mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{FK}(f), f')$*: Algorithm* Update *(run by the trusted source) takes as input the secret key* SK*, the public key* PK*, the function public key* $\mathsf{FK}(f)$ *for the old function f and the updated function description f'. It outputs the updated function public key* $\mathsf{FK}(f')$.

Definition 8 (Correctness of a PVC scheme) *Let λ be the security parameter and let $\mathcal{P}$ be a PVC scheme* $(\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Challenge}, \mathsf{Compute}, \mathsf{Verify}, \mathsf{Update})$ *for a function family $\mathcal{F}$. 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* $\mathsf{FK}(f_i)$ *is the output of* $\mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{FK}(f_{i-1}), f_i)$*, where* $\mathsf{FK}(f_0)$ *is output by algorithm* $\mathsf{Setup}(\mathsf{SK}, \mathsf{PK}, f_0)$ *for some $f_0 \in \mathcal{F}$. We say that $\mathcal{P}$ is* correct*, if for any* $i = 0, \ldots, \mathsf{poly}(\lambda)$*, for any* $\mathbf{a} \in \mathsf{domain}(f_i)$*, for any* $\mathsf{chal}(\mathbf{a})$ *output by* $\mathsf{Challenge}(\mathsf{PK}, \mathbf{a})$*, it is* $1 \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f_i), \mathbf{a}, \mathsf{chal}(\mathbf{a})), v, w)$*, where* $(v, w) \leftarrow \mathsf{Compute}(\mathsf{PK}, f_i, \mathbf{a}, \mathsf{chal}(\mathbf{a})))$.

Definition 9 (Adaptive security of a PVC scheme) *Let λ be the security parameter and let $\mathcal{P}$ be a PVC scheme* $(\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Challenge}, \mathsf{Compute}, \mathsf{Verify}, \mathsf{Update})$ *for a function family $\mathcal{F}$. We say that $\mathcal{P}$ is* adaptively secure *if no PPT adversary $\mathcal{A}$ has more than negligible probability* $\mathsf{neg}(\lambda)$ *in winning the following security game, played between the adversary $\mathcal{A}$ and a challenger:*

1. ***Initialization.*** *The challenger runs algorithm* KeyGen *which outputs* $(\mathsf{PK}, \mathsf{SK})$ *and then gives* PK *to the adversary but maintains* SK *secret;*
2. ***Setup and Update.*** *The adversary initially makes an oracle query to algorithm* $\mathsf{Setup}(\mathsf{SK}, \mathsf{PK}, f_0)$*, specifying an initial function $f_0 \in \mathcal{F}$, outputting* $\mathsf{FK}(f_0)$*. Then, for $i = 1, \ldots, k$, where* $k = \mathsf{poly}(\lambda)$*, he makes a polynomial number of oracle queries to the* $\mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{FK}(f_{i-1}), f_i)$ *algorithm, each time specifying $f_i \in \mathcal{F}$. The challenger answers the queries by returning the resulting* $\mathsf{FK}(f_i)$*;*
3. ***Challenge and Forgery.*** *The adversary $\mathcal{A}$ outputs a point* $\mathbf{b}$ *and sends it to the challenger. The challenger returns* $\mathsf{chal}(\mathbf{b})$ *output by* $\mathsf{Challenge}$. *The adversary $\mathcal{A}$ outputs the forgery* $(\mathbf{b}, \mathsf{chal}(\mathbf{b}), v, w)$ *for one of the functions f_i $(0 \leq i \leq k)$ that has been queried.*

The adversary $\mathcal{A}$ wins if $1 \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f_i), \mathbf{b}, \mathsf{chal}(\mathbf{b}), v, w)$ *and* $f_i(\mathbf{b}) \neq v$.

B Constructions

B.1 Construction of adaptively secure PVC for multivariate polynomial evaluation

In this section, we present our adaptively-secure PVC construction without random oracles. We give the algorithms of our new PVC scheme. Not that since we are in the PVC model, apart from algorithm Compute and Verify, we need to also present algorithm Challenge. The remaining algorithms (KeyGen, Setup and Update) are the same

Algorithm $\mathsf{chal}(\mathbf{a}) \leftarrow \mathsf{Challenge}(\mathsf{PK}, \mathbf{a})$**:** Let $\mathbf{a} = [a_1\ a_2 \ldots\ a_n]$ be an input point. The algorithm picks $r_1, r_2, \ldots, r_{n-1} \in \mathbb{Z}_p$ uniformly at random and outputs a challenge $\mathsf{chal}(\mathbf{a}) = [r_1\ r_2\ \ldots\ r_{n-1}]$.

Algorithm $(v, w) \leftarrow \mathsf{Compute}(\mathsf{PK}, f, \mathbf{a}, \mathsf{chal}(\mathbf{a}))$**:** Parse $\mathbf{a}$ as $[a_1\ a_2\ \ldots\ a_n]$ and $\mathsf{chal}(\mathbf{a})$ as $[r_1\ r_2\ \ldots\ r_{n-1}]$. The algorithm first computes $v = f(\mathbf{a})$. Using Lemma 2, it finds an appropriate set of polynomials $q_1(\mathbf{t}), q_2(\mathbf{t}), \ldots, q_n(t_n)$ to express polynomial $f(\mathbf{x}) - f(\mathbf{a})$ in the canonical form as

$$f(\mathbf{x}) - f(\mathbf{a}) = \sum_{i \in [n-1]} \left[r_i(x_i - a_i) + x_{i+1} - a_{i+1}\right] q_i(\mathbf{x}) + (x_n - a_n) q_n(x_n)\,.$$

The witness w contains a vector of $n-1$ witnesses $w_1, w_2, \ldots, w_{n-1}$, such that $w_i = g^{q_i(\mathbf{t})}$ for all $i \in [n-1]$ and the coefficients $\beta_d, \beta_{d-1}, \ldots, \beta_0$ of the polynomial $q_n(x_n)$. The algorithm 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 $(v, w) \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f), \mathbf{a}, \mathsf{chal}(\mathbf{a}), v, w)$**:** Parse $\mathbf{a}$ as $[a_1\ a_2\ \ldots\ a_n] \in \mathbb{Z}_p^n$, $\mathsf{chal}(\mathbf{a})$ as $[r_1\ r_2\ \ldots\ r_{n-1}] \in \mathbb{Z}_p^{n-1}$ and w as $[w_1\ w_2\ \ldots\ w_{n-1}] \in \mathbb{G}^{n-1}$ and $[\beta_d\ \beta_{d-1}\ \ldots\ \beta_0] \in \mathbb{Z}_p^{d+1}$. To verify that v is indeed the outcome of the correct polynomial evaluated at point $\mathbf{a} \in \mathbb{Z}_p^n$, algorithm Verify checks if the following equation holds:

$$\mathsf{e}\left(\mathsf{FK}(f) g^{-v}, g\right) \stackrel{?}{=} \prod_{i \in [n-1]} \mathsf{e}\left(g^{r_i(t_i - a_i) + t_{i+1} - a_{i+1}}, w_i\right) \mathsf{e}\left(g^{t_n - a_n}, g^{\pi(t_n)}\right)\,, \tag{B.9}$$

where $\pi(t_n) = \beta_d t_n^d + \beta_{d-1} t_{n-1}^{d-1} + \ldots + \beta_0$. In the above, the terms g^{t_i} are contained in PK (specifically in $\mathcal{W}_{n,d}$) and $\mathsf{FK}(f)$ equals $g^{f(\mathbf{t})}$. The algorithm accepts the result v, and outputs 1 if the above equations hold; otherwise, it rejects and outputs 0.

B.2 Construction of selectively secure SCC scheme for multivariate polynomial differentiation

In this section we provide the detailed construction for the verification of derivatives evaluation in the selective security model. We describe in detail algorithms Compute and Verify. All other algorithms of the SCC scheme are the same.

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 $i = n$. In other words, the algorithm should evaluate the k-th partial derivative with respect to x_n at point $\mathbf{a}$. Due to Lemma 3, $f(\mathbf{x})$ can be expressed as $f(\mathbf{x}) = \sum_{i=1}^{n-1}(x_i - a_i) u_i(\mathbf{x}) + (x_n - a_n)^{k+1} q(x_n) + c_k x_n^k + \ldots + c_1 x_n + c_0$. The signature w for the derivative computation is the following tuple:

$$\left(g^{u_1(\mathbf{t})}, g^{u_2(\mathbf{t})}, \ldots, g^{u_{n-1}(\mathbf{t})}, g^{q(t_n)}, c_{k-1}, \ldots, c_1, c_0\right) \in \mathbb{G}^n \times \mathbb{Z}_p^k\,.$$

Note that the proof does not contain the term c_k. This is going to be implicitly retrieved by the result of the derivative computation v, i.e., $c_k = v/k!$. Finally, the result of the computation v is returned.

Algorithm $\mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f), \mathbf{a}, v, w, k, i)$**:** Let $c_k = \frac{v}{k!}$. Parse PK as the signature generation set $\mathcal{W}_{n,d}$. To verify that v is indeed the outcome of the k-th partial derivative on variable x_n evaluated at point $\mathbf{a} \in \mathbb{Z}_p^n$, perform the following steps. Parse w as $(w_1, \ldots, w_{n-1}, \omega, c_{k-1}, \ldots, c_1, c_0)$. Given the witness w, algorithm Verify checks if the following equation holds:

$$\mathsf{e}\left(\mathsf{FK}(f), g\right) \stackrel{?}{=} \prod_{i=1}^{n-1} \mathsf{e}\left(g^{t_i} g^{-a_i}, w_i\right) \cdot \mathsf{e}\left(g^{(t_n - a_n)^{k+1}}, \omega\right) \cdot \prod_{i=0}^{k} \mathsf{e}\left(g^{t_n^i}, g\right)^{c_i}\,.$$

In the above, all the used expressions are computable by using the elements in $\mathcal{W}_{n,d}$. Also, $\mathsf{FK}(f)$ equals $g^{f(\mathbf{t})}$. The algorithm accepts v and outputs 1 if the above equation holds; otherwise, it outputs 0 and rejects.

C Proofs

C.1 Selectively secure SCC for multivariate polynomial evaluation

Since for $n = 1$ our scheme is the same with the scheme of Kate et al. [25], its adaptive security follows from that work. We now prove selective security for the case of $n > 1$.

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 SCC scheme — such that if an adversary $\mathcal{A}$ can break the full security of the SCC scheme with more than negligible probability when interacting with the simulator, 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.

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

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

1. **Initialization.** The adversary first commits to a challenge point

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

The simulator runs algorithm KeyGen which outputs $(\mathsf{PK}, \mathsf{SK})$ and then gives PK to the adversary but maintains SK secret. To do that, the simulator needs to choose a random point $\mathbf{t}$ and create the corresponding signature generation set $\mathcal{W}_{n,d}$ at the chosen point $\mathbf{t}$. 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 $\mathcal{W}_{n,d}$. 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 simulator can still compute all terms in $\mathcal{W}_{n,d}$ since when one plugs in Equations C.37, it is not hard to see that all terms in $\mathcal{W}_{n,d}$ are essentially of the form $g^{q(\tau)}$, where $q(\tau)$ is some polynomial of degree at most d. Since the simulator knows the values

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

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 KeyGen and Setup algorithms, since every v_i and w_i are chosen uniformly at random.

2. **Setup and Update.** 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 $\mathsf{FK}(f_0)$. Then, for $i = 1, \ldots, k$, where $k = \mathsf{poly}(\lambda)$, he makes a polynomial number of oracle queries to the $\mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{FK}(f_{i-1}), f_i)$ algorithm, each time specifying $f_i \in \mathcal{F}$. The simulator answers the queries by returning the resulting $\mathsf{FK}(f_i)$. It is not hard to see that the simulator does not need to know the t_i's to update $\mathsf{FK}(f_i)$;

3. **Forgery.** The adversary outputs a forgery for the committed point $\mathbf{a}$. The forgery consists of some $\mathsf{FK}(f)$, a claimed outcome v of the polynomial at $\mathbf{a}$, and a signature $w = (w_1, w_2, \ldots, w_n)$. Due to the security of the signature scheme, $\mathsf{FK}(f)$ 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 $\mathsf{FK}(f)$. If the forgery is successful, the following must be true: $v \neq f(\mathbf{a})$ and $\mathsf{Verify}(\mathsf{FK}, \mathbf{x}, v, w) = 1$. The simulator will now leverage this forgery to break the ℓ-SBDH instance it got from the challenger.
   Specifically, let $\delta = v - f(\mathbf{a}) \neq 0 \in \mathbb{Z}_p$, i.e., the difference between the true outcome and the claimed outcome. Since the verification succeeds, we have
$$\mathsf{e}\left(g, g\right)^{f(\mathbf{t})-v} = \prod_{i\in[n]} \mathsf{e}\left(g^{t_i - a_i}, w_i\right) .$$
   Or equivalently,
$$\mathsf{e}\left(g, g\right)^{\delta} = \mathsf{e}\left(g, g\right)^{f(\mathbf{t})-f(\mathbf{a})} \cdot \prod_{i\in[n]} \mathsf{e}\left(g^{t_i - a_i}, w_i^{-1}\right) . \tag{C.13}$$
   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:
$$\mathsf{e}\left(g, g\right)^{\delta} = \prod_{i\in[n]} \mathsf{e}\left(g^{t_i - a_i}, g^{q_i(\mathbf{t})}\right) \cdot \prod_{i\in[n]} \mathsf{e}\left(g^{t_i - a_i}, w_i^{-1}\right) . \tag{C.14}$$
   Or equivalently,
$$\mathsf{e}(g, g)^{\delta} = \prod_{i\in[n]} \mathsf{e}\left(g^{t_i - a_i}, \quad g^{q_i(\mathbf{t})} \cdot w_i^{-1}\right) . \tag{C.15}$$
   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 $\mathsf{e}(g, g)^{\frac{\delta}{\tau - a_1}}$—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.37, C.11, and C.12, it is not hard to see that
$$t_i - a_i = r_i(\tau - a_1) .$$
   As a result,
$$\left(\prod_{i\in[n]} \mathsf{e}\left(g^{t_i - a_i}, \quad g^{q_i(\mathbf{t})} \cdot w_i\right)\right)^{\frac{1}{\tau - a_1}} = \prod_{i\in[n]} \mathsf{e}\left(g^{r_i}, \quad g^{q_i(\mathbf{t})} \cdot w_i^{-1}\right) .$$
   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
$$e\left(g, g\right)^{\frac{1}{\tau - a_1}} = \left(\prod_{i\in[n]} \mathsf{e}\left(g^{r_i}, \quad g^{q_i(\mathbf{t})} \cdot w_i^{-1}\right)\right)^{\delta^{-1}} ,$$
   breaking in this way the ℓ-SBDH assumption. This completes the proof.

C.2 Adaptively secure SCC 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 SCC scheme—such that if an adversary $\mathcal{A}$ can break the full security of the SCC scheme with more than negligible probability when interacting with the simulator, 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. Our proof is under the random oracle model, i.e., the simulator also implements a random oracle for the adversary to query.

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

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

1. **Initialization.** The simulator runs algorithm KeyGen which outputs $(\mathsf{PK}, \mathsf{SK})$ and then gives PK to the adversary but maintains SK secret. To do that, the simulator needs to choose a random point $\mathbf{t}$ and create the corresponding signature generation set $\mathcal{W}_{n,d}$ at the chosen point $\mathbf{t}$. The simulator implicitly sets
$$t_i = \chi_i \tau + \zeta_i \text{ for } i = 1, \ldots, n\,, \tag{C.16}$$
where χ_i and ζ_i are chosen uniformly at random. The simulator remembers the values of all χ_i's and ζ_i's for later usage. Now the simulator needs to compute $\mathcal{W}_{n,d}$. 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 simulator can still compute all terms in $\mathcal{W}_{n,d}$ since when one plugs in Equations C.37, it is not hard to see that all terms in $\mathcal{W}_{n,d}$ are essentially of the form $g^{q(\tau)}$, where $q(\tau)$ is some polynomial of degree at most d. Since the simulator knows the values
$$\left(g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^{\ell}}\right) ,$$
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 KeyGen and Setup algorithms, since every v_i and w_i are chosen uniformly at random.
The simulator also chooses a random secret $c \in \mathbb{Z}_p$, which will be useful later in answering random oracle queries.
2. **Setup and Update.** 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 $\mathsf{FK}(f_0)$. Then, for $i = 1, \ldots, k$, where $k = \mathsf{poly}(\lambda)$, he makes a polynomial number of oracle queries to the $\mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{FK}(f_{i-1}), f_i)$ algorithm, each time specifying $f_i \in \mathcal{F}$. The simulator answers the queries by returning the resulting $\mathsf{FK}(f_i)$. It is not hard to see that the simulator does not need to know the t_i's to update $\mathsf{FK}(f_i)$;
3. **Random oracle queries.** Upon each random oracle query $\mathbf{a}||i$, the simulator guesses if $\mathbf{a}$ is going to be the challenge point in the challenge phases. Since the same $\mathbf{a}$ may be queried multiple times with different values of i, it is possible that the guess has been made before. In this case, just use the previous guess for $\mathbf{a}$. Let q_o denote the total number of random oracle queries. The simulator can guess right with probability at least $1/q_o$. If it turns out that the simulator guessed wrong, it simply aborts the simulation.
If the simulator guesses that $\mathbf{a}$ is not the challenge point, the simulator chooses a random number and returns it to the adversary.
If the simulator guesses that $\mathbf{a}$ will bet the challenge point, the simulator carefully crafts the response r_i so that it satisfies the following equations:
$$r_i(t_i - a_i) + t_{i+1} - a_{i+1} = \lambda_i(\tau + c) \tag{C.17}$$

where c was chosen uniformly at random in the initialization stage of the simulation. However, note that in the initialization phase, the simulator had set $t_i = \chi_i\tau + \zeta_i$ for $i = 1, \ldots, n$. By doing some algebra, one can see that, for Equation C.17 to hold for all $\tau \in \mathbb{Z}_p$, the simulator needs to set

$$r_i = \frac{\zeta_{i+1} - c\chi_{i+1} - a_{i+1} - a_i}{c\chi_i - \zeta_i} \tag{C.18}$$

and

$$\lambda_i = \frac{\chi_i\zeta_{i+1} - \chi_{i+1}\zeta_i - \chi_i a_{i+1} - \chi_i a_i}{c\chi_i - \zeta_i} \tag{C.19}$$

where $c\chi_i - \zeta_i \neq 0$ except with negligible probability. The simulator now returns this r_i to the adversary.

4. **Forgery.** The adversary outputs a forgery $(\mathbf{a}, v, w)$. The forgery consists of a claimed outcome $v \neq f(\mathbf{a})$ of the polynomial at $\mathbf{a}$, a witness $w = (w_1, w_2, \ldots, w_{n-1})$ and coefficients $\beta_d, \beta_{d-1}, \ldots, \beta_0$ of a polynomial $\pi(t_n)$, namely it holds

$$\pi(t_n) = \beta_d t_n^d + \beta_{d-1} t_{n-1}^{d-1} + \ldots + \beta_0 \,.$$

If it turns out that the challenge point $\mathbf{a}$ was not what the simulator guessed in the random oracle queries, then the simulator simply aborts.

The simulator will now leverage this forgery to break the ℓ-SBDH instance it got from the challenger. Specifically, let $\delta = f(\mathbf{a}) - v \neq 0 \in \mathbb{Z}_p$, i.e., the difference between the true outcome and the claimed outcome. Since the verification succeeds, we have

$$\mathsf{e}\,(g,g)^{f(\mathbf{t})-v} = \prod_{i\in[n-1]} \mathsf{e}\left(g^{r_i(t_i-a_i)+t_{i+1}-a_{i+1}}, w_i\right) \mathsf{e}\left(g^{t_n-a_n}, g^{\pi(t_n)}\right) \,.$$

Or equivalently,

$$\mathsf{e}\,(g,g)^{f(\mathbf{t})-f(\mathbf{a})+\delta} = \prod_{i\in[n-1]} \mathsf{e}\left(g^{r_i(t_i-a_i)+t_{i+1}-a_{i+1}}, w_i\right) \mathsf{e}\left(g^{t_n-a_n}, g^{\pi(t_n)}\right) \,. \tag{C.20}$$

Due to Corollary 2, the simulator can find polynomials $q_1(\mathbf{t}), q_2(\mathbf{t}), \ldots, q_n(t_n)$ such that

$$f(\mathbf{t}) - f(\mathbf{a}) = \sum_{i\in[n-1]} \left[r_i(t_i - a_i) + t_{i+1} - a_{i+1}\right] q_i(\mathbf{t}) + (t_n - a_n) q_n(t_n) \,.$$

Therefore, we can re-write Equation C.20 as below (note that $q_n(t_n)$ is a polynomial in one variable, the variable t_n):

$$\mathsf{e}\,(g,g)^{\delta+(t_n-a_n)[q_n(t_n)-\pi(t_n)]} = \prod_{i\in[n-1]} \mathsf{e}\left(g^{r_i(t_i-a_i)+t_{i+1}-a_{i+1}}, w_i g^{-q_i(\mathbf{t})}\right) \,. \tag{C.21}$$

Note now that by Equation C.17, it is $r_i(t_i - a_i) + t_{i+1} - a_{i+1} = \lambda_i(\tau + c)$. Therefore Equation C.23 can be written as

$$\mathsf{e}\,(g,g)^{\delta+(t_n-a_n)[q_n(t_n)-\pi(t_n)]} = \prod_{i\in[n-1]} \mathsf{e}\left(g^{\lambda_i(\tau+c)}, w_i g^{-q_i(\mathbf{t})}\right) \,. \tag{C.22}$$

Consider now the polynomial $z(t_n) = \delta + (t_n - a_n)[q_n(t_n) - \pi(t_n)]$, which is a polynomial in one variable t_n. Also recall that $t_n = \chi_n\tau + \zeta_n$. Therefore

$$z(t_n) = y(\tau) = \delta + (\chi_n\tau + \zeta_n - a_n)[q_n(\chi_n\tau + \zeta_n) - \pi(\chi_n\tau + \zeta_n)] \,.$$

Note that since $\delta \neq 0$, $y(\tau)$ is *not* the zero polynomial. Therefore, the simulator can divide it with $t + c$ and therefore it can be written as

$$y(\tau) = \Pi(\tau)(\tau + c) + \gamma\,,$$

where not both $\Pi(t)$ and γ can be zero. Specifically, $\gamma \neq 0$ with overwhelming probability $1 - d/2^k$. This is because if $\gamma = 0$, $\tau + c$ divides $y(\tau)$. However $y(\tau)$ has at most d divisors of the form $(\tau + A)$ (as many as its maximum number of roots) and c is chosen at random from $\mathbb{Z}_p$. Therefore Equation C.22 can be written as

$$\mathsf{e}\left(g, g\right)^{\Pi(\tau)(\tau+c)+\gamma} = \prod_{i\in[n-1]} \mathsf{e}\left(g^{\lambda_i(\tau+c)}, w_i g^{-q_i(\mathbf{t})}\right)\,, \tag{C.23}$$

which gives

$$\mathsf{e}\left(g, g\right)^{\frac{1}{\tau+c}} = \left(\prod_{i\in[n-1]} \mathsf{e}\left(g^{\lambda_i}, w_i g^{-q_i(\mathbf{t})}\right) \mathsf{e}\left(g, g\right)^{-\Pi(\tau)}\right)^{\gamma^{-1}}\,.$$

Therefore the simulator is able to break the ℓ-SBDH assumption. This completes the proof.

C.3 Adaptively secure PVC 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 PVC scheme—such that if an adversary $\mathcal{A}$ can break the full security of the PVC scheme with more than negligible probability when interacting with the simulator, 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.

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

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

1. **Initialization.** The simulator runs algorithm KeyGen which outputs $(\mathsf{PK}, \mathsf{SK})$ and then gives PK to the adversary but maintains SK secret. To do that, the simulator needs to choose a random point $\mathbf{t}$ and create the corresponding signature generation set $\mathcal{W}_{n,d}$ at the chosen point $\mathbf{t}$. The simulator implicitly sets

$$t_i = \chi_i \tau + \zeta_i \text{ for } i = 1, \ldots, n\,, \tag{C.24}$$

where χ_i and ζ_i are chosen uniformly at random. The simulator remembers the values of all χ_i's and ζ_i's for later usage. Now the simulator needs to compute $\mathcal{W}_{n,d}$. 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 simulator can still compute all terms in $\mathcal{W}_{n,d}$ since when one plugs in Equations C.37, it is not hard to see that all terms in $\mathcal{W}_{n,d}$ are essentially of the form $g^{q(\tau)}$, where $q(\tau)$ is some polynomial of degree at most d. Since the simulator knows the values

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

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 KeyGen and Setup algorithms, since every χ_i and ζ_i are chosen uniformly at random.

2. **Setup and Update.** 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 $\mathsf{FK}(f_0)$. Then, for $i = 1, \ldots, k$, where $k = \mathsf{poly}(\lambda)$, he makes a polynomial number of oracle queries to the $\mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{FK}(f_{i-1}), f_i)$ algorithm, each time specifying $f_i \in \mathcal{F}$. The simulator answers the queries by returning the resulting $\mathsf{FK}(f_i)$. It is not hard to see that the simulator does not need to know the t_i's to update $\mathsf{FK}(f_i)$;
3. **Challenge and forgery.** The adversary $\mathcal{A}$ outputs a point $\mathbf{a} \in \mathsf{domain}(f_k)$ and sends point $\mathbf{a}$ to the simulator. The simulator outputs $\mathsf{chal}(\mathbf{a})$, not by calling $\mathsf{Challenge}(\mathsf{PK}, \mathbf{b})$, but in the following way. He carefully computes the random numbers $r_1, r_2, \ldots, r_{n-1}$ so that they satisfy the equations

$$r_i(t_i - a_i) + t_{i+1} - a_{i+1} = \lambda_i(\tau + c) \text{ for } i = 1, \ldots, n-1\,, \tag{C.25}$$

where c is chosen uniformly at random. However, note that in the initialization phase, the simulator had set $t_i = v_i\tau + w_i$ for $i = 1, \ldots, n$. By doing some algebra, one can see that, for Equation C.25 to hold for all $\tau \in \mathbb{Z}_p$, the simulator needs to set

$$r_i = \frac{\zeta_{i+1} - c\chi_{i+1} - a_{i+1} - a_i}{c\chi_i - \zeta_i} \text{ for } i = 1, \ldots, n-1\,, \tag{C.26}$$

and

$$\lambda_i = \frac{\chi_i\zeta_{i+1} - \chi_{i+1}\zeta_i - \chi_i a_{i+1} - \chi_i a_i}{c\chi_i - \zeta_i} \text{ for } i = 1, \ldots, n-1\,, \tag{C.27}$$

where $c\chi_i - \zeta_i \neq 0$ with overwhelming probability. Note that each number r_i has two degrees of freedom (w_{i+1} and v_{i+1}), therefore, since w_i and v_i are chosen uniformly at random at initialization phase, it follows that $r_1, r_2, \ldots, r_{n-1}$ appear random to the adversary. The simulator now sends $\mathsf{chal}(\mathbf{b}) = [r_1\ r_2 \ldots\ r_{n-1}]$ to the adversary, where each r_i is given in Equation C.26.
Then the adversary outputs the forgery $(\mathbf{a}, v, w)$. The forgery consists of a claimed outcome $v \neq f(\mathbf{a})$ of the polynomial at $\mathbf{a}$, a witness $w = (w_1, w_2, \ldots, w_{n-1})$ and coefficients $\beta_d, \beta_{d-1}, \ldots, \beta_0$ of a polynomial $\pi(t_n)$, namely it holds

$$\pi(t_n) = \beta_d t_n^d + \beta_{d-1} t_{n-1}^{d-1} + \ldots + \beta_0\,.$$

The simulator will now leverage this forgery to break the ℓ-SBDH instance it got from the challenger. Specifically, let $\delta = f(\mathbf{a}) - v \neq 0 \in \mathbb{Z}_p$, i.e., the difference between the true outcome and the claimed outcome. Since the verification succeeds, we have

$$\mathsf{e}\,(g,g)^{f(\mathbf{t})-v} = \prod_{i\in[n-1]} \mathsf{e}\left(g^{r_i(t_i-a_i)+t_{i+1}-a_{i+1}}, w_i\right) \mathsf{e}\left(g^{t_n-a_n}, g^{\pi(t_n)}\right)\,.$$

Or equivalently,

$$\mathsf{e}\,(g,g)^{f(\mathbf{t})-f(\mathbf{a})+\delta} = \prod_{i\in[n-1]} \mathsf{e}\left(g^{r_i(t_i-a_i)+t_{i+1}-a_{i+1}}, w_i\right) \mathsf{e}\left(g^{t_n-a_n}, g^{\pi(t_n)}\right)\,. \tag{C.28}$$

Due to Lemma 2, the simulator can find polynomials $q_1(\mathbf{t}), q_2(\mathbf{t}), \ldots, q_n(t_n)$ such that

$$f(\mathbf{t}) - f(\mathbf{a}) = \sum_{i\in[n-1]} \left[r_i(t_i - a_i) + t_{i+1} - a_{i+1}\right] q_i(\mathbf{t}) + (t_n - a_n)q_n(t_n)\,.$$

Therefore, we can re-write Equation C.28 as below (note that $q_n(t_n)$ is a polynomial in one variable, the variable t_n):

$$\mathsf{e}\,(g,g)^{\delta+(t_n-a_n)[q_n(t_n)-\pi(t_n)]} = \prod_{i\in[n-1]} \mathsf{e}\left(g^{r_i(t_i-a_i)+t_{i+1}-a_{i+1}}, w_i g^{-q_i(\mathbf{t})}\right)\,. \tag{C.29}$$

Note now that by Equation C.25, it is $r_i(t_i - a_i) + t_{i+1} - a_{i+1} = \lambda_i(\tau + c)$. Therefore Equation C.29 can be written as

$$\mathsf{e}\left(g,g\right)^{\delta+(t_n-a_n)[q_n(t_n)-\pi(t_n)]} = \prod_{i\in[n-1]} \mathsf{e}\left(g^{\lambda_i(\tau+c)}, w_i g^{-q_i(\mathbf{t})}\right) . \tag{C.30}$$

Consider now the polynomial $z(t_n) = \delta + (t_n - a_n)[q_n(t_n) - \pi(t_n)]$, which is a polynomial in one variable t_n. Also recall that $t_n = \chi_n\tau + \zeta_n$. Therefore

$$z(t_n) = y(\tau) = \delta + (\chi_n\tau + \zeta_n - a_n)[q_n(\chi_n\tau + \zeta_n) - \pi(\chi_n\tau + \zeta_n)] .$$

Note that since $\delta \neq 0$, $y(\tau)$ is *not* the zero polynomial. Therefore, the simulator can divide it with $t + c$ and therefore it can be written as

$$y(\tau) = \Pi(\tau)(\tau + c) + \gamma ,$$

where not both $\Pi(t)$ and γ can be zero. Specifically, $\gamma \neq 0$ with overwhelming probability $1 - d/2^k$. This is because if $\gamma = 0$, $\tau + c$ divides $y(\tau)$. However $y(\tau)$ has at most d divisors of the form $(\tau + A)$ (as many as its maximum number of roots) and c is chosen at random from $\mathbb{Z}_p$. Therefore Equation C.31 can be written as

$$\mathsf{e}\left(g,g\right)^{\Pi(\tau)(\tau+c)+\gamma} = \prod_{i\in[n-1]} \mathsf{e}\left(g^{\lambda_i(\tau+c)}, w_i g^{-q_i(\mathbf{t})}\right) , \tag{C.31}$$

which gives

$$\mathsf{e}\left(g,g\right)^{\frac{1}{\tau+c}} = \left(\prod_{i\in[n-1]} \mathsf{e}\left(g^{\lambda_i}, w_i g^{-q_i(\mathbf{t})}\right) \mathsf{e}\left(g,g\right)^{-\Pi(\tau)}\right)^{\gamma^{-1}} .$$

Therefore the simulator is able to break the ℓ-SBDH assumption. This completes the proof.

C.4 Selectively secure SCC for multivariate polynomial differentiation

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 SCC scheme—such that if an adversary $\mathcal{A}$ can break the full security of the SCC scheme with more than negligible probability when interacting with the simulator, 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.

Assume that the simulator $\mathcal{S}$ obtains the following ℓ-SBDH assumption from the challenger $\mathcal{C}$, where $\ell = (k+1)d$:

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

1. **Initialization.** The adversary first commits to a challenge point

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

The simulator runs algorithm KeyGen which outputs $(\mathsf{PK}, \mathsf{SK})$ and then gives PK to the adversary but maintains SK secret. To do that, the simulator needs to choose a random point $\mathbf{t}$ and create the corresponding signature generation set $\mathcal{W}_{n,d}$ at the chosen point $\mathbf{t}$. 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

$$t_j = \tau\,, \tag{C.32}$$

without actually computing t_j.
For $i \neq j$, the simulator picks random $r_i \in \mathbb{Z}_p$, and implicitly chooses

$$t_i = r_i(\tau - a_j)^{k+1} + a_i\,, \tag{C.33}$$

without actually computing the values. The simulator remembers the values of all r_i's for later usage. Now the simulator needs to compute $\mathcal{W}_{n,d}$. 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 simulator can still compute all terms in $\mathcal{W}_{n,d}$ since when one plugs in Equations C.37, it is not hard to see that all terms in $\mathcal{W}_{n,d}$ are essentially of the form $g^{q(\tau)}$, where $q(\tau)$ is some polynomial of degree at most d. Since the simulator knows the values

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

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 KeyGen algorithm.

2. **Setup and Update.** 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 $\mathsf{FK}(f_0)$. Then, for $i = 1, \ldots, k$, where $k = \mathsf{poly}(\lambda)$, he makes a polynomial number of oracle queries to the $\mathsf{Update}(\mathsf{SK}, \mathsf{PK}, \mathsf{FK}(f_{i-1}), f_i)$ algorithm, each time specifying $f_i \in \mathcal{F}$. The simulator answers the queries by returning the resulting $\mathsf{FK}(f_i)$. It is not hard to see that the simulator does not need to know the t_i's to update $\mathsf{FK}(f_i)$;
3. **Forgery.** The adversary outputs a forgery for the committed point $\mathbf{a}$, evaluating the derivative

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

at point $\mathbf{a}$. 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 $\mathsf{FK}(f)$, 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, $\mathsf{FK}(f)$ 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 $\mathsf{FK}(f)$. If the forgery is successful, the following must be true:

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

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:

$$\mathsf{e}\left(g^{f(\mathbf{t})}, g\right) = \prod_{i \neq j} \mathsf{e}\left(g^{t_i} g^{-a_i}, w_i\right) \cdot \mathsf{e}\left(g^{(t_j - a_j)^{k+1}}, \omega\right) \cdot \prod_{i=0}^{k} \mathsf{e}\left(g^{t_j^i}, g\right)^{c_i}\,, \tag{C.34}$$

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

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

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.34) still holds. Rewrite Equation C.34 as:

$$\prod_{i=0}^{k} \mathsf{e}\left(g^{t_j^i}, g\right)^{\hat{c}_i - c_i} = \mathsf{e}\left(g^{(t_j-a_j)^{k+1}}, \omega g^{-\hat{q}(\mathbf{t})}\right) \prod_{i\neq j} \mathsf{e}\left(g^{t_i - a_i}, w_i g^{-\hat{u}(\mathbf{t})}\right) . \tag{C.35}$$

Due to Equation C.33, for $i \neq j$, we have

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

The simulator can raise the right-hand side of Equation C.35 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:

$$\mathsf{e}\left(g, g\right)^{\frac{\sum_{i=0}^{k}\Delta_i \tau^i}{(\tau - a_j)^{k+1}}} = \mathsf{e}\left(g, \omega g^{-\hat{q}(\mathbf{t})}\right) \prod_{i\neq j} \mathsf{e}\left(g^{r_i}, w_i g^{-\hat{u}(\mathbf{t})}\right) . \tag{C.36}$$

Note here that $\sum_{i=0}^{k}\Delta_i\tau^i$ is not the zero polynomial since, $\hat{c}_k - c_k \neq 0$—this is due to the fact that v is not the correct derivative and $\hat{c}_k = v/k!$ but however $c_k = u/k!$, where u is the correct derivative. We now prove the following lemma.

Lemma 5 *Given* $\mathsf{e}\left(g, g\right)^{\frac{\sum_{i=0}^{k}\Delta_i \tau^i}{(\tau - a_j)^{k+1}}}$ *from Equation C.36, 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.36 is equivalent to

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

yielding

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

which eventually gives

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

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

C.5 Adaptively secure SCC for multivariate polynomial differentiation

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 SCC scheme—such that if an adversary $\mathcal{A}$ can break the full security of the SCC scheme with more than negligible probability when interacting with the simulator, 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. Our proof is under the random oracle model, i.e., the simulator also implements a random oracle for the adversary to query.

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

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

1. **Initialization.** The simulator runs algorithm KeyGen which outputs $(\mathsf{PK},\mathsf{SK})$ and then gives PK to the adversary but maintains SK secret. To do that, the simulator needs to choose a random point $\mathbf{t}$ and create the corresponding signature generation set $\mathcal{W}_{n,d}$ at the chosen point $\mathbf{t}$. The simulator implicitly sets $t_n = \tau$ and
$$t_i = \chi_i\tau + \zeta_i \text{ for } i = 1,\ldots,n-1\,, \tag{C.37}$$
where χ_i and ζ_i are chosen uniformly at random. The simulator remembers the values of all χ_i's and ζ_i's for later usage. Now the simulator needs to compute $\mathcal{W}_{n,d}$. 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 simulator can still compute all terms in $\mathcal{W}_{n,d}$ since when one plugs in Equations C.37, it is not hard to see that all terms in $\mathcal{W}_{n,d}$ are essentially of the form $g^{q(\tau)}$, where $q(\tau)$ is some polynomial of degree at most d. Since the simulator knows the values
$$\left(g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^{\ell}}\right)\,,$$
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 KeyGen and Setup algorithms, since every v_i and w_i are chosen uniformly at random.
The simulator also chooses a random secret $c \in \mathbb{Z}_p$, which will be useful later in answering random oracle queries.
2. **Setup and Update.** 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 $\mathsf{FK}(f_0)$. Then, for $i = 1,\ldots,k$, where $k = \mathsf{poly}(\lambda)$, he makes a polynomial number of oracle queries to the $\mathsf{Update}(\mathsf{SK},\mathsf{PK},\mathsf{FK}(f_{i-1}),f_i)$ algorithm, each time specifying $f_i \in \mathcal{F}$. The simulator answers the queries by returning the resulting $\mathsf{FK}(f_i)$. It is not hard to see that the simulator does not need to know the t_i's to update $\mathsf{FK}(f_i)$;

3. **Random oracle queries.** Upon each random oracle query $\mathbf{a}||\mathsf{ind}||k||i$, the simulator guesses if $(\mathbf{a}, \mathsf{ind}, k)$ is going to be the challenge point in the challenge phases. Since the same $(\mathbf{a}, \mathsf{ind}, k)$ may be queried multiple times with different values of i, it is possible that the guess has been made before. In this case, just use the previous guess for $(\mathbf{a}, \mathsf{ind}, k)$. Let q_o denote the total number of random oracle queries. The simulator can guess right with probability at least $1/q_o$. If it turns out that the simulator guessed wrong, it simply aborts the simulation.
   If the simulator guesses that $(\mathbf{a}, \mathsf{ind}, k)$ is not the challenge point, the simulator chooses a random number and returns it to the adversary.
   If the simulator guesses that $(\mathbf{a}, \mathsf{ind}, k)$ will bet the challenge point, the simulator carefully crafts the response r_i so that it satisfies the following equations:

$$r_i(t_i - a_i) + t_{i+1} - a_{i+1} = \lambda_i(\tau - a_n)\,. \tag{C.38}$$

   However, note that in the initialization phase, the simulator had set $t_i = \chi_i\tau + \zeta_i$ for $i = 1, \ldots, n-1$ and $t_n = \tau$ By doing some algebra, one can see that, for Equation C.38 to hold for all $\tau \in \mathbb{Z}_p$, the simulator needs to set

$$r_i = \frac{\zeta_{i+1} - a_n\chi_{i+1} - a_{i+1} - a_i}{c\chi_i - \zeta_i} \text{ for } i = 1, \ldots, n-2\,, \tag{C.39}$$

   and

$$\lambda_i = \frac{\chi_i\zeta_{i+1} - \chi_{i+1}\zeta_i - \chi_i a_{i+1} - \chi_i a_i}{a_n\chi_i - \zeta_i} \text{ for } i = 1, \ldots, n-2\,, \tag{C.40}$$

   where $a_n\chi_i - \zeta_i \neq 0$ except with negligible probability. The simulator now returns this r_i to the adversary.

4. **Forgery.** The adversary outputs a forgery $(\mathbf{a}, k, v, w)$. The forgery consists of a claimed outcome $v \neq f^{[k]}(\mathbf{a})$ of the polynomial at $\mathbf{a}$, a witness $w = (w_1, w_2, \ldots, w_{n-2})$, the polynomials $u_{n-1}(\mathbf{x})$, w_n and the numbers $c_0, c_2, \ldots, c_k$. If it turns out that the challenge point $\mathbf{a}$ was not what the simulator guessed in the random oracle queries, then the simulator simply aborts.
   Since the verification succeeds, we have

$$\mathsf{e}(g,g)^{f(\mathbf{t})} = \prod_{i=1}^{n-2} \mathsf{e}\left(g^{r_i(t_i-a_i)+t_{i+1}-a_{i+1}}, w_i\right) \cdot \mathsf{e}\left(g^{t_{n-1}-a_{n-1}}, g^{u_{n-1}(\mathbf{t})}\right) \cdot \mathsf{e}\left(g^{(t_n-a_n)^{k+1}}, w_n\right) \cdot \prod_{i=0}^{k} \mathsf{e}\left(g^{t_n^i}, g\right)^{c_i}\,. \tag{C.41}$$

   Due to Lemma 4, the simulator can find polynomials $u_1(\mathbf{t}), u_2(\mathbf{t}), \ldots, u_{n-2}(\mathbf{t}), \hat{u}_{n-1}(\mathbf{t})$ and $\hat{q}(t_n)$ and numbers $\hat{c}_i$ for $i = 0, \ldots, k$ such that

$$f(\mathbf{x}) = \sum_{i=1}^{n-2}[r_i(x_i-a_i)+x_{i+1}-a_{i+1}]u_i(\mathbf{x}) + (x_{n-1}-a_{n-1})u_{n-1}(\mathbf{x}) + (x_n-a_n)^{k+1}q(x_n) + c_k x_n^k + \sum_{i=0}^{k} c_i x_n^i\,.$$

   Therefore, by replacing $r_i(t_i - a_i) + t_{i+1} - a_{i+1} = \lambda_i(\tau - a_n)$ from Equation C.17, we can re-write Equation C.41 as below

$$\mathsf{e}(g,g)^{\sum_{i=0}^{k}(\hat{c}_i - c_i)t_n^i} = \prod_{i=1}^{n-2} \mathsf{e}\left(g^{\lambda_i(\tau-a_n)}, w_i g^{-u_i(\mathbf{t})}\right) \cdot \mathsf{e}\left(g^{t_{n-1}-a_{n-1}}, g^{u_{n-1}(\mathbf{t})-\hat{u}_{n-1}(\mathbf{t})}\right) \cdot \mathsf{e}\left(g^{(t_n-a_n)^{k+1}}, w_n g^{-\hat{q}(t_n)}\right)\,,$$

   which can be further be written as

$$\mathsf{e}(g,g)^{\sum_{i=0}^{k}(\hat{c}_i - c_i)t_n^i - (t_{n-1}-a_{n-1})u_{n-1}(\mathbf{t}) - \hat{u}_{n-1}(\mathbf{t})} = \prod_{i=1}^{n-2} \mathsf{e}\left(g^{\lambda_i(\tau+c)}, w_i g^{-u_i(\mathbf{t})}\right) \mathsf{e}\left(g^{(t_n-a_n)^{k+1}}, w_n g^{-\hat{q}(t_n)}\right)\,.$$

Note now that both side are divisible by $(\tau - a_n)$ since $t_i = \chi_i \tau + \zeta_i = \lambda_i(\tau - a_n)$ for $i \neq n$ (and $t_n = \tau$) and the polynomial in the exponent at the left hand side is not the zero polynomial (this is because $\hat{c}_k \neq c_k$ since the derivative is wrong). Therefore the simulator can write the above equation as

$$\mathsf{e}(g,g)^{(\tau-a_n)\Pi(t)+\gamma} = \prod_{i=1}^{n-2} \mathsf{e}\left(g^{\lambda_i(\tau-a_n)}, w_i g^{-u_i(\mathbf{t})}\right) \mathsf{e}\left(g^{(\tau-a_n)^{k+1}}, w_n g^{-\hat{q}(t_n)}\right) .$$

which gives

$$\mathsf{e}\left(g,g\right)^{\frac{1}{\tau-a_n}} = \left(\prod_{i=1}^{n-2} \mathsf{e}\left(g^{\lambda_i}, w_i g^{-u_i(\mathbf{t})}\right) \mathsf{e}\left(g^{(t_n-a_n)^k}, w_n g^{-\hat{q}(t_n)}\right) \mathsf{e}\left(g,g\right)^{-\Pi(\tau)}\right)^{\gamma^{-1}} .$$

Therefore the simulator is able to break the ℓ-SBDH assumption. This completes the proof.

D Smaller signature generation set for smaller polynomials

In our basic constructions, the signature generation set $\mathcal{W}_{n,d}$ contains one term corresponding to every possible monomial with at most n variables and degree d. Therefore, the signature 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 $\mathcal{W}_{n,d}$:

$$\forall T \subseteq S : g^{\prod_{i \in T} t_i^{T(i)}} , \tag{D.42}$$

where $T \subseteq S$ if and only if $T(i) \leq S(i)\ \forall i \in [n]$. 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 D.42 to the server during an update.

E Input and output privacy

Consider a verifiable computation setting involving a pharmaceutical company: We showed in previous sections how a PVC scheme is required to assure that the untrusted provider, on input the genome data received by the patients, correctly executes the genome analysis algorithm on behalf of the pharmaceutical company. However, the patient's sensitive genome data is still in plaintext, directly readable by the untrusted provider! To offer input and output privacy, we could potentially use a *fully-homomorphic* public-key encryption scheme [17] (FHE scheme) so that algorithm Compute executed by the untrusted server could operate on encrypted points. In this way, everybody that knows pk (i.e., the public key of user A) could send queries to the server. After Compute executes on the encryption of some point a, it outputs the encrypted version of a witness w and of the value $v = P(a)$ under the public key pk, allowing only the owner of the secret key to decrypt and retrieve (and verify) the output of the computation.

One of the main limitations of the PVC scheme with I/O privacy is the fact that Alice cannot verify the output to a query that was encrypted under Bob's public key, since this would require Alice having access to Bob's secret key, so that she can decrypt the output of algorithm Compute. However, our scheme with I/O privacy retains its core public verifiability properties and has the following unique features:

1. Consider the pharmaceutical company example again, where a doctor is responsible for collecting the outputs of the function f executed on the genome of many patients. In this way, the doctor could study the results and send his personalized recommendations to the patients. Our model makes that feasible by having all patients encrypting their genome data under the doctor's public key. Then Algorithm Compute will execute and will eventually output results that can only be decrypted by the doctor. This enables the doctor to analyze *verifiable* computations across patients in a *private* way (i.e., only the doctor will learn information about the patients data). Note that this was not feasible in recent works on verifiable computation in the secret key setting (e.g., [1, 9, 10, 15]), where a doctor would only be able to analyze *his* data;
2. Consider now a different scenario where a every patient would like to execute function f in a verifiable fashion on his genome data without revealing the output to anyone else. In this case, a patient could encrypt his genome under his public key and have the cloud execute Compute under his public key. Note that this *does not involve a preprocessing phase of complexity proportional to the size of the circuit* evaluated by Compute at the client side. After the execution of algorithm Setup by the pharmaceutical company (which is done once), the only action required by each patient is executing an algorithm to encrypt the input which takes constant time. This property again is not achievable in recent works on verifiable computation in the secret key setting (e.g., [1, 9, 10, 15]), where every patient would have to do separate expensive preprocessing based on *his* secret key.

F Applications

Polynomial evaluation has a wide range of applications in everyday cloud computations as they are widely used for statistical purposes (e.g., computing statistical measures, machine learning) and scientific computing (e.g., solving a system of equations). Moreover several useful constructions are also enabled by verifying polynomial operations, such as accumulators [30], verifiable set operations [33], verifiable databases [3] and proofs of retrievability [3].

Although applications of polynomial evaluation are more straightforward, one might wonder what is the meaning of a derivative in $\mathbb{Z}_p$ and why its verification can be of any use. Derivatives in $\mathbb{Z}_p$ have applications in coding theory. For example, the r-th *Hasse* derivative $f^{[r]}(x)$ of polynomials in finite fields is used by various families of codes [11], and is directly related to classic Newton-Leibniz r-th derivative $f^{(r)}(x)$ through a simple formula, i.e., $f^{(r)}(x) = r!f^{[r]}(x)$. Therefore, verification of the classic derivative in $\mathbb{Z}_p$ directly enables the verification of the Hasse derivative. Also, recently Guruswami and Wang [24] presented a family of codes, called *derivative codes* whose function depends on the evaluations of the first $m-1$ classic Newton-Leibniz derivatives at n distinct field elements. Therefore, we can envision the encoding and decoding algorithms of such family of codes running in the cloud and having respective clients verifying their function by using our publicly verifiable derivative evaluation schemes.