Publicly Verifiable Delegation of Large Polynomials and Matrix Computations, with Applications

Dario Fiore[1] and Rosario Gennaro[2]

[1] Department of Computer Science, New York University, USA
fiore@cs.nyu.edu
[2] IBM Research, USA
rosario@us.ibm.com

Abstract. Outsourced computations (where a client requests a server to perform some computation on its behalf) are becoming increasingly important due to the rise of Cloud Computing and the proliferation of mobile devices. Since cloud providers may not be trusted, a crucial problem is the verification of the integrity and correctness of such computation, possibly in a *public* way, i.e., the result of a computation can be verified by any third party, and requires no secret key – akin to a digital signature on a message. We present new protocols for publicly verifiable secure outsourcing of *Evaluation of High Degree Polynomials* and *Matrix Multiplication*. Compared to previously proposed solutions, ours improve in efficiency and offer security in a stronger model. The paper also discusses several practical applications of our protocols.

1 Introduction

The rise of *Cloud Computing* (a computational infrastructure that allows businesses to lease computing resources from a service provider) raises several new security problems that must be addressed by the research community. In particular, a fundamental component of any secure cloud computing approach is a mechanism that enforces the integrity and correctness of the computations done by the provider on behalf of a client.

This problem can be modeled as following: a computationally weak client asks a powerful server to perform some computation on its behalf. The server must provide the result of the computation together with a "certificate" of its correctness. Crucially, the verification of such correctness proof must be substantially "easier" than the computation that was initially outsourced, since otherwise the client would either not be able to verify the proof, or would perform the computation on its own to begin with. This verification mechanism should not come at the expense of increasing the server's overhead: in other words producing the certificate should be "almost for free" for the server, which might provide this service to many clients, and therefore it should not become a computational bottleneck.

Outsourced computations are also increasingly important for mobile devices, such as smart phones and netbooks, which might resort to a network server to perform heavy computations, e.g., a cryptographic operation or a photo manipulation. Here too, an efficiently verifiable proof of the correctness might be required.

An important question is whether the verification of the computation can be *public*: i.e., can any third party (possibly different from the client who outsourced the computation) verify it? This is important, for example, in contexts where the computation has to be checked by several clients who cannot necessarily share a secret key, or if the proof of correctness must be transferable – similarly to a digital signature on a message.

Our Contributions. The main results of this paper are two new protocols for *publicly verifiable* secure outsourcing of:

- *Evaluation of High Degree Polynomials.* In this case the client stores a large (degree d) polynomial F with the server and then requests the value $y = F(x)$ for several inputs x.
- *Matrix Multiplication.* Here the client stores a large $(n \times d)$ matrix M with the server and then provides an d-dimensional vector $\boldsymbol{x}$ and obtains $\boldsymbol{y} = M \cdot \boldsymbol{x}$. Note that this immediately generalizes to matrix multiplication (providing input M' and obtaining $Y = M \cdot M'$ by applying the above solution to each column of M').

Our schemes are in the *amortized model* of [8] in which the client invests a one-time expensive computation phase ($O(d)$ in the polynomial case, $O(nd)$ in the matrix case) when storing the data with the server. Then verification of each evaluation will be fast ($o(d)$ and $o(nd)$ respectively).

Other Contributions: As a crucial tool to establish our results, we rely on the use of *pseudo-random functions with closed-form efficiency* (as defined in [4]). However for our purposes we had to extend the definition in [4] to handle larger classes of functionalities (for example we could not see how to use the definition in [4] to handle matrix multiplication). Moreover, we develop new such PRFs based on the Decisional Linear Assumption, that in particular can be used in groups with bilinear maps. Both our new definition and the new construction might be of independent interest.

Finally we discuss "multi-function" extensions of our protocols: this is the dual case in which the client stores a large (say ℓ) number of inputs with the server in advance, and then later queries different functions (F or M). Here the technical challenge is that the authentication information stored with the inputs must be "oblivious" to any function that will be applied later.

Applications: As discussed in [4], polynomial computations have a large number of applications, which – using our solutions – now have publicly verifiable proofs:

- *Proof of Retrievability:* The client stores a large file F with the server and later wants a short proof that the entire file can be retrieved. Using our solution, the client encodes the file as a polynomial $F(x)$ of degree d (each block representing a coefficient), and a proof of retrievability consists of the value $F(r)$ together with a proof of its correctness for a random input r provided by the client.
- *Verifiable Keyword Search:* Consider a text file $T = \{w_1, \ldots, w_\ell\}$ where w_i are the words contained in it. Using our solution, encode T as the polynomial $F(\cdot)$ of degree ℓ such that $F(w_i) = 0$.
- *Discrete Fourier Transform:* Both our polynomial and matrix protocols can be adapted to obtain a fast verifiable computation protocol for the DFT. The challenge here is that using the FFT algorithm such computation is already almost linear – $O(n \log n)$ – and therefore the verification time had to be minimal, i.e., $O(n)$ linear in the input size.
- *Linear Transformations:* Consider the general problem of applying a geometric transformation (such as the ones largely used in computer graphics) to large d-dimensional vectors. Since a linear transformation on a vector $\boldsymbol{x}$ can be expressed by a matrix M (of size $n \times d$) multiplying $\boldsymbol{x}$, a weak client can use our matrix protocol to outsource and verify this computation in the optimal time $O(n + d)$, i.e., linear in the input and the output size.

1.1 Related Work

The subject of verifiable outsourced computation has a large body of prior work. In the theoretical arena this basic question motivated the work on Interactive Proofs [2, 11], efficient arguments based

on probabilistically checkable proofs (PCP) [13, 14], CS Proofs [17] and the *muggles proofs* in [10]. However, in PCP-based schemes, the client must store the large data in order to verify the result, and therefore these solutions might not be applicable to our setting. In the past, more practical solutions, but of limited provable security, were also proposed: e.g., solutions based on audit (e.g. [18, 3]) or secure co-processors (e.g. [21, 22]) which "sign" the computation as correct, under the assumption that the adversary cannot tamper with the processor.

As mentioned above, our work follows the paradigm introduced in [8] which is also adopted in [7, 1]. The protocols described in those papers allow a client to outsource the computation of an arbitrary function (encoded as a Boolean circuit) and use fully homomorphic encryption (i.e. [9]) resulting in solutions of limited practical relevance.

We follow [4] by considering only a very limited class of computations in order to obtain better efficiency. In [4] the problem of practical protocols for verifiable computation of polynomials was first proposed, however their solution only offers private verifiability.

To the best of our knowledge, apart from the work of [20] described below, no other prior work in the literature discusses efficient delegation of matrix multiplication and the multi-function case.

1.2 Recent Works with Public Verification

Recently two works have considered public verification [19, 20]. Here we give a detailed comparison of their results with our solutions.

Papamanthou, Shi and Tamassia in [19] consider only the case of polynomial evaluation. Their solutions satisfy only a weaker "selective" notion of security, where the adversary must commit in advance to the input point x on which it is going to cheat (i.e. provide the client with an incorrect value $y' \neq F(x)$). Moreover, the security of their scheme relies on the d-SDH assumption which asymptotically depends on the degree of the polynomial. In contrast, the main advantage of our work is to achieve security in the strongest adaptive sense, while relying on a "constant" size assumption, which is independent of the size of the input. Finally, our solutions can handle a larger class of polynomial functions: their scheme supports polynomials in m variables and total degree d – which we also support – but we additionally consider also polynomials of degree d in each variable. For the case we both support, we enjoy a much faster verification protocol: a constant amount of work (*one* pairing computation and *one* exponentiation) while they require $O(m)$ pairings[3].

Parno, Raykova and Vaikuntanathan in [20] explore a connection between *Attribute-Based Encryption* and the problem of verifiable computation, showing that for any function for which an efficient ABE exists, we can construct an efficient publicly verifiable computation scheme. The class of functions for which we know an efficient ABE scheme is unfortunately very limited, and therefore the scheme in [20] does not work for any arbitrary poly-time computation. However, it does work for computations that can be expressed as poly-size Boolean Formulas, which in particular include polynomial evaluation and matrix multiplication. Compared to [20], our scheme has two major advantages: (i) when evaluating a polynomial/matrix over a field with p elements, the scheme in [20] incurs a multiplicative overhead of $O(\log p)$ in its complexity compared to ours as they must

[3] In contrast the delegation phase is basically free in their case, while our delegation step requires $O(md)$ work – note however that in publicly verifiable scheme, the verification algorithm might be run several times and therefore its efficiency is more important.

express the computation as a Boolean formula (i.e. bit by bit)[4]; (ii) the result published in [20] is also secure only in the selective model, while we achieve full security.

The selective security limitation in [20] is inherited from the specific ABE scheme used in their protocol (the one in [12]). Recently, a new ABE scheme has been presented that removes the selective security issue [15], and therefore when combined with [20] should yield a fully secure verifiable computation solution. The efficiency of this new scheme is no better than the original one in [20], therefore our $O(\log p)$ efficiency advantage would stand even compared to this fully secure solution. Moreover, the new ABE scheme in [15] also requires a "non-constant-size" computational assumption that depends on the size of the Boolean Formula (i.e., of the polynomial or of the matrix, when instantiated in [20]). Again our protocol only requires constant-size assumptions.

In Table 1 we summarize a "features comparison" between our scheme and these two recent schemes (and also [4]).

Schemes	Properties		
	Public Verif.	Full Security	Constant Assumption
Benabbas et al. [4]	×	✓	✓
Papamanthou et al. [19]	✓	×	×
Parno et al. [20] + Goyal et al.[12]	✓	×	✓
Parno et al. [20] + Lewko et al.[15]	✓	✓	×
This work	✓	✓	✓

Table 1. Comparisons with related work.

1.3 An overview of our solutions

Polynomial Evaluation. Our starting point is the protocol of [4]: assume the client has a polynomial $F(\cdot)$ of large degree d, and it wants to compute the value $F(x)$ for arbitrary inputs x. In [4] the client stores the polynomial in the clear with the server as a vector c_i of coefficients in $\mathbb{Z}_p$. The client also stores with the server a vector t_i of group elements of the form $g^{ac_i+r_i}$ where g generates a cyclic group $\mathbb{G}$ of order p, $a \in_R \mathbb{Z}_p$, and r_i is the i^{th}-coefficient of a polynomial $R(\cdot)$ of the same degree as $F(\cdot)$. When queried on input x, the server returns $y = F(x)$ and $t = g^{aF(x)+R(x)}$, and the client accepts y iff $t = g^{ay+R(x)}$.

If $R(\cdot)$ was a random polynomial, then this is a secure way to authenticate y, however checking that $t = g^{ay+R(x)}$ would require the client to compute $R(x)$ – the exact work that we set out to avoid! The crucial point, therefore, is how to perform this verification fast, i.e., in $o(d)$ time. The fundamental tool in [4] is the introduction of pseudo-random functions (PRFs) with a special property called *closed-form efficiency*: if we define the coefficients r_i of $R(\cdot)$ as $PRF_K(i)$ (which preserves the security of the scheme), then for any input x the value $g^{R(x)}$ can be computed very efficiently (sub-linearly in d) by a party who knows the secret key K for the PRF.

[4] One should remark that in our scheme the field size p is the same p as the order of the underlying bilinear groups that we use to cryptographically prove security, therefore our solution cannot handle small field sizes. In contrast, [20] supports polynomials over any field, in particular $\mathbb{Z}_2$.

Note, however, that this approach implies a private verification algorithm by the same client who outsourced the polynomial in the first place, since it requires knowledge of the secret key K. To make this verification public we extended these techniques as follows. First we use a cyclic group $\mathbb{G}$ that admits a bilinear map $e(\cdot,\cdot)$ from $\mathbb{G}\times\mathbb{G}$ to a target group $\mathbb{G}_T$. Informally, (the actual protocol is slightly more complicated), when the client sends x to the server it also computes a public verification key for x, $A = e(g,g)^a$ and $\mathsf{VK}_x = e(g,g)^{R(x)}$ – note that this step will take $o(d)$ time thanks to the closed-form efficiency of the PRF. When the server returns y, t, anybody who knows[5] the correct A, VK_x can verify it by checking that $e(t,g) = A^y \cdot \mathsf{VK}_x$.

A technical complication is that the PRFs in [4] are based on the hardness of the Decisional Diffie-Hellman problem which however is easy on groups that admit bilinear maps. The first solution we propose to this problem is to instantiate our protocol in asymmetric bilinear groups, and assume that these PRFs are secure based on the External Diffie-Hellman Assumption. More interestingly, however, we also present new solutions that rely on the much weaker Decision Linear Assumption over bilinear groups, by devising a closed-form efficient variants of the Lewko-Waters PRF [16]. This result can be of independent interest.

As in [4], we have solutions not just for single-variable polynomials of degree d, but also for multivariate polynomials of degree d in each variable and of total degree d.

Matrix Multiplication. The client stores a $n\times d$ matrix M with the server and wants to compute the value $\boldsymbol{y} = M\cdot\boldsymbol{x}$ for a d-dimensional vector $\boldsymbol{x}$; the goal is to verify y in $O(n+d)$. Our solution also uses the concept of closed-form efficient PRFs: the client stores the matrix in the clear together with another matrix W whose elements are group elements of the form $W_{i,j} = g^{am_{i,j}+r_{i,j}}$, where $a \in_R \mathbb{Z}_p$, and $r_{i,j} = PRF_K(i,j)$ defines a $n \times d$ pseudo-random matrix R. We propose new PRFs with closed-form efficiency for matrix-vector multiplication, i.e., such that the vector $g^{R\cdot\boldsymbol{x}}$ can be computed in time $O(n+d)$ by somebody who knows the key K. The verification then proceeds as in the polynomial case. The server returns $\boldsymbol{y} = M\cdot\boldsymbol{x}$ and $\boldsymbol{t} = W\cdot\boldsymbol{x}$ (computed in the exponent) and private verification can be obtained by having the client check the vector of equations $\boldsymbol{t} = g^{a\boldsymbol{y}+R\boldsymbol{x}}$ using the closed form efficiency for computing $g^{R\boldsymbol{x}}$ efficiently. Public verification can be obtained by having the client publish $A = e(g,g)^a$ and the vector $\mathsf{VK}_x = e(g,g)^{R\cdot x}$ (again computed fast using closed-form efficiency of the PRF), and then anybody can verify the following vector equation $e(\boldsymbol{t},g) = A^{\boldsymbol{y}}\cdot\mathsf{VK}_x$ (component-wise).

1.4 Paper Organization

In Section 2 we recall the security definitions and the computational assumptions needed by our protocols. In Section 3 we present our new PRFs with closed form efficiency based on the Decision Linear Assumption, which are the basic tools used by our protocols that are described in Section 4. The multi-function extension and the DFT application are discussed in the Appendix.

2 Background and Definitions

In what follows we will denote with $\lambda \in \mathbb{N}$ a security parameter. We say that a function ϵ is negligible if it vanishes faster than the inverse of any polynomial. If S is a set, we denote with

[5] Here we can assume that the client has a way of reliably publishing the values A, VK_x. One possible way is for the client to sign them and give them to the server together with x in the input submission state. The server then will have to include the signed A, VK_x in the "correctness proof" and the verification algorithm must also check the validity of the client's signature.

$x \xleftarrow{\$} S$ the process of selecting x uniformly at random in S. Let $\mathcal{A}$ be a probabilistic algorithm. We denote with $x \xleftarrow{\$} \mathcal{A}(\cdot)$ the process of running $\mathcal{A}$ on some appropriate input and assigning its output to x.

2.1 Computational Assumptions

The co-Computational Diffie-Hellman problem was introduced by Boneh, Lynn and Shacham as a natural generalization of the Computational Diffie-Hellman problem in asymmetric bilinear groups [6]. It is defined as follows.

Definition 1 (co-CDH). *Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be groups of prime order p, so that $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map. Let $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ be generators, and $a, b \xleftarrow{\$} \mathbb{Z}_p$ be chosen at random. We define the advantage of an adversary $\mathcal{A}$ in solving the* co-Computational Diffie-Hellman *problem as*

$$\mathbf{Adv}^{cdh}_{\mathcal{A}}(\lambda) = \Pr[\mathcal{A}(p, g_1, g_2, g_1^a, g_2^b) = g_1^{ab}]$$

We say that the co-CDH Assumption ϵ-holds in $\mathbb{G}_1, \mathbb{G}_2$ if for every PPT algorithm $\mathcal{A}$ we have that $\mathbf{Adv}^{cdh}_{\mathcal{A}}(\lambda) \leq \epsilon$.

Notice that in symmetric bilinear groups, where $\mathbb{G}_1 = \mathbb{G}_2$, this problem reduces to standard CDH. For asymmetric groups, it is also easy to see that co-CDH reduces to the computational Bilinear Diffie-Hellman problem [5].

As it is well known the *decisional* version of the CDH Assumption (where the adversary cannot distinguish g_1^{ab} from a random value) is easy in symmetric bilinear groups where $\mathbb{G}_1 = \mathbb{G}_2$. However in the case of *asymmetric* bilinear groups $\mathbb{G}_1, \mathbb{G}_2$ where $\mathbb{G}_1 \neq \mathbb{G}_2$, then the DDH problem may still be hard in $\mathbb{G}_1$. This is called External Diffie-Hellman (XDH) assumption stated below.

Definition 2 (XDH). *Let $g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be as in Def. [1]. We define the advantage $\mathbf{Adv}^{xdh}_{\mathcal{A}}(\lambda)$ of an adversary $\mathcal{A}$ in deciding the* External Diffie-Hellman *(XDH) problem as*

$$|\Pr[\mathcal{A}(p, g_1, g_2, g_1^a, g_2^b, g_1^{ab}) = 1] - \Pr[\mathcal{A}(p, g, g_1^a, g_1^b, g_1^c) = 1]|$$

where $a, b, c \xleftarrow{\$} \mathbb{Z}_p$. We say that the XDH Assumption ϵ-holds over $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ if for every PPT algorithm $\mathcal{A}$ we have that $\mathbf{Adv}^{ddh}_{\mathcal{A}}(\lambda) \leq \epsilon$.

A weaker assumption than XDH which we are going to use in our protocols is the Decision Linear Assumption.

Definition 3 (Decision Linear). *Let $\mathbb{G}$ be a group of prime order p, $g_0, g_1, g_2 \xleftarrow{\$} \mathbb{G}$, and $r_0, r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$. We define the advantage of an adversary $\mathcal{A}$ in deciding the* Linear *problem in $\mathbb{G}$ as*

$$\begin{aligned}\mathbf{Adv}^{dlin}_{\mathcal{A}}(\lambda) = |\Pr[\mathcal{A}(p, g_0, g_1, g_2, g_1^{r_1}, g_2^{r_2}, g_0^{r_1+r_2}) = 1] - \\ \Pr[\mathcal{A}(p, g_0, g_1, g_2, g_1^{r_1}, g_2^{r_2}, g_0^{r_0}) = 1]|\end{aligned}$$

We say that the Decision Linear Assumption ϵ-holds in $\mathbb{G}$ if for every PPT algorithm $\mathcal{A}$ we have $\mathbf{Adv}^{cdh}_{\mathcal{A}}(\lambda) \leq \epsilon$.

We note that we do not know of an efficient way to solve the Linear Problem even if the group $\mathbb{G}$ admits an efficiently computable bilinear map.

2.2 Verifiable Computation

A verifiable computation scheme is a tuple of distributed algorithms that enable a client to outsource the computation of a function f to an untrusted worker, in such a way that the client can verify the correctness of the result returned by the worker. In order for the outsourcing to make sense, it is crucial that the cost of verification at the client must be cheaper than computing the function locally.

In our work we are interested in computation schemes that are *publicly verifiable* as defined by Parno *et al.* [20]: any third party (possibly different from the delegator) can verify the correctness of the results returned by the worker.

Let $\mathcal{F}$ be a family of functions. A Verifiable Computation scheme $\mathcal{VC}$ for $\mathcal{F}$ is defined by the following algorithms:

$\mathsf{KeyGen}(1^\lambda, f) \to (\mathsf{SK}_f, \mathsf{PK}_f, \mathsf{EK}_f)$**:** on input a function $f \in \mathcal{F}$, it produces a secret key SK_f that will be used for input delegation, a public verification key PK_f, used to verify the correctness of the delegated computation, and a public evaluation key EK_f which will be handed to the server to delegate the computation of f.

$\mathsf{ProbGen}(\mathsf{PK}_f, \mathsf{SK}_f, x) \to (\sigma_x, \mathsf{VK}_x)$**:** given a value $x \in \mathsf{Dom}(f)$, the problem generation algorithm is run by the delegator to produce an encoding σ_x of x, together with a public verification key VK_x.

$\mathsf{Compute}(\mathsf{EK}_f, \sigma_x) \to \sigma_y$**:** given the evaluation key EK_f and the encoding σ_x of an input x, this algorithm is run by the worker to compute an encoded version of $y = f(x)$.

$\mathsf{Verify}(\mathsf{PK}_f, \mathsf{VK}_x, \sigma_y) \to y \cup \perp$**:** on input the public key PK_f, the verification key VK_x, and an encoded output σ_y, this algorithm returns a value y or an error $\perp$.

Correctness. Informally, a verifiable computation scheme $\mathcal{VC}$ is *correct* if the values generated by the problem generation algorithm allows a honest worker to output values that will verify correctly. More formally, for any $f \in \mathcal{F}$, any $(\mathsf{SK}_f, \mathsf{PK}_f, \mathsf{EK}_f) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, f)$, any $x \in \mathsf{Dom}(f)$, if $(\sigma_x, \mathsf{VK}_x) \xleftarrow{\$} \mathsf{ProbGen}(\mathsf{PK}_f, \mathsf{SK}_f, x)$ and $\sigma_y \leftarrow \mathsf{Compute}(\mathsf{EK}_f, \sigma_x)$, then $f(x) \leftarrow \mathsf{Verify}(\mathsf{PK}_f, \mathsf{VK}_x, \sigma_y)$ holds with all but negligible probability.

Security. For any verifiable computation scheme $\mathcal{VC}$, let us define the following experiment:

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{PubVer}}[\mathcal{VC}, f, \lambda]$

$(\mathsf{SK}_f, \mathsf{PK}_f, \mathsf{EK}_f) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, f)$

For $i = 1$ to q:

$x_i \leftarrow \mathcal{A}(\mathsf{PK}_f, \mathsf{EK}_f, \sigma_{x,1}, \mathsf{VK}_{x,1}, \ldots, \sigma_{x,i-1}, \mathsf{VK}_{x,i-1})$

$(\sigma_{x,i}, \mathsf{VK}_{x,i}) \xleftarrow{\$} \mathsf{ProbGen}(\mathsf{SK}_f, x_i)$

$x^* \leftarrow \mathcal{A}(\mathsf{PK}_f, \mathsf{EK}_f, \sigma_{x,1}, \mathsf{VK}_{x,1}, \ldots, \sigma_{x,q}, \mathsf{VK}_{x,q})$

$(\sigma_{x^*}, \mathsf{VK}_{x^*}) \xleftarrow{\$} \mathsf{ProbGen}(\mathsf{SK}_f, x^*)$

$\hat{\sigma}_y \leftarrow \mathcal{A}(\mathsf{PK}_f, \mathsf{EK}_f, \sigma_{x,1}, \mathsf{VK}_{x,1}, \ldots, \sigma_{x,q}, \mathsf{VK}_{x,q}, \mathsf{VK}_{x^*})$

$\hat{y} \leftarrow \mathsf{Verify}(\mathsf{PK}_f, vk_{x^*}, \hat{\sigma}_y)$

If $\hat{y} \neq \perp$ and $\hat{y} \neq f(x^*)$, output 1, else output 0.

For any $\lambda \in \mathbb{N}$, any function $f \in \mathcal{F}$, we define the advantage of an adversary $\mathcal{A}$ making at most $q = poly(\lambda)$ queries in the above experiment against $\mathcal{VC}$ as

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{PubVer}}(\mathcal{VC}, f, q, \lambda) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{PubVer}}[\mathcal{VC}, f, \lambda] = 1].$$

Definition 4. *A verifiable computation scheme $\mathcal{VC}$ is secure for $\mathcal{F}$ if for any $f \in \mathcal{F}$, and any PPT $\mathcal{A}$ it holds that $\mathbf{Adv}^{\mathsf{PubVer}}_{\mathcal{A}}(\mathcal{VC}, f, q, \lambda)$ is negligible.*

Note that our definition captures full adaptive security, where the adversary decides "on the fly" on which input x^* it will try to cheat. The weaker selective security notion achieved in [19, 20] requires the adversary to commit to x^* at the beginning of the game.

3 Closed Form Efficient PRF

The notion of closed form efficient pseudorandom functions was introduced in [4]. Their definition however seemed geared specifically towards the application of polynomial evaluation and therefore proved insufficient for our matrix multiplication protocol. Here we extend it to include any computations run on a set of pseudo-random values and a set of arbitrary inputs.

A closed form efficient PRF consists of algorithms $(\mathsf{PRF.KG}, \mathsf{F})$. The key generation $\mathsf{PRF.KG}$ takes as input the security parameter 1^λ, and outputs a secret key K and some public parameters pp that specify domain $\mathcal{X}$ and range $\mathcal{Y}$ of the function. On input $x \in \mathcal{X}$, $\mathsf{F}_K(x)$ uses the secret key K to compute a value $y \in \mathcal{Y}$. It must of course satisfy the usual pseudorandomness property. Namely, $(\mathsf{PRF.KG}, \mathsf{F})$ is ϵ-secure if for every PPT adversary $\mathcal{A}$ it holds:

$$\left|\Pr[\mathcal{A}^{\mathsf{F}_K(\cdot)}(1^\lambda, \mathsf{pp}) = 1] - \Pr[\mathcal{A}^{R(\cdot)}(1^\lambda, \mathsf{pp}) = 1]\right| \leq \epsilon$$

where $(K, \mathsf{pp}) \xleftarrow{\$} \mathsf{PRF.KG}(1^\lambda)$, and $R(\cdot)$ is a random function from $\mathcal{X}$ to $\mathcal{Y}$.

In addition, it is required to satisfy the following *closed-form efficiency* property. Consider an arbitrary computation Comp that takes as input ℓ random values $R_1, \ldots, R_\ell \in \mathcal{Y}$ and m arbitrary values $\boldsymbol{x} = (x_1, \ldots, x_m)$, and assume that the best algorithm to compute $\mathsf{Comp}(R_1, \ldots, R_\ell, x_1, \ldots, x_m)$ takes time T. Let $z = (z_1, \ldots, z_\ell)$ a ℓ-tuple of arbitrary values in the domain $\mathcal{X}$ of F. We say that a PRF $(\mathsf{PRF.KG}, \mathsf{F})$ is *closed-form efficient* for (Comp, z) if there exists an algorithm $\mathsf{PRF.CFEval}_{\mathsf{Comp},z}$ such that

$$\begin{aligned}&\mathsf{PRF.CFEval}_{\mathsf{Comp},z}(K, x) = \\ &\quad \mathsf{Comp}(F_K(z_1), \ldots, F_K(z_\ell), x_1, \ldots, x_m)\end{aligned}$$

and its running time is $o(T)$. For $z = (1, \ldots, \ell)$ we usually omit the subscript z.

Note that depending on the structure of Comp, this property may enforce some constraints on the range $\mathcal{Y}$ of F. In particular in our case, $\mathcal{Y}$ will be an abelian group. We also remark that due to the pseudorandomness property the output distribution of $\mathsf{PRF.CFEval}_{\mathsf{Comp},z}(K, x)$ (over the random choice of K) is indistinguishable from the output distribution of $\mathsf{Comp}(R_1, \ldots, R_\ell, x_1, \ldots, x_m)$ (over the random choices of the R_i).

3.1 Closed-Form Efficient PRFs from Decision Linear

In this section we show constructions of pseudorandom functions that enjoy closed-form efficiency for multivariate polynomials, and matrix multiplication. Their security is based on the Decision Linear assumption.

Polynomials of degree d in each variable As our first construction, we show that the PRF of Lewko and Waters [16] has closed form efficiency for polynomials in m variables and degree at most d in each variable.

First, we recall the construction $\mathsf{PRF}_{\mathsf{LW}}$ of this PRF.

$\mathsf{PRF.KG}(1^\lambda, s, m)$**.** Generate a group description $(p, g, \mathbb{G}) \xleftarrow{\$} \mathcal{G}(1^\lambda)$. Choose $4ms + 2$ values

$$y_0, z_0, \{y_{i,j}, z_{i,j}, w_{i,j}, v_{i,j}\}_{1 \le i \le m,\, 1 \le j \le s} \xleftarrow{\$} \mathbb{Z}_p$$

Output $K = (y_0, z_0, \{y_{i,j}, z_{i,j}, w_{i,j}, v_{i,j}\}_{i,j})$.

$\mathsf{F}_K(i)$**.** The domain of the function is $i = (i_1, \ldots, i_m) \in [0..d]^m$, but we interpret each $i_j = (i_{j,1}, \ldots, i_{j,s})$ as a binary string of $s = \log d$ bits. The function is computed by the following algorithm:

Initialize $a \leftarrow y_0, b \leftarrow z_0$
For $j = 1$ to m:
  For $k = 1$ to s:
    If $i_{j,k} = 0$, then $a \leftarrow a, b \leftarrow b$.
    Else, $a \leftarrow a y_{j,k} + b z_{j,k}$, $b \leftarrow a w_{j,k} + b v_{j,k}$
Output g^a

Except for a few changes in the notation, the function above is the same as the one in [16]. Therefore, its security follows from the following theorem.

Theorem 1 ([16]). *If the Decision Linear assumption holds for $\mathcal{G}$, then $\mathsf{PRF}_{\mathsf{LW}}$ is a pseudorandom function.*

In what follows we show that $\mathsf{PRF}_{\mathsf{LW}}$ admits closed form efficiency for polynomials.

Consider any polynomial $p(x_1, \ldots, x_m)$ in m variables of degree d in each variable. This polynomial has up to $l = (d+1)^m$ terms which we can index with $(i_1, \ldots, i_m)$ with each $0 \le i_j \le d$. We want to compute

$$\mathsf{Poly}(\{R_{(i_1,\ldots,i_m)}\}_{0 \le i_1,\ldots,i_m \le d}, x_1, \ldots, x_m) =$$

$$\prod_{0 \le i_1,\ldots,i_m \le d} R_{(i_1,\ldots,i_m)}^{(x_1^{i_1} \cdots x_m^{i_m})} = g^{p(x_1,\ldots,x_m)}$$

where $p(\cdot)$ is the polynomial whose coefficients are the discrete logs of the R values. We now show that if we set $R_{(i_1,\ldots,i_m)} = F_K(i_1, \ldots, i_m)$, then there exists an algorithm $\mathsf{PRF.CFEval}_{\mathsf{Poly}}(K, x_1, \ldots, x_m)$ that can compute

$$g^{p(x_1,\ldots,x_m)} = \prod_{0 \le i_1,\ldots,i_m \le d} \mathsf{F}_K(i_1, \ldots, i_m)^{x_1^{i_1} \cdots x_m^{i_m}}$$

in time $O(m \log d)$, instead of the regular computation running in time $O(d^m \cdot m \cdot \log d)$.

For ease of exposition, we first describe an alternative equivalent algorithm for computing $\mathsf{F}_K(i_1, \ldots, i_m)$. Denote by $f_K(i) = (f_K^1(i), f_K^2(i))$ the following recursive function:

If $i = 0$, then $f_K^1(0) = y_0$ and $f_K^2(0) = z_0$.
If $i > 0$, then: let $i = (i_1, \ldots, i_m)$
  let $\bar{m}$ be such that $i_{\bar{m}+1} = \cdots = i_m = 0$ and $i_{\bar{m}} \neq 0$.
  let $i_{\bar{m}} = 2^{j_{\bar{m}} + \ell_{\bar{m}}}$ where $j_{\bar{m}} = \lfloor \log i_{\bar{m}} \rfloor$, $0 \le \ell_{\bar{m}} \le 2^{j_{\bar{m}}} - 1$

$$f^1_K(i_1,\ldots,i_m) = f^1_K(i_1,\ldots,i_{\bar{m}-1},\ell_{\bar{m}},0,\ldots,0)y_{\bar{m},j_{\bar{m}}+1} + f^2_K(i_1,\ldots,i_{\bar{m}-1},\ell_{\bar{m}},0,\ldots,0)z_{\bar{m},j_{\bar{m}}+1}$$
$$f^2_K(i_1,\ldots,i_m) = f^1_K(i_1,\ldots,i_{\bar{m}-1},\ell_{\bar{m}},0,\ldots,0)w_{\bar{m},j_{\bar{m}}+1} + f^2_K(i_1,\ldots,i_{\bar{m}-1},\ell_{\bar{m}},0,\ldots,0)v_{\bar{m},j_{\bar{m}}+1}$$

Finally, the value of the function is $\mathsf{F}_K(i) = g^{f^1_K(i)}$.

Using this notation, we can now write

$$p(x_1,\ldots,x_m) = \sum_{0\le i_1,\ldots,i_m\le d} f^1_K(i_1,\ldots,i_m)x_1^{i_1}\cdots x_m^{i_m}$$

$\mathsf{PRF.CFEval}_{\mathsf{Poly}}(K,x_1,\ldots,x_m)$.
For all $j=1,\ldots,m$, let $s_j = \lceil \log d \rceil$, and $p_{s_1,\ldots,s_m}(x_1,\ldots,x_m) = (p^1_{s_1,\ldots,s_m}(x_1,\ldots,x_m), p^2_{s_1,\ldots,s_m}(x_1,\ldots,x_m))$ be the following recursive function:

If $s_1 = s_2 = \cdots = s_m = 1$, then:
$p^1_{s_1,\ldots,s_m}(x_1,\ldots,x_m) = y_0$, $p^2_{s_1,\ldots,s_m}(x_1,\ldots,x_m) = z_0$.
Else
let $\bar{m}$ be such that $s_{\bar{m}+1} = \cdots = s_m = 1$ and $s_{\bar{m}} > 1$.
$$p^1_{s_1,\ldots,s_m}(x_1,\ldots,x_m) = p^1_{s_1,\ldots,s_{\bar{m}}-1,0,\ldots,0}(x_1,\ldots,x_m) + x^{2^{s_{\bar{m}}-1}}\left(p^1_{s_1,\ldots,s_{\bar{m}}-1,0,\ldots,0}(x_1,\ldots,x_m)y_{\bar{m},s_{\bar{m}}} + p^2_{s_1,\ldots,s_{\bar{m}}-1,0,\ldots,0}(x_1,\ldots,x_m)z_{\bar{m},s_{\bar{m}}}\right)$$
$$p^2_{s_1,\ldots,s_m}(x_1,\ldots,x_m) = p^2_{s_1,\ldots,s_{\bar{m}}-1,0,\ldots,0}(x_1,\ldots,x_m) + x^{2^{s_{\bar{m}}-1}}\left(p^1_{s_1,\ldots,s_{\bar{m}}-1,0,\ldots,0}(x_1,\ldots,x_m)w_{\bar{m},s_{\bar{m}}} + p^2_{s_1,\ldots,s_{\bar{m}}-1,0,\ldots,0}(x_1,\ldots,x_m)v_{\bar{m},s_{\bar{m}}}\right)$$

Finally, the algorithm outputs $g^{p(x_1,\ldots,x_m)} = g^{p^1_{s_1,\ldots,s_m}(x_1,\ldots,x_m)}$.

We prove correctness of $\mathsf{PRF.CFEval}_{\mathsf{Poly}}(K,x_1,\ldots,x_m)$ by induction on $s_1,\ldots,s_m$. If $s_1 = \cdots = s_m = 1$, then the algorithm is clearly correct. Without loss of generality, assume that $s_m > 1$. If the algorithm is correct for $s_1,\ldots,s_m-1$, i.e., $p^1_{s_1,\ldots,s_m-1}(x_1,\ldots,x_m)$ is

$$\sum_{i_1,\ldots,i_{m-1}\le d, i_m=0}^{2^{s_m-1}-1} f^1_K(i_1,\ldots,i_m)x_1^{i_1}\cdots x_m^{i_m}$$

and $p^2_{s_1,\ldots,s_m-1}(x_1,\ldots,x_m)$ is

$$\sum_{i_1,\ldots,i_{m-1}\le d, i_m=0}^{2^{s_m-1}-1} f^2_K(i_1,\ldots,i_m)x_1^{i_1}\cdots x_m^{i_m}$$

Then we show correctness for $s_1,\ldots,s_m$.

First, by definition of our algorithm we have:

$$p^1_{s_1,\ldots,s_m}(x_1,\ldots,x_m) = p^1_{s_1,\ldots,s_m-1}(x_1,\ldots,x_m) + x^{2^{s_m-1}}\cdot \left(p^1_{s_1,\ldots,s_m-1}(x_1,\ldots,x_m)y_{m,s_m} + p^2_{s_1,\ldots,s_m-1}(x_1,\ldots,x_m)z_{m,s_m}\right)$$

If we then apply our inductive assumption we obtain:

$$\sum_{i_1,\ldots,i_{m-1}\leq d, i_m=0}^{2^{s_m-1}-1} f_K^1(i_1,\ldots,i_m)x_1^{i_1}\cdots x_m^{i_m} + x^{2^{s_m-1}} \cdot \left(\sum_{i_1,\ldots,i_{m-1}\leq d, i_m=0}^{2^{s_m-1}-1} (f_K^1(i_1,\ldots,i_m)y_{m,s_m} + f_K^2(i_1,\ldots,i_m)z_{m,s_m})x_1^{i_1}\cdots x_m^{i_m}\right)$$

Next, we can apply the definition of $f_K(i_1,\ldots,i_m)$:

$$\sum_{i_1,\ldots,i_{m-1}\leq d, i_m=0}^{2^{s_m-1}-1} f_K^1(i_1,\ldots,i_m)x_1^{i_1}\cdots x_m^{i_m} + \left(\sum_{i_1,\ldots,i_{m-1}\leq d, i_m=0}^{2^{s_m-1}-1} f_K^1(i_1,\ldots,i_m+2^{s_m-1})x_1^{i_1}\cdots x_m^{i_m+2^{s_m-1}}\right)$$

Finally, by simple rewriting this equation, we obtain:

$$\sum_{i_1,\ldots,i_{m-1}\leq d, i_m=0}^{2^{s_m-1}-1} f_K^1(i_1,\ldots,i_m)x_1^{i_1}\cdots x_m^{i_m} + \left(\sum_{i_1,\ldots,i_{m-1}\leq d, i_m=2^{s_m-1}}^{2^{s_m}-1} f_K^1(i_1,\ldots,i_m)x_1^{i_1}\cdots x_m^{i_m}\right) = \sum_{i_1,\ldots,i_{m-1}\leq d, i_m=0}^{2^{s_m}-1} f_K^1(i_1,\ldots,i_m)x_1^{i_1}\cdots x_m^{i_m}$$

Analogously, one can show correctness of $p^2_{s_1,\ldots,s_m}(x_1,\ldots,x_m)$.

The algorithm makes only one recursive call at each step, and thus it runs in time $O(m\log d)$.

Polynomials of degree d in each monomial We show a variation of the PRF by Lewko and Waters, described in the previous section, that achieves closed form efficiency for polynomials in m variables and degree d *in each monomial*. The previous construction $\mathsf{PRF}_{\mathsf{LW}}$ is more general as it can support any polynomials where d is an upper bound of each variable's degree. However, $\mathsf{PRF}_{\mathsf{LW}}$ is tailored to that type of polynomials that have $(d+1)^m$ terms. In contrast, m-variate polynomials of total degree d have $\binom{m+d}{d} = O(m^d)$ terms. These two quantities seem incomparable as they crucially depend on the size of m vs. d. In particular, the $\mathsf{PRF}_{\mathsf{LW}}$ construction may not be suitable for this case. So, here we propose another variant, which offers closed form efficiency for m-variate polynomials of total degree at most d.

Our construction $\mathsf{PRF}_{\mathsf{eLW}}$ works as follows.

$\mathsf{PRF.KG}(1^\lambda, d, m)$**.** Generate a group description $(p, g, \mathbb{G}) \xleftarrow{\$} \mathcal{G}(1^\lambda)$. Choose $4(m+1)d+2$ values

$$y_0, z_0, \{y_{i,j}, z_{i,j}, w_{i,j}, v_{i,j}\}_{1\leq i\leq d,\ 0\leq j\leq m} \xleftarrow{\$} \mathbb{Z}_p$$

Output $K = (y_0, z_0, \{y_{i,j}, z_{i,j}, w_{i,j}, v_{i,j}\}_{i,j})$.

$\mathsf{F}_K(i)$**.** The domain of the function is $i = (i_1, \ldots, i_d) \in [0..m]^d$, and we interpret each i_j as an integer in $[0..m]$. The function $\mathsf{F}_K(i_1, \ldots, i_d)$ is computed by the following algorithm:

Initialize $a \leftarrow y_0, b \leftarrow z_0$
For $j = 1$ to d:
$a \leftarrow a y_{j,i_j} + b z_{j,i_j}$, $b \leftarrow a w_{j,i_j} + b v_{j,i_j}$
Output g^a

This function can be seen as an extension of $\mathsf{PRF}_{\mathsf{LW}}$ as follows. First, each i_j is in $[0..m]$ instead of $[0..1]$. Second, the algorithm instead of "copying" a and b when $i_j = 0$, it always makes a new linear combination.

Theorem 2. *If the Decision Linear assumption holds for $\mathcal{G}$, then $\mathsf{PRF}_{\mathsf{eLW}}$ is a pseudorandom function.*

Proof. The proof of this theorem can be obtained by adapting the proof of the Lewko-Waters PRF [16]. We sketch below how to adapt that proof to our case.

Even for our PRF, it is possible to define an inefficient evaluation algorithm:

Initialize $A \leftarrow g^{y_0}, B \leftarrow g^{z_0}$
For $j = 1$ to d:
$A \leftarrow A^{y_{j,i_j}} \cdot B^{z_{j,i_j}}$, $B \leftarrow A^{w_{j,i_j}} \cdot B^{v_{j,i_j}}$
Output A

We observe that the core of our construction is the following pseudorandom generator:

$$G_{\{y_j, z_j, w_j, v_j\}_{0 \le j \le m}}(A, B) =$$
$$(A^{y_0} \cdot B^{z_0}, \ldots, A^{y_m} \cdot B^{z_m}, A^{w_0} \cdot B^{v_0}, \ldots, A^{w_m} \cdot B^{v_m})$$

We define $d + 1$ hybrid games $Game_0, \ldots, Game_d$ as follows. In $Game_k$ the challenger chooses only exponents

$$\{y_{k+1,j}, z_{k+1,j}, w_{k+1,j}, v_{k+1,j}, \ldots, y_{d,j}, z_{d,j}, w_{d,j}, v_{d,j}\}_{0 \le jm}$$

and answers queries for input $i = (i_1, \ldots, i_d)$ as follows. A and B are output of a random function on the first k components $(i_1, \ldots, i_k)$. Next, the above iterative algorithm is applied from $j = k+1$ to d.

It is possible to show that any adversary $\mathcal{A}$ causing a noticeable difference between $Game_k$ and $Game_{k+1}$, can be reduced to an adversary $\mathcal{B}$ against the above PRG. In particular, $\mathcal{B}$ receives q different instances of the PRG, one for each of the possible q different queries on the first k components. For each of these queries, $\mathcal{B}$ proceeds as follows. Let

$$(C_0, \ldots, C_m, D_0, \ldots, D_m)$$

be the output of the PRG instance for query $(i_1, \ldots, i_k)$, where each C_j, D_j are either pseudorandom $(A^{y_j} \cdot B^{z_j}, A^{w_j} \cdot B^{v_j})$ or random. $\mathcal{B}$ simulates the iterative algorithm on $(i_1, \ldots, i_d)$ by first picking $C_{i_{k+1}}, D_{i_{k+1}}$ from the appropriate PRG instance, and then it applies the natural algorithm to $A = C_{i_{k+1}}, B = D_{i_{k+1}}$, from $k + 2$ to d.

The final step of the proof is to show that the above PRG is secure. This can be done by using Lemma 1 in [16] to create many random independent instances of the Decision Linear problem. Precisely, this requires $2m$ hybrid steps to slowly change the distribution of each C_j, D_j from pseudorandom to random. □

In what follows we show that $\mathsf{PRF}_{\mathsf{eLW}}$ admits closed form efficiency for polynomials $p(x_1, \ldots, x_m)$ in m variables and total degree at most d. Such polynomials have $\binom{m+d}{d}$ terms, but for our purposes we use slightly more values to define the coefficients: $R_1, \ldots, R_l$, for $l = (m+1)^d$. We interpret each $1 \leq i \leq l$ as $(i_1, \ldots, i_d)$, with each $0 \leq i_j \leq m$. We want to compute

$$\mathsf{Poly}(\{R_{(i_1,\ldots,i_d)}\}_{0 \leq i_1,\ldots,i_d \leq m}, x_1, \ldots, x_m) =$$

$$\prod_{0 \leq i_1,\ldots,i_d \leq m} R_{(i_1,\ldots,i_d)}^{\prod_{j=1}^{d} x_{i_j}} = g^{p(x_1,\ldots,x_m)}$$

We now show that if we set $R_{(i_1,\ldots,i_d)} = F_K(i_1, \ldots, i_d)$ (and we denote $x_0 = 1$), then there exists an algorithm $\mathsf{PRF.CFEval}_{\mathsf{Poly}}(K, x_1, \ldots, x_m)$ that can compute

$$g^{p(x_1,\ldots,x_m)} = \prod_{0 \leq i_1,\ldots,i_d \leq m} F_K(i_1, \ldots, i_d)^{\prod_{j=1}^{d} x_{i_j}}$$

in time $O(md)$, instead of the regular computation running in time $O(dm^d)$.

For ease of exposition, we first describe an alternative equivalent algorithm for computing $\mathsf{F}_K(i_1, \ldots, i_m)$. Denote by $f_K(i) = (f_K^1(i), f_K^2(i))$ the following recursive function:

Let $i = (i_1, \ldots, i_d)$
If $i_1 = \cdots = i_d = *$, then
$f_K^1(*, \ldots, *) = y_0$ and $f_K^2(*, \ldots, *) = z_0$.
Else:
let $\bar{d}$ be such that $i_{\bar{d}+1} = \cdots = i_d = *$ and $i_{\bar{d}} \neq *$.
$f_K^1(i_1, \ldots, i_{\bar{d}}, *, \ldots, *) = f_K^1(i_1, \ldots, i_{\bar{d}-1}, *, \ldots, *) y_{\bar{d}, j_{\bar{d}}} +$
$\quad f_K^2(i_1, \ldots, i_{\bar{d}-1}, *, \ldots, *) z_{\bar{d}, j_{\bar{d}}}$
$f_K^2(i_1, \ldots, i_{\bar{d}}, *, \ldots, *) = f_K^1(i_1, \ldots, i_{\bar{d}-1}, *, \ldots, *) w_{\bar{m}, j_{\bar{d}}} +$
$\quad f_K^2(i_1, \ldots, i_{\bar{d}-1}, *, \ldots, *) v_{\bar{d}, j_{\bar{d}}}$

Finally, the function's output is $\mathsf{F}_K(i_1, \ldots, i_d) = g^{f_K^1(i_1,\ldots,i_d)}$.

Using this notation, we can write

$$p(x_1, \ldots, x_m) = \sum_{0 \leq i_1,\ldots,i_d \leq m} F_K(i_1, \ldots, i_d) \prod_{j=1}^{d} x_{i_j}$$

$\mathsf{PRF.CFEval}_{\mathsf{Poly}}(K, x_1, \ldots, x_m)$.
Let $p_d(x_1, \ldots, x_m) = (p_d^1(x_1, \ldots, x_m), p_d^2(x_1, \ldots, x_m))$ be the following recursive function (where $0 - 1$ is denoted by $*$):

If $d = *$, then:
$p_*^1(x_1, \ldots, x_m) = y_0$, $p_*^2(x_1, \ldots, x_m) = z_0$.
Else:
$p_d^1(x_1, \ldots, x_m) = p_{d-1}^1(x_1, \ldots, x_m) \sum_{k=0}^{m} x_k y_{k,i_k} +$
$\quad p_{d-1}^2(x_1, \ldots, x_m) \sum_{k=0}^{m} x_k z_{k,i_k}$
$p_d^2(x_1, \ldots, x_m) = p_{d-1}^1(x_1, \ldots, x_m) \sum_{k=0}^{m} x_k w_{k,i_k} +$
$\quad p_{d-1}^2(x_1, \ldots, x_m) \sum_{k=0}^{m} x_k v_{k,i_k}$

Finally, the algorithm outputs $g^{p(x_1,\ldots,x_m)} = g^{p_d^1(x_1,\ldots,x_m)}$.

We prove correctness of $\mathsf{PRF.CFEval}_{\mathsf{Poly}}(K, x_1, \ldots, x_m)$ by induction on d. If $d = *$, then the algorithm is clearly correct. Without loss of generality, assume that $d \neq *$. If the algorithm is correct for $d-1$, i.e.,

$$p_{d-1}^1(x_1,\ldots,x_m) = \sum_{0\leq i_1,\ldots,i_{d-1}\leq m} f_K^1(i_1,\ldots,i_{d-1},*)\prod_{j=1}^{d-1} x_{i_j}$$

$$p_{d-1}^2(x_1,\ldots,x_m) = \sum_{0\leq i_1,\ldots,i_{d-1}\leq m} f_K^2(i_1,\ldots,i_{d-1},*)\prod_{j=1}^{d-1} x_{i_j}$$

Then we show its correctness for d.

First, by definition of our algorithm we have:

$$p_d^1(x_1,\ldots,x_m) = p_{d-1}^1(x_1,\ldots,x_m)\sum_{k=0}^{m} x_k y_{k,i_k} + p_{d-1}^2(x_1,\ldots,x_m)\sum_{k=0}^{m} x_k z_{k,i_k} \quad (1)$$

If we then apply our inductive assumption to (1) we obtain:

$$\sum_{0\leq i_1,\ldots,i_{d-1}\leq m} f_K^1(i_1,\ldots,i_{d-1},*)\prod_{j=1}^{d-1} x_{i_j}\sum_{k=0}^{m} x_k y_{k,i_k} + \sum_{0\leq i_1,\ldots,i_{d-1}\leq m} f_K^2(i_1,\ldots,i_{d-1},*)\prod_{j=1}^{d-1} x_{i_j}\sum_{k=0}^{m} x_k z_{k,i_k}$$

that by rewriting becomes

$$\sum_{0\leq i_1,\ldots,i_{d-1}\leq m}\prod_{j=1}^{d-1} x_{i_j}\left(\sum_{k=0}^{m} x_k(f_K^1(i_1,\ldots,i_{d-1},*)y_{k,i_k} + f_K^2(i_1,\ldots,i_{d-1},*)z_{k,i_k})\right)$$

Next, we can apply the recursive definition of $f_K(i_1,\ldots,i_d)$:

$$\sum_{0\leq i_1,\ldots,i_{d-1}\leq m}\prod_{j=1}^{d-1} x_{i_j}\sum_{k=0}^{m} x_k f_K^1(i_1,\ldots,i_{d-1},k)$$

and for $i_d = k$ we get the desired result for d:

$$\sum_{0\leq i_1,\ldots,i_d\leq m} f_K^1(i_1,\ldots,i_d)\prod_{j=1}^{d} x_{i_j}$$

Analogously, one can show correctness of $p_d^2(x_1,\ldots,x_m)$.

At each step, the algorithm makes only one recursive call and m operations. Thus, $p_d(x_1,\ldots,x_m)$ can be computed in time $O(dm)$.

Matrix-Vector Multiplication In this case we define a PRF for a "small" input domain, namely the set $[1..n] \times [1..d]$, where both n and d are polynomial in the security parameter. The function $\mathsf{PRF}_{\mathsf{M}}$ is defined as follows.

$\mathsf{PRF.KG}(1^\lambda, n, d)$**.** Generate a group description $(p, g, \mathbb{G}) \xleftarrow{\$} \mathcal{G}(1^\lambda)$. The key K consists of $2(n+d)$ random values: $A_1, B_1, \ldots, A_d, B_d \xleftarrow{\$} \mathbb{G}$ and $\alpha_1, \beta_1, \ldots, \alpha_n, \beta_n \xleftarrow{\$} \mathbb{Z}_p$.

$\mathsf{F}_K(i, j)$**.** For $i \in [1..n]$, $j \in [1..d]$ define $\mathsf{F}_K(i, j) = A_j^{\alpha_i} B_j^{\beta_i}$

Theorem 3. *If the Decision Linear assumption holds for $\mathcal{G}$, then $\mathsf{PRF}_{\mathsf{M}}$ is a pseudorandom function.*

The proof of this theorem follows in a straightforward way from the random self-reducibility of the Decision Linear problem (it is a simple extension of Lemma 1 in [16]).

In what follows we show that $\mathsf{PRF}_{\mathsf{M}}$ admits closed form efficiency for matrix-vector multiplication.

Let $R = [R_{i,j}]$ be an $n \times d$ matrix defined over $\mathbb{G}$. And let $\boldsymbol{x}$ be a d-dimensional vector $\boldsymbol{x} = [x_1, \ldots, x_d]$ defined over $\mathbb{Z}_p$. Denote with with $\mathsf{Matrix}(M, \boldsymbol{x})$ the n dimensional vector $\boldsymbol{\rho} = [\rho_1, \ldots, \rho_n] \in \mathbb{G}^n$ defined by $\rho_i = \prod_j R_{i,j}^{x_j}$. We now show an algorithm $\mathsf{PRF.CFEval}_{\mathsf{Matrix}}(K, \boldsymbol{x})$ which computes this $\boldsymbol{\rho}$ in $O(n+d)$ time (rather than $O(nd)$) when $R = (\mathsf{F}_K(i, j))$.

Compute $A = \prod_{j=1}^{d} A_j^{x_j}$ and $B = \prod_{j=1}^{d} B_j^{x_j}$
For $i = 1$ to n:
  $\rho_i = A^{\alpha_i} B^{\beta_i}$
Output $\boldsymbol{\rho} = [\rho_1, \ldots, \rho_n]$

Correctness is easily seen since

$$\rho_i = \prod_j \mathsf{F}_K(i, j)^{x_j} = \prod_j (A_j^{\alpha_i} B_j^{\beta_j})^{x_j} =$$

$$(\prod A_j^{x_j})^{\alpha_i} (\prod B_j^{x_j})^{\beta_i} = A^{\alpha_i} B^{\beta_i}$$

REMARK. We note that by removing the S_j and β_i values, one can obtain a more efficient version (twice as fast) of this PRF, that is secure under the (stronger) DDH assumption.

4 Our Protocols

4.1 Polynomials of Degree d in each variable

In this section we propose the construction of a scheme, $\mathcal{VC}_{\mathsf{Poly}}$, for delegating the computation of multivariate polynomials. Our scheme builds upon the techniques in [4], additionally providing a mechanism for public verifiability.

The family of functions $\mathcal{F}$ supported by our protocol is the set of polynomials $f(x_1, \ldots, x_m)$ with coefficients in $\mathbb{Z}_p$ (for some large prime p that we define later), m variables, and degree at most d in each variable. These polynomials have up to $l = (d+1)^m$ terms which we index by $(i_1, \ldots, i_m)$, for $0 \leq i_j \leq d$. For simplicity, we define the following function $h : \mathbb{Z}_p^m \to \mathbb{Z}_p^l$ which expands the input $\boldsymbol{x}$ to the vector $(h_1(\boldsymbol{x}), \ldots, h_l(\boldsymbol{x}))$ of all the monomials as follows: for all $1 \leq j \leq l$, write

$j = (i_1, \ldots, i_m)$ with $0 \leq i_k \leq d$, then $h_j(x) = (x_1^{i_1} \cdots x_m^{i_m})$. So, using this notation we can write the polynomial as $f(\boldsymbol{x}) = \langle \boldsymbol{f}, h(\boldsymbol{x}) \rangle = \sum_{j=1}^{l} f_j \cdot h_j(\boldsymbol{x})$.

Our construction works over groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of the same prime order p, equipped with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let $\boldsymbol{R} \in \mathbb{G}_1^l$ be an l-dimensional vector of random group elements. In the previous Section we defined $\mathsf{Poly}(\boldsymbol{R}, \boldsymbol{x}) = \prod_{j=1}^{l} R_j^{h_j(\boldsymbol{x})}$.

Our scheme $\mathcal{VC}_{\mathsf{Poly}}$ works generically for any family of functions $\mathcal{F}$ as described above for which there exists a PRF that has closed form efficiency relative to $\mathsf{Poly}(\boldsymbol{R}, \boldsymbol{x})$, and has range $\mathcal{Y} = \mathbb{G}_1$. Its security is based on the security of the PRF and on the hardness of solving co-CDH in these groups.

For the PRF we can use the ones presented in [4], which have the required closed form efficiency, if we instantiate them over the group $\mathbb{G}_1$. However those constructions are based on assumptions (DDH, Strong-DDH) that in general may *not* hold in bilinear groups. Therefore, to securely adapt them to our case, we must use asymmetric bilinear groups where the External Diffie-Hellman assumption holds in $\mathbb{G}_1$. Details of this solution will appear in the final version.

A better approach is to use the PRF construction $\mathsf{PRF}_{\mathsf{LW}}$, $\mathsf{PRF}_{\mathsf{eLW}}$ given in Section 3.1, that are based on the Decision Linear assumption for which we know no attacks even in the presence of bilinear maps. As we describe, these PRFs have closed form efficiency with respect to Poly. This allows to base our scheme on a weaker assumption and to instantiate it also with symmetric pairings, i.e., where $\mathbb{G}_1 = \mathbb{G}_2$.

The description of our scheme $\mathcal{VC}_{\mathsf{Poly}}$ follows.

$\mathsf{KeyGen}(1^\lambda, f)$**.** Generate the description of bilinear groups $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$, and a key of a PRF, $K \xleftarrow{\$} \mathsf{PRF.KG}(1^\lambda, \lceil \log d \rceil, m)$, with range in $\mathbb{G}_1$. Choose a random $\alpha \xleftarrow{\$} \mathbb{Z}_p$, and compute $W_i = g_1^{\alpha \cdot f_i} \cdot \mathsf{F}_K(i)$, $\forall i = 1, \ldots, l$. Let $W = (W_1, \ldots, W_l) \in \mathbb{G}_1^l$.
Output $\mathsf{EK}_f = (f, W)$, $\mathsf{PK}_f = e(g_1, g_2)^\alpha$, $\mathsf{SK}_f = k$.

$\mathsf{ProbGen}(\mathsf{PK}_f, \mathsf{SK}_f, \boldsymbol{x})$**.** Output $\sigma_x = \boldsymbol{x}$ and
$\mathsf{VK}_x = e(\mathsf{PRF.CFEval}_{\mathsf{Poly}}(K, h(\boldsymbol{x})), g_2)$.

$\mathsf{Compute}(\mathsf{EK}_f, \sigma_x)$**.** Let $\mathsf{EK}_f = (f, W)$ and $\sigma_x = \boldsymbol{x}$. Compute $y = f(\boldsymbol{x}) = \sum_{i=1}^{l} f_i \cdot h_i(\boldsymbol{x})$, $V = \prod_{i=1}^{l} W_i^{h_i(\boldsymbol{x})}$, and return $\sigma_y = (y, V)$.

$\mathsf{Verify}(\mathsf{PK}_f, \mathsf{VK}_x, \sigma_y)$**.** Parse σ_y as (y, V). If $e(V, g_2) = (\mathsf{PK}_f)^y \cdot \mathsf{VK}_x$, then output y, otherwise output $\perp$.

Theorem 4. *If $\mathcal{G}$ is such that the co-CDH assumption ϵ_{cdh}-holds, and F is ϵ_{prf}-secure, then any PPT adversary $\mathcal{A}$ making at most $q = poly(\lambda)$ queries has advantage*

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{PubVer}}(\mathcal{VC}_{\mathsf{Poly}}, \mathcal{F}, q, \lambda) \leq \epsilon_{cdh} + \epsilon_{prf}$$

To prove the theorem, we define the following games, where $G_i(\mathcal{A})$ denotes the output of Game i run with adversary $\mathcal{A}$:

Game 0: this is the same as $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{PubVer}}(\mathcal{VC}_{\mathsf{Poly}}, \mathcal{F}, q, \lambda)$.

Game 1: this game is similar to Game 0, except for the following change in the evaluation of the $\mathsf{ProbGen}$ algorithm. For any $\boldsymbol{x}$ asked by the adversary during the game, instead of computing VK_x using the efficient $\mathsf{PRF.CFEval}$ algorithm, the inefficient evaluation $\mathsf{VK}_x = \prod_{i=1}^{l} \mathsf{F}_K(i)^{h_i(\boldsymbol{x})}$ is used.

Game 2: this game is the same as Game 1, except that each value $\mathsf{F}_k(i)$ is replaced by an element $R_i \xleftarrow{\$} \mathbb{G}_1$ chosen uniformly at random.

The proof of the theorem proceeds by a standard hybrid argument, and is obtained by combining the proofs of the following claims.

Claim. $\Pr[G_0(\mathcal{A}) = 1] = \Pr[G_1(\mathcal{A}) = 1]$.

Proof. The only difference between the two games is in the computation of the $\mathsf{ProbGen}$ algorithm. However, by correctness of $\mathsf{PRF.CFEval}$, such difference does not change the distribution of the values VK_x returned to the adversary. Thus, the probability of the adversary winning in Game 1, i.e., $\Pr[G_1(\mathcal{A}) = 1]$, cannot change.

Claim. $|\Pr[G_1(\mathcal{A}) = 1] - \Pr[G_2(\mathcal{A}) = 1]| \leq \epsilon_{prf}$

Proof. The difference between Game 2 and Game 1 is that we replaced the output of the pseudo-random function F_K, with uniformly random group elements. It is easy to see that any adversary $\mathcal{A}$ for which such difference is greater than ϵ_{prf} can be reduced to an attacker that has the same advantage against the security of the PRF.

Claim. $\Pr[G_2(\mathcal{A}) = 1] \leq \epsilon_{cdh}$.

Proof. Assume by contradiction that there exists a PPT adversary $\mathcal{A}$ that has advantage greater than ϵ_{cdh} of winning in Game 2, then we show that we can build an efficient algorithm $\mathcal{B}$ which uses $\mathcal{A}$ to solve the co-CDH problem for $\mathcal{G}$ with the same probability ϵ_{cdh}.

$\mathcal{B}$ takes as input a group description $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, e)$ and random elements g_1^a, g_2^b, and it proceeds as follows. It chooses $W_1, \ldots, W_l \xleftarrow{\$} \mathbb{G}_1$, sets $\mathsf{EK}_f = (f, W)$, and computes $\mathsf{PK}_f = e(g_1^a, g_2^b)$. It is easy to check that the public and evaluation keys are perfectly distributed as in Game 2. Next, for $i = 1$ to l, it computes $Z_i = e(W_i, g_2)/\mathsf{PK}_f^{f_i} \in \mathbb{G}_T$. $\mathcal{B}$ runs $\mathcal{A}(\mathsf{PK}_f, \mathsf{EK}_f)$ and answers its queries as follows. Let $\boldsymbol{x}$ be the queried value. $\mathcal{B}$ computes $\mathsf{VK}_x = \prod_{i=1}^{l} Z_i^{h_i(\boldsymbol{x})}$, and returns it to $\mathcal{A}$. By the bilinear property of $e(\cdot, \cdot)$, this computation of VK_x is equivalent to the one in Game 2.

Finally, let $\hat{\sigma}_y = (\hat{y}, \hat{V})$ be the output of $\mathcal{A}$ at the end of the game, such that for some $\boldsymbol{x}^*$ chosen by $\mathcal{A}$ it holds $\mathsf{Verify}(\mathsf{PK}_f, \mathsf{VK}_{x^*}, \hat{\sigma}_y) = \hat{y}$, $\hat{y} \neq \perp$ and $\hat{y} \neq f(\boldsymbol{x}^*)$. By verification, this means that

$$e(\hat{V}, g_2) = e(g_1^a, g_2^b)^{\hat{y}} \cdot \mathsf{VK}_{x^*}.$$

Let $y = f(\boldsymbol{x}^*)$ be the correct output of the computation. Then, by correctness it also holds:

$$e(V, g_2) = e(g_1^a, g_2^b)^{y} \cdot \mathsf{VK}_{x^*}$$

where $V = \prod_{i=1}^{l} W_i^{h_i(\boldsymbol{x}^*)}$. So, dividing the two verification equations, we obtain that $e(\hat{V}/V, g_2) = e(g_1^a, g_2^b)^{\hat{y}-y}$. $\mathcal{B}$ can thus compute $g_1^{ab} = (\hat{V}/V)^{1/(\hat{y}-y)}$. Therefore, if $\mathcal{A}$ wins in Game 2 with probability ϵ_{cdh}, then $\mathcal{B}$ solves co-CDH with the same probability.

Efficiency Analysis. The offline cost for running KeyGen is $O((d+1)^m)$, whereas the client's online cost for outsourcing a computation on $\boldsymbol{x}$ (i.e., running $\mathsf{ProbGen}$) is $O(m \log d)$. The cost at the server for running the $\mathsf{Compute}$ algorithm is $O((d+1)^m)$, specifically twice the cost of computing $f(\boldsymbol{x})$. Finally, the cost of verifying the result is completely independent of the size of the input and the function, $O(1)$. This property is interesting because it means that clients (other than the delegator) can verify the result of the computation almost for free. To the best of our knowledge, this property is not achieved by other protocols.

4.2 m-Variate Polynomials of Total Degree d

The protocol for this case is exactly the same as the protocol $\mathcal{VC}_{\mathsf{Poly}}$ described above, with the following changes: (i) adjust the number of monomials to $l = (m+1)^d$; (ii) use a PRF with closed-form efficiency for polynomials of this form.

Again, we can use the PRFs in [4], provided we adapt them to work over $\mathbb{G}_1$ and we use asymmetric bilinear groups where the External Diffie-Hellman assumption holds in $\mathbb{G}_1$. Or we can use the variation of the Lewko-Waters PRF ($\mathsf{PRF}_{\mathsf{eLW}}$) described in Section 3.1 that is based on the weaker Decision Linear assumption.

The efficiency analysis of this solution is as follows: the offline cost for running KeyGen is $O((m+1)^d)$ which is also the cost for the server's computation. The client's online cost for outsourcing a computation on $\boldsymbol{x}$ (i.e., running $\mathsf{ProbGen}$) is $O(md)$. Again, verification is constant.

4.3 Matrix Multiplication

Let p be a large prime, and $d \geq 1$ an integer. In this section we propose a scheme $\mathcal{VC}_{\mathsf{Matrix}}$ that allows to verifiably delegate computation for the following family of functions $\mathcal{F}$. $\mathcal{F}$ is the set of matrices $M \in \mathbb{Z}_p^{n \times d}$, for any $n \geq 1$, and the inputs are vectors $\boldsymbol{x} \in \mathbb{Z}_p^d$. So, the function is the multiplication $M \cdot \boldsymbol{x}$. Given such a scheme, we observe that it is straightforward to extend it to the case of matrix multiplication, i.e., $M \cdot M'$, where $M' \in \mathbb{Z}_p^{d \times m}$.

$\mathsf{KeyGen}(1^\lambda, M)$**.** Let $M \in \mathbb{Z}_p^{n \times d}$ be a matrix. Generate the description of bilinear groups $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$. Generate a key K for an algebraic PRF with domain $[1..n] \times [1..d]$ and range $\mathbb{G}_1$.
For $1 \leq i \leq d$, $1 \leq j \leq n$, compute $W_{i,j} = g_1^{\alpha \cdot M_{i,j}} \cdot \mathsf{F}_K(i,j)$, and let $W = (W_{i,j}) \in \mathbb{G}_1^{n \times d}$.
Output $\mathsf{SK}_M = K$, $\mathsf{EK}_M = (M, W)$, and $\mathsf{PK}_M = A$.

$\mathsf{ProbGen}(\mathsf{SK}_M, \boldsymbol{x})$**.** Let $\boldsymbol{x} = (x_1, \ldots, x_d) \in \mathbb{Z}_p^d$ be the input. Let R be the matrix defined by $R = [\mathsf{F}_K(i,j)]$. Compute $\boldsymbol{\rho}_x = \mathsf{PRF.CFEval}_{\mathsf{Matrix}}(K, x)$ in $O(n+d)$ using the closed form efficiency. Recall that $\rho_{x,i} = \prod_{j=1}^d \mathsf{F}_K(i,j)^{x_j}$, and define $\tau_{x,i} = e(\rho_{x,i}, g_2)$. Finally, output the encoding $\sigma_{\boldsymbol{x}} = \boldsymbol{x}$, and the verification key $\mathsf{VK}_{\boldsymbol{x}} = (\tau_{x,1}, \ldots, \tau_{x,n})$.

$\mathsf{Compute}(\mathsf{EK}_M, \sigma_{\boldsymbol{x}})$**.** Let $\mathsf{EK}_M = (M, W)$ and $\sigma_{\boldsymbol{x}} = \boldsymbol{x}$. Compute $\boldsymbol{y} = M \cdot \boldsymbol{x}$ and $\boldsymbol{V} = (V_1, \ldots, V_n)$ as follows: $V_j = (\prod_{i=1}^d W_{i,j}^{x_i})$. Output $\sigma_y = (\boldsymbol{y}, \boldsymbol{V})$.

$\mathsf{Verify}(\mathsf{PK}_M, \mathsf{VK}_{\boldsymbol{x}}, \sigma_y)$**.** Parse σ_y as $(\boldsymbol{y}, \boldsymbol{V})$. If $e(V_i, g_2) = (\mathsf{PK}_M)^{y_i} \cdot \tau_{x,i}$ $\forall i = 1, \ldots, n$, then output $\boldsymbol{y}$, otherwise output $\perp$.

We prove security via the following theorem.

Theorem 5. *If $\mathcal{G}$ is such that the co-CDH assumption ϵ_{cdh}-holds, and F is ϵ_{prf}-secure, then any PPT adversary $\mathcal{A}$ making at most $q = poly(\lambda)$ queries has advantage*

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{PubVer}}(\mathcal{VC}_{\mathsf{Matrix}}, \mathcal{F}, q, \lambda) \leq \epsilon_{cdh} + \epsilon_{prf}$$

Let us define the following games, where $G_i(\mathcal{A})$ is the output of Game i run with adversary $\mathcal{A}$:

Game 0: this is the same as $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{PubVer}}(\mathcal{VC}_{\mathsf{Matrix}}, \mathcal{F}, q, \lambda)$.

Game 1: this game is similar to Game 0, except for the following change in the evaluation of the $\mathsf{ProbGen}$ algorithm. For any $\boldsymbol{x}$ asked by the adversary during the game, instead of computing VK_x using the efficient $\mathsf{PRF.CFEval}$ algorithm, the inefficient evaluation of $\boldsymbol{\rho}_x$ is used $\rho_{x,i} = \prod_{j=1}^d \mathsf{F}_K(i,j)^{x_j}$.

Game 2: this game is defined as Game 1, except that the matrix W is computed as $W_{i,j} = g^{\alpha M_{i,j}} \cdot R_{i,j}$ where for all i, j $R_{i,j} \xleftarrow{\$} \mathbb{G}_1$ is chosen uniformly at random.

The proof of this theorem is similar to that of Theorem 4, and is obtained by proving the following claims.

Claim. $\Pr[G_0(\mathcal{A}) = 1] = \Pr[G_1(\mathcal{A}) = 1]$.

Proof. The only difference between the two games is in the computation of the $\mathsf{ProbGen}$ algorithm. However, by the correctness of the closed form efficiency of our PRF, there is no change in the distribution of the values $\tau_{x,j}$ returned to the adversary. Thus, the probability of the adversary winning in Game 1, i.e., $\Pr[G_1(\mathcal{A}) = 1]$, remains the same.

Claim. $|\Pr[G_1(\mathcal{A}) = 1] - \Pr[G_2(\mathcal{A}) = 1]| \leq \epsilon_{prf}$

Proof. The difference between Game 2 and Game 1 is that we replaced each pseudorandom value $\mathsf{F}_K(i, j)$ with a random value $R_{i,j}$.

Claim. $\Pr[G_2(\mathcal{A}) = 1] \leq \epsilon_{cdh}$.

Proof. This proof is a simple extension of the proof of Claim 4.1. We describe it below for completeness.

Assume by contradiction that there exists a PPT adversary $\mathcal{A}$ such that the probability of $\mathcal{A}$ winning in Game 2 is a non-negligible function ϵ, then we show that we can build an efficient algorithm $\mathcal{B}$ which uses $\mathcal{A}$ to solve the co-CDH problem with probability $\epsilon_{cdh} \geq \epsilon$. $\mathcal{B}$ takes as input a group description $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, e)$ and two random elements g_1^a, g_2^b, and proceeds as follows. For $i = 1, \ldots, d$ and $j = 1, \ldots, n$, $\mathcal{B}$ chooses $W_{i,j} \xleftarrow{\$} \mathbb{G}_1$, sets $\mathsf{EK}_M = (M, W)$, and computes $\mathsf{PK}_M = e(g_1^a, g_2^b)$. It is easy to check that the public and evaluation keys are perfectly distributed as in Game 2. Next, for $i = 1, \ldots, d$ and $j = 1, \ldots, n$, it computes $Z_{i,j} = e(W_{i,j}, g_2)/\mathsf{PK}_M^{M_{i,j}}$. Then $\mathcal{B}$ runs $\mathcal{A}(\mathsf{PK}_M, \mathsf{EK}_M)$ and answers its queries as follows. Let $\boldsymbol{x}$ be the queried vector. $\mathcal{B}$ computes $\tau_{x,j} = \prod_{i=0}^{d} Z_{i,j}^{x_i}$ for $j = 1$ to n, and returns $\mathsf{VK}_x = (\tau_{x,1}, \ldots, \tau_{x,n})$ to $\mathcal{A}$. By the bilinear property of $e(\cdot, \cdot)$, this computation of VK_x is equivalent to the one done in Game 2.

Finally, let $\hat{\sigma}_y = (\hat{\boldsymbol{y}}, \hat{\boldsymbol{V}})$ be the output of $\mathcal{A}$ at the end of the game, such that for some $\boldsymbol{x}^*$ chosen by $\mathcal{A}$ it holds $\mathsf{Verify}(\mathsf{PK}_f, \mathsf{VK}_{\boldsymbol{x}^*}, \hat{\sigma}_y) = \hat{\boldsymbol{y}}$, $\hat{\boldsymbol{y}} \neq \perp$ and $\hat{\boldsymbol{y}} \neq M \cdot \boldsymbol{x}^*$. Let $\boldsymbol{y} = M \cdot \boldsymbol{x}^*$ be the correct output of the multiplication. Since $\hat{\boldsymbol{y}} \neq \boldsymbol{y}$ there must exist an index $j \in \{1, \ldots, n\}$ such that $\hat{y}_j \neq y_j$. However, by verification, for such j it holds

$$e(\hat{V}_j, g_2) = e(g_1^a, g_2^b)^{\hat{y}_j} \cdot \tau_{\boldsymbol{x}^*,j}$$

Moreover, by correctness we have:

$$e(V_j, g_2) = e(g_1^a, g_2^b)^{y_j} \cdot \tau_{\boldsymbol{x}^*,j}$$

where $V_j = \prod_{i=0}^{d} W_{i,j}^{x_i^*}$. Dividing the two verification equations, we obtain that $e(\hat{V}_j/V_j, g_2) = e(g_1^a, g_2^b)^{\hat{y}_j - y_j}$. $\mathcal{B}$ can thus compute $g_1^{ab} = (\hat{V}_j/V_j)^{1/(\hat{y}_j - y_j)}$. Therefore, if $\mathcal{A}$ wins in Game 2 with probability ϵ, then $\mathcal{B}$ solves the co-CDH problem with probability $\epsilon_{cdh} \geq \epsilon$, which completes the proof.

Efficiency Analysis Using our protocol, a client can spend a single offline cost $O(dn)$ (for each matrix M – running the key generation algorithm), while keeping a secret key of size $O(d+n)$. It can then outsource the computation of multiplying $M \cdot \boldsymbol{x}$ on many $\boldsymbol{x}$'s with an online cost $O(d+n)$ per vector (running ProbGen), whereas the verification can be performed in time $O(n)$. The cost at the server for computing the function, i.e., running Compute, is $O(nd)$, and more precisely it is twice the cost of performing $M \cdot \boldsymbol{x}$ (once for M and once for W). We remark that the running times of KeyGen and Verify are optimal as they are linear in the size of their respective inputs.

Our protocol extends to the case of matrix multiplications $M \cdot M'$, where $M \in \mathbb{Z}_p^{n \times d}$ and $M' \in \mathbb{Z}_p^{d \times m}$ by considering each column of M'. In this case a client can outsource and verify such computation with a total online cost $O(dm + nm)$, which is *optimal* as it is the cost of processing the input $M' \in \mathbb{Z}_p^{d \times m}$ and the output $M \cdot M' \in \mathbb{Z}_p^{n \times m}$.

As we pointed out the server has to perform two matrix multiplications: one $(M \cdot M')$ in $\mathbb{Z}_p$ and the other one $(W \cdot M')$ "in the exponent". We note that the server can perform both computations using one of the optimized algorithms for rectangular matrix multiplication (e.g. Strassen's algorithm), that require time strictly less than the trivial $O(nmd)$. Therefore the server's computation is asymptotically the same as a basic matrix multiplication.

Acknowledgement

The research of the second author was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

1. B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. G. Spirakis, editors, *ICALP 2010: 37th International Colloquium on Automata, Languages and Programming, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 152–163, Bordeaux, France, July 6–10, 2010. Springer, Berlin, Germany.
2. L. Babai. Trading group theory for randomness. In *17th Annual ACM Symposium on Theory of Computing*, pages 421–429, Providence, Rhode Island, USA, May 6–8, 1985. ACM Press.
3. M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya. Incentivizing outsourced computation. In *Workshop on Economics of Networked Systems – NetEcon*, pages 85–90, 2008.
4. S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In P. Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 111–131, Santa Barbara, CA, USA, Aug. 14–18, 2011. Springer, Berlin, Germany.
5. D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229, Santa Barbara, CA, USA, Aug. 19–23, 2001. Springer, Berlin, Germany.
6. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, Dec. 9–13, 2001. Springer, Berlin, Germany.
7. K.-M. Chung, Y. Kalai, and S. P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In T. Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 483–501, Santa Barbara, CA, USA, Aug. 15–19, 2010. Springer, Berlin, Germany.

8. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482, Santa Barbara, CA, USA, Aug. 15–19, 2010. Springer, Berlin, Germany.
9. C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, Maryland, USA, May 31 – June 2, 2009. ACM Press.
10. S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In R. E. Ladner and C. Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 113–122, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press.
11. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
12. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In A. Juels, R. N. Wright, and S. Vimercati, editors, *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 89–98, Alexandria, Virginia, USA, Oct. 30 – Nov. 3, 2006. ACM Press. Available as Cryptology ePrint Archive Report 2006/309.
13. J. Kilian. A note on efficient zero-knowledge proofs and arguments. In *24th Annual ACM Symposium on Theory of Computing*, pages 723–732, Victoria, British Columbia, Canada, May 4–6, 1992. ACM Press.
14. J. Kilian. Improved efficient arguments. In *International Cryptology Conference on Advances in Cryptology*, pages 311–314, London (UK), 1995. Springer-Verlag.
15. A. Lewko and B. Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. Crypto 2012, to appear, 2012.
16. A. B. Lewko and B. Waters. Efficient pseudorandom functions from the decisional linear assumption and weaker variants. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *ACM CCS 09: 16th Conference on Computer and Communications Security*, pages 112–120, Chicago, Illinois, USA, Nov. 9–13, 2009. ACM Press.
17. S. Micali. Cs proofs. In *35th Annual Symposium on Foundations of Computer Science*, Santa Fe, New Mexico, Nov. 20–22, 1994.
18. F. Monrose, P. Wyckoff, and A. D. Rubin. Distributed execution with remote audit. In *ISOC Network and Distributed System Security Symposium – NDSS'99*, San Diego, California, USA, Feb. 3–5, 1999. The Internet Society.
19. C. Papamanthou, E. Shi, and R. Tamassia. Publicly verifiable delegation of computation. Cryptology ePrint Archive, Report 2011/587, 2011.
20. B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. TCC 2012, 2012.
21. S. W. Smith and S. Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks*, 31:831–860, 1999.
22. B. Yee. *Using Secure Coprocessors*. PhD thesis, Carnegie Mellon University, 1994.

A Multi-Function Verifiable Computation

In this section we consider the problem of realizing *multi-function* verifiable computation schemes. In the following section, we define the notion of multi-function verifiable computation, and then we propose a new scheme.

A.1 Definition

Multi-Function verifiable computation [20] considers a setting in which a client wants to delegate the computation of multiple functions on some a-priori fixed inputs. Technically, this means that the $\mathsf{ProbGen}$ algorithm must be completely decoupled from KeyGen, in the sense that one can prepare the input before knowing which function will be applied on it. More formally, let $\mathcal{F}$ be a family of functions. A multi-function verifiable computation scheme is defined by the following algorithms:

$\mathsf{Setup}(1^\lambda, \mathcal{F}) \to (\mathsf{PK}, \mathsf{SK})$**:** on input the security parameter λ and the description of a family of functions $\mathcal{F}$, the setup algorithm generates a pair of public and private parameters PK, SK. In particular, we assume that $\mathcal{F}$ specifies the domain Dom of the functions $f \in \mathcal{F}$.

$\mathsf{KeyGen}(\mathsf{PK}, \mathsf{SK}, f) \to (\mathsf{EK}_f, \mathsf{VK}_f)$**:** on input the security parameter λ and a function $f \in \mathcal{F}$, this algorithm produces a verification key VK_f, used to verify the correctness of the delegated computations, and a public evaluation key EK_f which is handed to the server to delegate computations of f.

$\mathsf{ProbGen}(\mathsf{PK}, \mathsf{SK}, x) \to (\sigma_x, \mathsf{VK}_x)$**:** on input the public parameters, and a value $x \in \mathsf{Dom}$, the problem generation algorithm produces an encoding σ_x of x together with a verification key VK_x.

$\mathsf{Compute}(\mathsf{EK}_f, \sigma_x) \to \sigma_y$**:** given the public evaluation key EK_f and the encoding σ_x of an input x, this algorithm is run by the worker to compute an encoded version σ_y of $y = f(x)$.

$\mathsf{Verify}(\mathsf{VK}_f, \mathsf{VK}_x, \sigma_y) \to y \cup \perp$**:** on input the verification keys VK_f and VK_x, and an encoded output σ_y, this algorithm returns a value y or an error $\perp$.

According to whether VK_f and VK_x can be publicly revealed or kept secret, we obtain a definition of a multi-function verifiable computation that is public verifiable or secret verifiable respectively. Moreover, if KeyGen and $\mathsf{ProbGen}$ do not use the secret key SK, then the scheme is *publicly delegatable*.

SECURITY. For any multi-function verifiable computation scheme $\mathcal{VC}$, consider the following experiment:

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{PriVerif}}[\mathcal{VC}, \mathcal{F}, \lambda]$

$(\mathsf{PK}, \mathsf{SK}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{F})$
$(f, x^*, \hat{\sigma}_y) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyGen}}(\cdot), \mathcal{O}_{\mathsf{ProbGen}}(\cdot), \mathcal{O}_{\mathsf{Verify}}(\cdot,\cdot,\cdot)}(\mathsf{PK})$
$\hat{y} \leftarrow \mathsf{Verify}(\mathsf{VK}_f, vk_{x^*}, \hat{\sigma}_y)$
If $\hat{y} \neq \perp$ and $\hat{y} \neq f(x^*)$, output 1, else output 0.

In the above experiment, the adversary is given access to three oracles that works as follows. On input $f \in \mathcal{F}$, $\mathcal{O}_{\mathsf{KeyGen}}(f)$ runs $(\mathsf{EK}_f, \mathsf{VK}_f) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{PK}, \mathsf{SK}, f)$, returns EK_f and stores VK_f. On input $x \in \mathsf{Dom}$, $\mathcal{O}_{\mathsf{ProbGen}}(x)$ runs $(\sigma_x, \mathsf{VK}_x) \xleftarrow{\$} \mathsf{ProbGen}(\mathsf{PK}, \mathsf{SK}, x)$, returns σ_x and stores VK_x. On input $f \in \mathcal{F}$, $x \in \mathsf{Dom}$ and a purported output σ_y, $\mathcal{O}_{\mathsf{Verify}}(f, x, \sigma_y)$ runs $y \leftarrow \mathsf{Verify}(\mathsf{VK}_f, \mathsf{VK}_x, \sigma_y)$ and returns y.

For any $\lambda \in \mathbb{N}$, any family of functions $\mathcal{F}$, we define the advantage of $\mathcal{A}$ making at most $q = poly(\lambda)$ queries in the above experiment against $\mathcal{VC}$ as

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{PriVerif}}(\mathcal{VC}, \mathcal{F}, q, \lambda) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{PriVerif}}[\mathcal{VC}, \mathcal{F}, \lambda] = 1].$$

Definition 5. *Let $\lambda \in \mathbb{N}$ be the security parameter, and $\mathcal{F}$ be a family of functions. A multi-function verifiable computation scheme $\mathcal{VC}$ is secure for $\mathcal{F}$ if for any PPT $\mathcal{A}$ it holds that $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{PriVerif}}(\mathcal{VC}, \mathcal{F}, q, \lambda)$ is negligible.*

Our definition is in the private setting, though one in the public verifiable or public delegatable settings can be easily obtained by slightly changing the output of the oracles.

A.2 Tools: Homomorphic weak PRFs

A homomorphic weak pseudorandom function consists of algorithms $(\mathsf{wPRF.KG}, \mathsf{G})$. The key generation $\mathsf{wPRF.KG}$ takes as input the security parameter 1^λ and outputs a secret key k and some public parameters pp that specify domain $\mathcal{X}$ and range $\mathcal{Y}$ of the function. On input $X \in \mathcal{X}$, $\mathsf{G}_k(X)$ uses the key k to output a value $Y \in \mathcal{Y}$. First, we require $(\mathsf{wPRF.KG}, \mathsf{G})$ to satisfy the usual weak

pseudorandomness property, i.e., it is ϵ-secure if for any PPT adversary $\mathcal{A}$ and any polynomial $t = t(\lambda)$:

$$\big|\Pr[\mathcal{A}(1^\lambda, \mathsf{pp}, \{X_i, Y_i\}_{i=1}^t) = 1] - \Pr[\mathcal{A}(1^\lambda, \mathsf{pp}, \{X_i, Z_i\}_{i=1}^t) = 1]\big| \leq \epsilon$$

where $(k, \mathsf{pp}) \xleftarrow{\$} \mathsf{wPRF.KG}(1^\lambda)$, and $\forall i = 1, \ldots, t$, $X_i \xleftarrow{\$} \mathcal{X}$, $Y_i = \mathsf{G}_k(X_i)$, $Z_i \xleftarrow{\$} \mathcal{Y}$. In addition, we ask for a *homomorphic* property as follows: for any inputs $X_1, X_2 \in \mathcal{X}$, and any integer coefficients $c_1, c_2 \in \mathbb{Z}$, it holds:

$$\mathsf{G}_k(X_1^{c_1} \cdot X_2^{c_2}) = \mathsf{G}_k(X_1)^{c_1} \cdot \mathsf{G}_k(X_2)^{c_2}$$

An example based on DDH. Let $\mathbb{G}$ be a group of order p. The following construction is an algebraic homomorphic weak PRF. The key is a random $k \xleftarrow{\$} \mathbb{Z}_p$, domain and range are $\mathcal{X} = \mathcal{Y} = \mathbb{G}$, and the function is $\mathsf{G}_k(X) = X^k$. It is trivial to observe that it is homomorphic. Pseudoranomness follows in a straightforward way from random self-reducibility of DDH. For the purpose of our applications, we remark that if $\mathbb{G}_1, \mathbb{G}_2$ are asymmetric bilinear groups, this function can be instantiated in $\mathbb{G}_1$ and proven secure under the XDH assumption in $\mathbb{G}_1$.

An example based on Decision Linear Let $\mathbb{G}$ be a group of order p. The following construction is an algebraic homomorphic weak PRF. The key is a random pair $k_1, k_2 \xleftarrow{\$} \mathbb{Z}_p$, the domain is $\mathcal{X} = \mathbb{G}^2$, and the range is $\mathcal{Y} = \mathbb{G}$. On any pair $(X_1, X_2) \in \mathcal{X}$, the function is $\mathsf{G}_{k_1,k_2}(X_1, X_2) = X_1^{k_1} \cdot X_2^{k_2}$. If for every $(X_1, X_2) \in \mathcal{X}$ we define the operation $(X_1, X_2)^c$ component-wise, i.e., $(X_1, X_2)^c = (X_1^c, X_2^c)$, then it is easy to verify the homomorphic property. The function is weak pseudorandom based on the Decision Linear assumption. The proof follows in a straightforward way from the random self-reducibility of the Decision Linear problem, (it is a simple extension of Lemma 1 in [16]).

A.3 Our Multi-Function Scheme

Here we propose a multi-function verifiable computation scheme, $\mathcal{VC}_{MultiF}$, that is publicly delegatable and private verifiable. This construction works for any family of functions $\mathcal{F}$ that is a vector space of dimension d over a finite field $\mathbb{Z}_p$ for some large prime p.

A function $f \in \mathcal{F}$ is represented as a vector $\boldsymbol{f} = (f_1, \ldots, f_d) \in \mathbb{Z}_p^d$, whereas the domain is $\mathsf{Dom}(\mathcal{F}) = \mathbb{Z}_p^d$. For any $x \in \mathsf{Dom}(\mathcal{F})$, we define $f(x) = \sum_{i=1}^d f_i \cdot x_i$.

$\mathcal{VC}_{MultiF}$ is defined by the following algorithms:

$\mathsf{Setup}(1^\lambda, \mathcal{F})$. Generate the description of a group $\mathbb{G}$ of prime order p, and let $g \in \mathbb{G}$ be a generator. Let pp be the parameters for a homomorphic weak-PRF with domain $\mathcal{X}$ and range $\mathcal{Y} = \mathbb{G}$. Select $R = (R_1, \ldots, R_d) \xleftarrow{\$} \mathcal{X}^d$ at random, and output $\mathsf{SK} = \perp$ and $\mathsf{PK} = (p, g, \mathbb{G}, R)$.

$\mathsf{KeyGen}(\mathsf{PK}, \mathsf{SK}, f)$. Let $R = (R_1, \ldots, R_d)$. Generate a key $k \xleftarrow{\$} \mathsf{wPRF.KG}(1^\lambda)$ of a homomorphic weak PRF with range $\mathcal{Y} = \mathbb{G}$. Choose random $\alpha \xleftarrow{\$} \mathbb{Z}_p$, for all $i = 1$ to d compute $W_i = g^{\alpha \cdot f_i} \cdot \mathsf{G}_k(R_i)$. Let $W = (W_1, \ldots, W_d) \in \mathbb{G}^d$. Output $\mathsf{EK}_f = (f, W)$, $\mathsf{VK}_f = (\alpha, k)$.

$\mathsf{ProbGen}(\mathsf{PK}, \mathsf{SK}, x)$. Let $x = (x_1, \ldots, x_d) \in \mathbb{Z}_p^d$ be the input. Its encoding σ_x is x itself, whereas the verification key is $\mathsf{VK}_x = \prod_{i=1}^d R_i^{x_i}$.

$\mathsf{Compute}(\mathsf{PK}, \mathsf{EK}_f, \sigma_x)$. Let $\mathsf{EK}_f = (f, W)$ and $\sigma_x = x$. Compute $y = f(x) = \sum_{i=1}^d f_i \cdot x_i$, and $V = \prod_{i=1}^d W_i^{x_i}$, and return $\sigma_y = (y, V)$.

$\mathsf{Verify}(\mathsf{VK}_f, \mathsf{VK}_x, \sigma_y)$**.** Parse σ_y as (y, V). If $V = (g^\alpha)^y \cdot \mathsf{G}_k(\mathsf{VK}_x)$, then output y, otherwise output $\perp$.

Theorem 6. *If the weak PRF* $(\mathsf{wPRF.KG}, \mathsf{G})$ *is* ϵ_{wprf}*-secure in a group* $\mathbb{G}$ *of order* p*, then any PPT adversary* $\mathcal{A}$ *making at most* $q = poly(\lambda)$ *queries has advantage*

$$\mathbf{Adv}^{\mathsf{PriVerif}}_{\mathcal{A}}(\mathcal{VC}_{MultiF}, \mathcal{F}, q, \lambda) \leq q \cdot \epsilon_{wprf} + q/p$$

Let us define the following games, and denote by $G_i(\mathcal{A})$ the output of Game i run with adversary $\mathcal{A}$:

Game 0: this is the same as $\mathbf{Exp}^{\mathsf{PriVerif}}_{\mathcal{A}}(\mathcal{VC}_{\mathsf{Matrix}}, \mathcal{F}, q, \lambda)$.

Game 1: this game is similar to Game 0, except for the following change in the way verification queries are answered. Let $\mathcal{O}_{\mathsf{Verify}}(f, x, \sigma_y)$ be the query made by the adversary, and let $\mathsf{VK}_f = (\alpha, k), \mathsf{VK}_x$ be the stored verification keys. Instead of computing $\mathsf{G}_k(\mathsf{VK}_x)$, the challenger computes $\mathsf{VK}'_x = \prod_{i=1}^d \mathsf{G}_k(R_i)^{x_i}$ and makes the comparison by checking whether $V = g^{\alpha y} \cdot \mathsf{VK}'_x$.

Game 2,j: for all $j = 0$ to q, Game 2,j is the same as Game 0, except that the first j queries made by the adversary to the oracle $\mathcal{O}_{\mathsf{KeyGen}}(\cdot)$ are answered as follows. On input a function f, instead of generating a key k for the weak PRF, each W_i is computed as $W_i = g^{\alpha \cdot f_i} \cdot Z_i$, where $Z_i \xleftarrow{\$} \mathbb{G}$ is chosen uniformly at random, and it is stored in $\mathsf{VK}_f = (\alpha, Z_1, \ldots, Z_d)$. Accordingly, all verification queries $\mathcal{O}_{\mathsf{Verify}}(f, x, \sigma_y)$ where f was asked to $\mathcal{O}_{\mathsf{KeyGen}}(\cdot)$ in the first j queries, are answered by checking whether $V = g^{\alpha y} \cdot \prod_{i=1}^d Z_i^{x_i}$. Notice that Game 1 is Game 2,0.

Game 3: for ease of exposition Game 3 is only a renaming of Game 2,q.

Claim. $\Pr[G_0(\mathcal{A}) = 1] = \Pr[G_1(\mathcal{A}) = 1]$.

Proof. The only difference between the two games is in the computation made when answering verification queries. However, by the homomorphic property of the weak PRF, this computation is equivalent, and it does not affect at all the distribution of the outcome of verification. Thus, the probability of the adversary winning in Game 1, $\Pr[G_1(\mathcal{A}) = 1]$, remains the same.

Claim. For any $j = 1$ to q, we have $|\Pr[G_{2,j-1}(\mathcal{A}) = 1] - \Pr[G_{2,j}(\mathcal{A}) = 1]| \leq \epsilon_{wprf}$

Proof. For any two consecutive games Game 2,j-1 and Game 2,j, their difference is basically that in Game 2,j, we replaced with a random function, the weak pseudorandom function G_k that is generated in the j-th query to $\mathcal{O}_{\mathsf{KeyGen}}$. A proof of the claim can be obtained by a straightforward reduction to the security of the weak PRF.

Claim. $\Pr[G_3 = 1] \leq q/p$.

The proof of this claim is a simple extension of the proof of Claim 5.6 in [4]. The only difference is that in our game the adversary may ask to generate keys for many functions f. However, the run of the modified KeyGen for each of these queries is a completely independent process, as a new fresh tuple $\alpha, Z_1, \ldots, Z_d$ is chosen every time.

Efficiency. The Setup, KeyGen, and $\mathsf{ProbGen}$ algorithms run in time $O(d)$, whereas the complexity of verification is independent of d, i.e., $O(1)$. Our protocol $\mathcal{VC}_{MultiF}$ achieves amortized efficiency in a setting in which the delegator computes several functions $f_1, \ldots, f_n$ on some a-priori fixed inputs $x_1, \ldots, x_m$. While the cost of computing a single function on m inputs would be $O(dm)$, using our scheme, a client can outsource and verify the computation of $f(x_1), \ldots, f(x_m)$ in time $O(d + m)$.

B Discrete Fourier Transform

The Discrete Fourier Transform (DFT) of an n-dimensional vector $\boldsymbol{f}$ is defined as $\boldsymbol{y} = [f(x_1), \ldots, f(x_n)]$ where f is the $(n-1)$-degree polynomial naturally defined by $\boldsymbol{f}$ evaluated over the n roots of unity. It is well known that using the Fast Fourier Transform algorithm (FFT), the DFT can be computed in $O(n \log n)$ time. Note that the vector $\boldsymbol{y}$ can alternatively be computed as $\boldsymbol{y} = X \cdot \boldsymbol{f}$ where X is the $n \times n$ *Vandermonde* matrix defined by the x_i's.

To obtain a verifiable computation scheme for the DFT using our protocols we have three options.

Using the Matrix Multiplication Scheme. This solution works well for the client as it incurs $O(n)$ delegation and verification cost. However, the computation at the server is $O(n^2)$: indeed, the server can compute the DFT in $O(n \log n)$ but to compute the verification value $\boldsymbol{V}$ it cannot use the FFT algorithm (as the matrix W is a pseudo-random matrix without the "special" structure such as X). Therefore with this approach the server's computation is asymptotically higher and this could be unacceptable for large n.

Using the Polynomial Evaluation scheme over n inputs. To outsource the DFT of $\boldsymbol{f}$ the client would perform the key generation part of our polynomial evaluation scheme, and the delegation of the input x_i's all in one blow. The only way this can be done in $O(n)$ time is to use a closed-form PRF that allows the delegation of a single variable input in constant time. Unfortunately, our PRFs from Decision Linear require $O(\log n)$ time to delegate a single input. We can however use the constant time scheme from [4] adapted to be publicly verifiable. The drawback is that we need to assume the External n-SDH Assumption over asymmetric bilinear groups, which is a strong "non-constant size" assumption.

Using the Multi-Function Scheme. Finally, the Multi-Function Scheme described in the previous section satisfies the efficiency constraint given by FFT, but is only privately verifiable.