Efficiently Verifiable Computation on Encrypted Data

Dario Fiore[1], Rosario Gennaro[2], and Valerio Pastro[2]

[1] IMDEA Software Institute
dario.fiore@imdea.org
[2] Center for Algorithms and Interactive Scientific Software, The City College of New York
{rosario,pastro}@cs.ccny.cuny.edu

Abstract. We study the task of efficient verifiable delegation of computation on encrypted data. First, we improve previous definitions in order to tolerate adversaries that learn whether or not clients accept the result of a delegated computation. Then, in this strong model, we show a scheme for arbitrary computations, and we propose highly efficient schemes for delegation of various classes of functions, such as linear combinations, high-degree univariate polynomials, and multivariate quadratic polynomials. Notably, the latter class includes many useful statistics. Using our solution, a client can store a large encrypted dataset with a server, query statistics over this data, and receive encrypted results that can be efficiently verified and decrypted.

As a key contribution for the efficiency of our schemes, we develop a novel homomorphic hashing technique that allows us to efficiently authenticate computations, at the same cost as if the data were in the clear, avoiding a 10^4 overhead, which would occur with a naive approach. We confirm our theoretical analysis with extensive implementation tests that show the practical feasibility of our schemes.

Table of Contents

1 Introduction . . . 3
1.1 Our Contribution . . . 3
1.2 Other Related Work . . . 6
2 Problem Definition . . . 8
3 A Generic Solution . . . 11
3.1 Homomorphic Encryption . . . 11
3.2 The Generic Scheme . . . 12
4 Tools . . . 14
4.1 The BGV Homomorphic Encryption . . . 14
4.2 Homomorphic Hash Functions . . . 16
4.3 Amortized closed-form Efficient Pseudorandom Functions . . . 18
5 Computing Multi-Variate Polynomials of Degree 2 . . . 21
6 Computing Polynomials of Large Degree . . . 27
7 Computing Linear Combinations . . . 29
8 Computing Linear Functions over the ring $\mathbb{Z}_{2^k}$ . . . 31
9 Applications . . . 33
9.1 Statistics on Encrypted Data Sets . . . 33
9.2 Distance and Correlation Measures on Encrypted Data Sets . . . 35
9.3 Discrete Fourier Transform . . . 35
10 Experimental Evaluation . . . 36
10.1 Setup . . . 36
10.2 Timings . . . 36

1 Introduction

Can an outside party compute for us, without learning our private data? Can we efficiently check that it performed the computation correctly? These are some central questions related to the *privacy* and the *security* of cloud computing, a paradigm where businesses buy computational time from a service, rather than purchase and maintain their own resources. Both questions have a long research history.

Computing arbitrary functions on encrypted data was a research interest recognized very early by cryptographers [50], and it remained open until Gentry's construction of the first fully homomorphic encryption (FHE) scheme [28]. Gentry's work revealed a new set of techniques that were immediately used for many more efficient schemes [29, 53, 30, 19, 18, 31, 17]; today we seem to stand at the verge of having FHE schemes which can be used in practice.

Efficient verification of arbitrarily complex computations was the underlying goal of interactive proofs [3, 36], where a powerful (e.g. super-polynomial) prover can (probabilistically) convince a weak (e.g. polynomial) verifier of the truth of statements that the verifier could not compute on its own. Research on making the verifier as efficient as possible led to the concept of probabilistically checkable proofs (PCPs) [2, 5, 4], and their cryptographic applications: the arguments of Kilian [41, 42] and Micali's non-interactive CS Proofs [46].

The application to cloud computing has rekindled attention to this area, particularly in the search for protocols where arbitrary poly-time (as opposed to superpoly-time) computations can be efficiently verified by a linear (or quasi-linear) verifier, and performed by a prover without too much overhead. Starting with the work on proofs for muggles [34], a line of research revisited and "scaled down" the PCP machinery [35, 12, 13]. Another line of work explored alternative ways of arithmetizing computations to construct efficient proofs [37, 44, 26]. Yet another approach used FHE as a tool to build efficient verification of arbitrary computations [25, 1, 22]. Several implementational efforts [23, 55, 47, 10] show that in this area, too, we are on the verge of achieving practical efficiency, with the quadratic span program (QSP) techniques of [26, 47] showing particular promise.

1.1 Our Contribution

Given the practical and theoretical relevance of this topic, as well as the level of maturity of the field, it is somewhat surprising to notice that most of the research was focused on solving either one of the two main questions. There are many results about finding efficient FHE schemes, and therefore efficient computation on encrypted data, but without verification of its correctness. On the other hand, the works on verifying computation mostly focused on the case where the data is in the clear or in a restricted model for privacy – a notable exception is the construction of Goldwasser et al. [33], based on functional encryption.

The protocols for verifiable computation (VC) in [25, 22, 1] use FHE as a tool for verifiability, and almost as a by-product, they achieve data privacy as well, but in a restricted sense: namely, only if the client's acceptance bit is kept hidden from the server. Also, in [25] only the topology of the function is revealed [9]. In [25], a formal definition of all the properties needed by a VC scheme is given, including input privacy – in this very weak model in which the server is not allowed to issue *verification queries* to the client.

This model is quite restrictive in practice, as the client's acceptance bit may leak due to multiple reasons: error messages, protocol termination, recomputation, etc.

To the best of our knowledge, no definition of private and secure outsourced computation in the presence of verification queries can be found in the literature. We address this somewhat problematic

state of affairs, provide constructions that satisfy the new definitions, and give experimental results for our schemes. Our contribution, in a list:

1. An upgrade to the definition in [25] that allows verification queries by the adversary, and an extension of this definition that models adaptive security and function privacy. This is the strongest possible model in which we can define security.
2. A protocol based on FHE for arbitrary computations.
3. Constructions for specific families of functions to compute: multivariate quadratic polynomials, univariate polynomials of high degree, linear combinations, all on encrypted data (details on Table 1). Using our schemes a client can outsource the storage of large, privacy-sensitive, data sets (e.g. location, medical, genomic data) on a server, and ask the server to compute statistics or distance measures on these data sets, with guarantees of correctness of the computation and privacy of the data.
4. A key technical contribution is the introduction of a homomorphic hashing technique that allowed us to obtain improvements of more than *four* orders of magnitude, compared to a naive approach.
5. Implementations and performance numbers of our schemes for practical security parameters.

In the following we discuss our contribution and the related work in more in detail.

Our Generic Protocol. Our protocol assumes the existence of an FHE scheme and a not (necessarily) private, but secure VC scheme, e.g. [26]. The basic idea is to encrypt the data x with the FHE scheme, and to run the VC scheme on the function $\mathsf{Eval}_{\mathsf{FHE}}(f)$, *instead of* f, and on input $\mathsf{Enc}_{\mathsf{FHE}}(x)$ *instead of* x. This technique prevents the server from using the client as a decryption oracle for the FHE, since the acceptance bit is determined before decryption, just according to the correctness of the evaluation of $\mathsf{Eval}_{\mathsf{FHE}}(f)$.

Ad-Hoc Protocols. We focus on the problem where a client stores a large data set $x = (x_0, \ldots, x_t)$ on a server and later asks the server to compute functions f over x. Our solutions work in the *amortized model* in which the client spends a single pre-processing phase whose cost is as running $f(x)$, and later amortizes this one-time cost over several function evaluations. Moreover, our protocols work also in the so-called *streaming model*: clients can process and outsource single data items x_i in a separate fashion. This is desirable for storing applications, since it enables clients to work with a very small memory, independent of the size of the (possibly huge) data set. As an example, think of a set of weak devices that read the temperature in given locations, and daily send the data to a server to store for later analysis (e.g. calculate the mean temperature in a year).

We focus on the case where the data sets consist of elements in a finite field $\mathbb{F}_p$, and the outsourced functions can be: (1) quadratic multi-variate polynomials, (2) univariate polynomials of large degree, and (3) linear combinations.

Multi-Variate Quadratic Polynomials. We constructed a VC scheme in the case the client stores several encrypted data sets at the server, and then asks it to compute a quadratic polynomial f on any of the outsourced sets. In our solution, after a single pre-processing for every f, the client can verify results in *constant* time. Moreover, our scheme achieves input and output privacy.

In terms of applications, this scheme allows to compute several statistics (average, variance, covariance, RMS, linear regression, correlation coefficient, and many more) on remotely stored data sets in a private and verifiable manner. We also consider the application in which a client stores a large matrix X on the server, and then asks to compute a distance measure (e.g. Euclidean distance) between a given vector y and any row of X. We propose a variant of our scheme that is function private for a restricted class of quadratic polynomials (privacy holds only for the coefficients of the

Scheme	Input Priv.	Function Priv.	Amort. Verif.
Linear combinations t-variables over $\mathbb{F}_p$	✓	✓	$O(t)$
Linear combinations t-variables over $\mathbb{Z}_{2^k}$	✓	×	$O(1)$
Univariate Poly of degree t over $\mathbb{F}_p$	×	✓	$O(1)$
t-variate Poly of degree 2 over $\mathbb{F}_p$	✓	×	$O(1)$
t-variate Restricted Poly of degree 2 over $\mathbb{F}_p$	✓	✓	$O(1)$

Table 1. Summary of our schemes.

linear terms). Yet, since the above application fits such restriction, we give a solution in which both X and y are private.

We are not aware of any other existing solution for privately evaluating multivariate quadratic polynomials, except by instantiating our paradigm with existing tools [18, 6]. Even compared to these solutions, our experiments show that our ad-hoc protocol improves significantly: for instance, in our scheme the computation of the variance function at the server is more than 10^4 times faster! If we consider cloud computing, in which clients pay for the server's CPU cycles, such improvement leads to worthwhile savings.

Polynomials of High Degree. We also constructed a VC scheme for a setting which is complementary to the one above. Namely, we think of x as the coefficients of a *uni-variate* polynomial $P_x(z) = \sum_{j=0}^{t} x_i z^i$ of degree t. With our protocol the client stores the large polynomial $\mathsf{Enc}(P_x)$ at the server, and then asks it to compute $\mathsf{Enc}(P_x(z))$ on many different points z (provided in the clear). Here, after the single preprocessing to outsource P_x, the client can verify all the computations in *constant* time. Moreover, our scheme is function private as P_x is encrypted.

In terms of applications, we discuss how this solution can be used to outsource the Discrete Fourier Transform computation on encrypted vectors.

Linear Combinations. Finally, we considered the task of outsourcing linear combinations. For this we constructed a very clean and efficient solution which provides both input and function privacy, but has no efficient verification. However, the client can work with a very short memory in the streaming model. We note that a solution with efficient verification (achieving both input and function privacy) can be obtained by using our variant scheme for quadratic polynomials. Compared to the latter, the advantage of our dedicated scheme is efficiency: we achieve verifiability by using information-theoretic techniques that do not require expensive, cryptographic computations (e.g. over bilinear groups).

An Overview of Our Techniques. To design our protocols we follow the blueprint of our generic scheme and we develop additional techniques that provide significant efficiency improvements. Our basic idea is to encrypt the data with a somewhat homomorphic encryption scheme (for privacy), and to add an authentication mechanism on top of the ciphertexts (for security). For the encryption, we chose a simplified version of the scheme by Brakerski, Gentry, and Vaikuntanathan (BGV) [18]. The server stores $\mu = (\mu_0, \ldots, \mu_t)$ where μ_i is a BGV encryption of x_i. For authentication, we rely on homomorphic MACs [27]. In a nutshell, this primitive enables a client to use a secret key

to authenticate a set of messages $m_1, \ldots, m_t$ by producing corresponding tags $\sigma_1, \ldots, \sigma_t$. Later, given such tags, anyone (without any secret key) can produce a tag σ that authenticates $m = f(m_1, \ldots, m_t)$ as the output of f on the previously authenticated inputs. Interestingly, verification can be performed without knowing the original messages, and recently proposed schemes [6] allow to verify more efficiently than running f. The generic idea for our schemes is to generate a MAC σ_i for every ciphertext μ_i, and then use the homomorphic property of the MAC to authenticate the BGV homomorphic evaluation. While this idea may work for appropriate choices of the MAC scheme (e.g. [6]), we note that BGV ciphertexts consist of several components over a field $\mathbb{F}_q$: in detail, if $f : \mathbb{F}_p^t \to \mathbb{F}_p$ is the desired function, the BGV evaluation circuit is a function $\hat{f} : \mathbb{F}_q^{2nt} \to \mathbb{F}_q^{3n}$. We improve this situation using our key technical contribution: *homomorphic hash functions* H_κ that allow to compress a BGV ciphertext into a *single* $\mathbb{F}_q$-component, while preserving the homomorphic properties, i.e. $f(H_\kappa(\mu_1), \ldots, H_\kappa(\mu_t)) = H_\kappa(\hat{f}(\mu_1, \ldots, \mu_t))$. By applying a MAC on top of the hashed ciphertexts, we save at least a factor of $3n$ in all operations (e.g. input outsourcing, computation, verification). Considering that for security reasons (and technical details discussed later) n can be as large as 5000, applying the homomorphic hash leads to schemes that are up to *four* orders of magnitude faster. Compare the last two columns of Table 2 for the concrete example of computing the variance of 1000 items.

	Without Privacy [6]	**Ours (Sec.5)**	**Naive Approach**
Server Computation	0.98sec	1.11sec	15210.61sec
Client Verification	0.21ms	0.42ms	3316.63ms

Table 2. Comparision of the amortized costs for calculating the variance in a database of $t = 1000$ entries. Security in the second and third column is 80bit.

Experiments. We implemented the above solutions, and tested their practical performances. Remarkably, all our protocols run considerably fast. See section 10 for details.

An interesting point is that the cost (in terms of CPU time) of adding privacy to verifiability is relatively small. As an example, for 80bit security, scheme in section 5: on the server side, the total execution time of any operation, is between 1.1 and 2.2 times the execution time for authentication operations only (i.e. excluding encryptions, FHE evaluations, etc. from the total cost), depending on the particular function. The highest privacy cost (defined as: total execution time over authentication time, as above), 2.9, occurs in one-time operations: encrypting and authenticating the data when loading it into the server. Finally, our verification algorithm has a privacy cost of $1.9\times$, due to decryption of the result. Compare the first two columns of Table 2 for a concrete example, and see Table 4 for more details. Notice that our timing for authentication only is comparable with the timing of [6] (adapted with a secure pairing).

Finally, we point out that, even if minimal, the privacy overhead can be mitigated by using the batching technique of BGV: for 80bit (resp. 128 bit) security we could encrypt 165 (resp. 275) 32bit plaintext items in a single ciphertext. This means the amortized cost (per plaintext item) of our scheme is actually better than [6].

1.2 Other Related Work

Generic Protocols. Another generic protocol for private and verifiable outsourced computation appears in [33], based on functional encryption (FE). In the introduction of [33] it is shown how to

use an FE scheme to construct a VC scheme that operates on encrypted inputs and is secure even in the presence of verification queries; in particular it is publicly verifiable – anybody can verify the correctness of the computation – and this property is achieved using an idea that originates in [48], for the case of attribute-based encryption.

We point out that the above construction is only sketched in [33], together with an intuitive motivation for its security, but no formal definition or proof is given. When one formally analyzes the protocol in [33] in light of our definition the following issues come up.

- As discussed in [16], a simulation-based definition of security for FE cannot be achieved in general in an "adaptive" model in which the adversary chooses the function f to be evaluated after seeing the ciphertext encoding x. It appears that this limitation might be inherited by the FE-based verifiable computation in [33], though the situation is far from clear. Our security definition is a game-based one, but the security proof might require the FE tool to be simulatable. A proof of the [33] protocol in this model is important, since in many cloud computing application the data might sit encrypted on a cloud server, before any processing is performed on it. In contrast, we can provably show that our proposed protocol does not have this limitation;
- Additionally, the protocol in [33] only achieves selective security (the adversary has to commit in advance to the input value on which it wants to cheat), essentially because known constructions of FE [33, 24] achieve only selective security against poly-time adversaries, and full security can be obtained at the cost of assuming security against sub-exponential adversaries: a much stronger assumption. Our protocol has the advantage of being full secure against poly-time adversaries.
- Finally, the scheme in [33] intrinsically works for binary functions. While this is sufficient in theory, in practice it requires the overhead of working bit-by-bit on the function output. Our protocol is more versatile and potentially more efficient, since it can handle arithmetic circuits if the underlying FHE and VC schemes can (several FHE schemes work over arithmetic circuits, and the QSP/QAP approach in [26] yields an efficient VC over arithmetic circuits).

An issue on the security of the scheme in [25] was noted by Bellare et al. in [8] where the authors investigate the notion of adaptive garbling schemes, and show that under the existence of these schemes (realized later in [7]) the construction of [25] becomes secure in the presence of verification queries. Compared to [8], we fully formalize the notion of security in the presence of verification queries and, in particular, propose a stronger notion of *adaptive* security that allows the adversary to obtain encodings of the input even before choosing the function.

Ad-Hoc Protocols. The task of evaluating univariate polynomials of large degree was earlier considered in [11]. However, in that protocol, when the polynomial is encrypted, the client's acceptance bit depends on the decrypted value, which creates the opportunity for a verification query attack. In contrast, our solution (which builds on [11] in a slightly different way) enjoys security in the stronger model where verification queries are allowed.

In [43] Libert et al. propose a protocol for the evaluation of linear combinations over encrypted data using additively homomorphic encryption and structure-preserving linearly homomorphic signatures. However, their protocol is restrictive, since the linear combinations have to reside in a very small range in order for the client to retrieve the correct result: the client's decryption consists in solving discrete log, by which the exponents must be small. Compared to this work, our solutions for linear combinations can support large domains and additionally provide function privacy.

The task of verifying computations on encrypted data has also been considered in [39, 21] via the notion of homomorphic authenticated encryption. In a nutshell, this primitive enables a client to encrypt a set of data using a secret key in such a way that anyone can then execute functions on

this data obtaining encrypted results whose correctness can be verified. In particular, [39] proposes a construction for the evaluation of low-degree polynomials which is secretly verifiable, while [21] proposes a scheme for linear functions that achieves public verifiability. However, we note that both these constructions [39, 21] are *not* outsourceable, in the sense that verifying each computation is as expensive as running the delegated function.

2 Problem Definition

We work in the amortized model of [25] where the client runs a one-time expensive phase to outsource the function f to the server (this phase can cost as much as the computation of f). Later the client queries the server on (an encrypted form of) input x and receives back (an encryption of) the value $f(x)$ and a proof of its correctness: this phase should be efficient for the client (ideally linear in $|x| + |f(x)|$).

In [25] the authors give a definition that includes both security (i.e. the client only accepts correct outputs) and privacy (i.e. the client's input x is semantically hidden to the server) but does *not* allow verification queries. In this section we upgrade the definition by adding verification queries to it. Moreover, we introduce the concept of adaptive security.

A *verifiable computation scheme* $\mathcal{VC} = (\mathbf{KeyGen}, \mathbf{ProbGen}, \mathbf{Compute}, \mathbf{Verify})$ consists of the following algorithms:

- $\mathbf{KeyGen}(f, \lambda) \to (PK, SK)$: Based on the security parameter λ, the randomized key generation algorithm generates a public key (that encodes the target function f) which is used by the server to compute f. It also computes a matching secret key, kept private by the client.
- $\mathbf{ProbGen}_{SK}(x) \to (\sigma_x, \tau_x)$: The problem generation algorithm uses the secret key SK to encode the input x as a public value σ_x which is given to the server to compute with, and a secret value τ_x which is kept private by the client.
- $\mathbf{Compute}_{PK}(\sigma_x) \to \sigma_y$: Using the client's public key and the encoded input, the server computes an encoded version of the function's output $y = f(x)$.
- $\mathbf{Verify}_{SK}(\tau_x, \sigma_y) \to (acc, y)$: Using the secret key SK and the secret τ_x, the verification algorithm converts the server's output into a bit acc and a string y. If $acc = 1$ we say the client accepts $y = f(x)$, if $acc = 0$ we say the client rejects.

We now recall the three main properties defined in [25] for a verifiable computation scheme: correctness, security, privacy, and outsourceability, but we define them *in the presence of verification queries* by the adversary. Next, we introduce function privacy, that is the ability of a scheme to hide from the server the function that it needs to compute. Finally we move to adaptive security

A scheme is correct if the problem generation algorithm produces values that allow a honest server to compute values that will verify successfully and correspond to the evaluation of f on those inputs. More formally:

Definition 1 (Correctness). *A verifiable computation scheme $\mathcal{VC}$ is* correct *if for all f, x: if*

- $(PK, SK) \leftarrow \mathbf{KeyGen}(f, \lambda)$,
- $(\sigma_x, \tau_x) \leftarrow \mathbf{ProbGen}_{SK}(x)$, *and*
- $\sigma_y \leftarrow \mathbf{Compute}_{PK}(\sigma_x)$,

then $(1, y = f(x)) \leftarrow \mathbf{Verify}_{SK}(\tau_x, \sigma_y)$.

For the other notions, we need to define the following oracles.

- $\mathbf{PProbGen}(x)$ calls $\mathbf{ProbGen}_{SK}(x)$ to obtain (σ_x, τ_x) and returns only σ_x.

– $\mathbf{PVerify}(\tau, \sigma)$ returns acc if and only if $\mathbf{Verify}_{SK}(\tau, \sigma) = (acc, y)$. In other words, **PVerify** is the public acceptance/rejection bit which results from a verification query. When we write $A^{\mathbf{PVerify}}$ we mean that A is allowed to query $\mathbf{PVerify}(\tau, \cdot)$ where τ can be the secret encoding of any of the queries made in **PProbGen**, or also τ_b in the case of the privacy experiments.

Intuitively, a verifiable computation scheme is secure if a malicious server cannot persuade the verification algorithm to accept an incorrect output.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{Verif}[\mathcal{VC}, f, \lambda]$
$(PK, SK) \leftarrow \mathbf{KeyGen}(f, \lambda)$;
For $i = 1, \ldots, \ell = \mathsf{poly}(\lambda)$;
$x_i \leftarrow \mathcal{A}^{\mathbf{PVerify}}(PK, x_1, \sigma_1, \ldots, x_{i-1}, \sigma_{i-1})$;
$(\sigma_i, \tau_i) \leftarrow \mathbf{ProbGen}_{SK}(x_i)$;
$(i, \hat{\sigma}_y) \leftarrow \mathcal{A}^{\mathbf{PVerify}}(PK, x_1, \sigma_1, \ldots, x_\ell, \sigma_\ell)$;
$(\hat{acc}, \hat{y}) \leftarrow \mathbf{Verify}_{SK}(\tau_i, \hat{\sigma}_y)$
If $\hat{acc} = 1$ and $\hat{y} \neq f(x_i)$, output '1', else '0';

Essentially, the adversary is given oracle access to generate the encoding of multiple problem instances, and to check the response of the client on arbitrary "encodings". The adversary succeeds if it produces an output that convinces the verification algorithm to accept on the wrong output value for a given input value. We can now define the security of the system based on the adversary's success in the above experiment.

Definition 2 (Security). *A verifiable computation scheme $\mathcal{VC}$ is* secure *for a function f, if for any adversary $\mathcal{A}$ running in probabilistic polynomial time,*

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{Verif}[\mathcal{VC}, f, \lambda] = 1] \leq \mathsf{negl}(\lambda).$$

We say that $\mathcal{VC}$ is secure if it is secure for every function f.

Input privacy is defined based on a typical indistinguishability argument that guarantees that no information about the inputs is leaked. Input privacy, of course, immediately yields output privacy.

Intuitively, a verifiable computation scheme is *private* when the public outputs of the problem generation algorithm **ProbGen** over two different inputs are indistinguishable(i.e., nobody can decide which encoding is the correct one for a given input). More formally, consider the following experiment: the adversary is given the public key for the scheme and selects two inputs x_0, x_1. He is then given the encoding of a randomly selected one of the two inputs and must guess which one was encoded. During this process the adversary is allowed to request the encoding of any input he desires, and also is allowed to make verification queries on any input. The experiment is described below.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{Priv}[\mathcal{VC}, f, \lambda]$
$b \leftarrow \{0, 1\}$;
$(PK, SK) \leftarrow \mathbf{KeyGen}(f, \lambda)$;
$(x_0, x_1) \leftarrow \mathcal{A}^{\mathbf{PVerify}, \mathbf{PProbGen}}(PK)$
$(\sigma_0, \tau_0) \leftarrow \mathbf{ProbGen}_{SK}(x_0)$;
$(\sigma_1, \tau_1) \leftarrow \mathbf{ProbGen}_{SK}(x_1)$;
$\hat{b} \leftarrow \mathcal{A}^{\mathbf{PVerify}, \mathbf{PProbGen}}(PK, x_0, x_1, \sigma_b)$
If $\hat{b} = b$, output '1', else '0';

Definition 3 (Privacy). *A verifiable computation scheme $\mathcal{VC}$ is* private *for a function f, if for any adversary $\mathcal{A}$ running in probabilistic polynomial time,*

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{Priv}[\mathcal{VC}, f, \lambda] = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Function privacy is the requirement that the public key PK, sampled via $(PK, SK) \leftarrow \mathbf{KeyGen}(f, \lambda)$, does not leak information on the encoded function f, even after a polynomial amount of runs of $\mathbf{ProbGen}_{SK}$ on adversarially chosen inputs. More formally, we define function privacy based on an indistinguishability experiment as follows.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{FPriv}[\mathcal{VC}, \lambda]$
- $(f_0, f_1) \leftarrow \mathcal{A}(\lambda)$;
- $b \leftarrow \{0, 1\}$;
- $(PK, SK) \leftarrow \mathbf{KeyGen}(f_b, \lambda)$;
- For $i = 1, \ldots, \ell = \mathsf{poly}(\lambda)$;
  - $x_i \leftarrow \mathcal{A}^{\mathbf{PVerify}}(PK, x_1, \sigma_1, \ldots, x_{i-1}, \sigma_{i-1})$;
  - $(\sigma_i, \tau_i) \leftarrow \mathbf{ProbGen}_{SK}(x_i)$;
- $\hat{b} \leftarrow \mathcal{A}^{\mathbf{PVerify}}(PK, x_1, \sigma_1, \ldots, x_\ell, \sigma_\ell)$;
- If $\hat{b} = b$, output '1', else '0';

Definition 4 (Function Privacy). *A verifiable computation scheme $\mathcal{VC}$ is* function private*, if for any adversary $\mathcal{A}$ running in probabilistic polynomial time,*

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{FPriv}[\mathcal{VC}, \lambda] = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

The next property of a verifiable computation scheme is that the time to encode the input and verify the output must be smaller than the time to compute the function from scratch.

Definition 5 (Outsourceability). *A $\mathcal{VC}$ can be outsourced if it allows efficient generation and efficient verification. This implies that for any x and any σ_y, the time required for $\mathbf{ProbGen}_{SK}(x)$ plus the time required for $\mathbf{Verify}(\sigma_y)$ is $o(T)$, where T is the time required to compute $f(x)$.*

We now introduce the notion of *adaptive security* for a verifiable computation scheme. Intuitively, an adaptively secure scheme is a scheme that is secure even if the adversary chooses f *after* having seen many "encodings" σ_x for adaptively-chosen values x. At first sight, this property is non-trivial to achieve, since not every scheme allows σ_x to be computed before choosing f (in particular schemes based on FE such as [33]). This observation leads us to first define a refined class of schemes, for which adaptivity is not ruled out by this restriction, and then proceed with the actual definition of adaptivity.

Definition 6 (Split Scheme). *Let $\mathcal{VC} = (\mathbf{KeyGen}, \mathbf{ProbGen}, \mathbf{Compute}, \mathbf{Verify})$ be a verifiable computation scheme. We say that $\mathcal{VC}$ is a* split scheme *if the following conditions hold:*

- *There exist PPT algorithms $\mathbf{KeyGen}^E(\lambda)$, $\mathbf{KeyGen}^V(f, \lambda)$ such that:*
  if $(PK, SK) \leftarrow \mathbf{KeyGen}(f, \lambda)$, then
  $PK = (PK_E, PK_V)$ and $SK = (SK_E, SK_V)$,
  where $\mathbf{KeyGen}^E(\lambda) \rightarrow (PK_E, SK_E)$ and $\mathbf{KeyGen}^V(f, \lambda, PK_E, SK_E) \rightarrow (PK_V, SK_V)$.
- *There exist PPT algorithms $\mathbf{ProbGen}_{SK_E}^E(x)$, $\mathbf{ProbGen}_{SK_V}^V(x)$ such that:*
  if $(\sigma_x, \tau_x) \leftarrow \mathbf{ProbGen}_{SK_E, SK_V}(x)$, then
  $\sigma_x = \sigma_x^E \leftarrow \mathbf{ProbGen}_{SK_E}^E(x)$ and $\tau_x = \tau_x^V \leftarrow \mathbf{ProbGen}_{SK_V}^V(x)$.

Notice that for a split scheme one can generate valid values σ_x for any function f to be delegated *before* knowing f, since σ_x is independent of f. This can be done by running $(PK_E, SK_E) \leftarrow \mathbf{KeyGen}^E(\lambda)$, and setting $\sigma_x \leftarrow \mathbf{ProbGen}^E_{SK_E}(x)$ before knowing f. The validity of this encoding applies for all keys $(PK, SK) = (PK_E, PK_V, SK_E, SK_V)$ where $(PK_V, SK_V) \leftarrow \mathbf{KeyGen}^V(f, \lambda)$ for *any* f.

We can now describe the experiment that is used to define adaptive security for split schemes.

Experiment $\mathbf{Exp}^{Adap-Verif}_{\mathcal{A}}[\mathcal{VC}, \lambda]$
$(PK_E, SK_E) \leftarrow \mathbf{KeyGen}^E(\lambda)$;
For $i = 1, \ldots, \ell' = \mathsf{poly}'(\lambda)$:
$x'_i \leftarrow \mathcal{A}(PK_E, x'_1, \sigma'_1, \ldots, x'_{i-1}, \sigma'_{i-1})$;
$\sigma'_i \leftarrow \mathbf{ProbGen}^E_{SK_E}(x)$;
$f \leftarrow \mathcal{A}(x'_1, \sigma'_1, \ldots, x'_{\ell'}, \sigma'_{\ell'})$;
$(PK_V, SK_V) \leftarrow \mathbf{KeyGen}^V(f, \lambda)$;
$(PK, SK) \leftarrow (PK_E, PK_V, SK_E, SK_V)$;
For $i = 1, \ldots, \ell = \mathsf{poly}(\lambda)$:
$x_i \leftarrow \mathcal{A}^{\mathbf{PVerify}}(PK, x_1, \sigma_1, \ldots, x_{i-1}, \sigma_{i-1})$;
$(\sigma_i, \tau_i) \leftarrow \mathbf{ProbGen}_{SK}(x_i)$;
$(i, \hat{\sigma}_y) \leftarrow \mathcal{A}^{\mathbf{PVerify}}(PK, x_1, \sigma_1, \ldots, x_\ell, \sigma_\ell)$;
$(\hat{acc}, \hat{y}) \leftarrow \mathbf{Verify}_{SK}(\tau_i, \hat{\sigma}_y)$;
If $\hat{acc} = 1$ and $\hat{y} \neq f(x_i)$, output '1', else '0'.

Definition 7 (Adaptive Security). *A split scheme $\mathcal{VC}$ is* adaptively secure*, if for any adversary $\mathcal{A}$ running in probabilistic polynomial time,*

$$\Pr[\mathbf{Exp}^{Adap-Verif}_{\mathcal{A}}[\mathcal{VC}, \lambda] = 1] \leq \mathsf{negl}(\lambda).$$

3 A Generic Solution

In this section we describe our generic solution to outsource computation over encrypted data.

As we discussed in the introduction, where we give an intuitive idea about how our scheme works, we assume the existence of a FHE scheme, and a VC scheme to outsource the computation of generic functions, secure even in the presence of verification queries. The latter can be achieved using a variety of schemes, e.g. [26, 35, 12, 13].

Before proceeding with the actual construction, we give a description of the requirements we ask for an FHE scheme

3.1 Homomorphic Encryption

A *fully homomorphic (public-key) encryption* (FHE) scheme is a tuple of PPT algorithms $\mathsf{FHE} = (\mathsf{FHE.ParamGen}, \mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$ defined as follows.

$\mathsf{FHE.ParamGen}(\lambda)$**.** Define the parameters for the scheme, such as plaintext space $\mathcal{M}$, ciphertext space, keyspace, randomness distributions, etc. The output of $\mathsf{ParamGen}$ is assumed to be input to any subsequent algorithm.

$\mathsf{FHE.KeyGen}(\lambda) \rightarrow (\mathsf{pk}, \mathsf{evk}, \mathsf{dk})$**.** Output a public encryption key pk, a public evaluation key evk, and a secret decryption key dk.

$\mathsf{FHE.Enc}_{\mathsf{pk}}(m) \rightarrow c$**.** Encrypt message $m \in \mathcal{M}$ under public key pk. Outputs ciphertext c.

$\mathsf{FHE.Dec}_{\mathsf{dk}}(c) \rightarrow b$**.** Decrypt ciphertext c using dk to a plaintext bit $m \in \mathcal{M}$.

$\mathsf{FHE.Eval}_{\mathsf{evk}}(g, c_1, \ldots, c_t) \to c^*$**.** Given the evaluation key evk, a circuit $g : \mathcal{M}^t \to \mathcal{M}$, and a set of t ciphertexts $c_1, \ldots, c_t$, deterministically compute and output a ciphertext c^*.

An FHE should also satisfy the following properties.

Encryption Correctness. For all $m \in \mathcal{M}$ we have:

$$\Pr\left[\mathsf{FHE.Dec}_{\mathsf{dk}}(\mathsf{FHE.Enc}_{\mathsf{pk}}(m)) = m \mid (\mathsf{pk}, \mathsf{evk}, \mathsf{dk}) \xleftarrow{\$} \mathsf{FHE.KeyGen}(\lambda)\right] = 1.$$

Evaluation Correctness. For $(\mathsf{pk}, \mathsf{evk}, \mathsf{dk}) \xleftarrow{\$} \mathsf{FHE.KeyGen}(\lambda)$, any ciphertexts $c_1, \ldots, c_t$ such that $\mathsf{FHE.Dec}_{\mathsf{dk}}(c_i) = m_i \in \mathcal{M}$, and any circuit $g : \mathcal{M}^t \to \mathcal{M}$, we have

$$\mathsf{FHE.Dec}_{\mathsf{dk}}(\mathsf{FHE.Eval}_{\mathsf{evk}}(g, c_1, \ldots, c_t)) = g(m_1, \ldots, m_t).$$

Succinctness. The ciphertext size is bounded by some fixed polynomial in the security parameter, and is independent of the size of the evaluated circuit or the number of inputs it takes. I.e. there exists some polynomial p such that, for any $(\mathsf{pk}, \mathsf{evk}, \mathsf{dk}) \xleftarrow{\$} \mathsf{FHE.KeyGen}(\lambda)$, the output size of $\mathsf{FHE.Enc}_{\mathsf{pk}}$ and of $\mathsf{Eval}_{\mathsf{evk}}$ is bounded by p, for any choice of their inputs.

Semantic Security. An FHE is a *semantically secure* public-key encryption scheme, where we consider the evaluation key evk as a part of the public key. I.e. for any PPT attacker $\mathcal{A}$:

$$|\Pr[\mathcal{A}(\lambda, \mathsf{pk}, \mathsf{evk}, c_0) = 1] - \Pr[\mathcal{A}(\lambda, \mathsf{pk}, \mathsf{evk}, c_1) = 1]| \leq \mathsf{negl}(\lambda),$$

where the probability is over $(\mathsf{pk}, \mathsf{evk}, \mathsf{dk}) \xleftarrow{\$} \mathsf{KeyGen}(\lambda), c_b \xleftarrow{\$} \mathsf{FHE.Enc}_{\mathsf{pk}}(m_b), m_0, m_1 \xleftarrow{\$} \mathcal{M}$, and the coins of $\mathcal{A}$.

3.2 The Generic Scheme

Let $\mathsf{FHE} = (\mathsf{FHE.ParamGen}, \mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$ be an FHE scheme as defined above. Also let $\mathcal{VC} = (\mathbf{KeyGen}, \mathbf{ProbGen}, \mathbf{Compute}, \mathbf{Verify})$ be a VC scheme which is correct, secure (resp. adaptively secure), and outsourceable, as defined in Section 2. In particular note that $\mathcal{VC}$ does not need to be private, and that the security (resp. adaptive security) is guaranteed in the presence of verification queries.

We describe a new VC scheme $\mathcal{PVC} = (\mathbf{PrKeyGen}, \mathbf{PrProbGen}, \mathbf{PrCompute}, \mathbf{PrVerify})$ (for private VC) which uses the above two tools as follows.

$\mathbf{PrKeyGen}(f, \lambda) \to (PK_P, SK_P)$:
- Run $\mathsf{FHE.KeyGen}(\lambda)$ to generate $(\mathsf{pk}, \mathsf{dk}, \mathsf{evk})$ for FHE.
- Run $\mathbf{KeyGen}(eval_f, \lambda)$ to generate the PK, SK for $\mathcal{VC}$, where $eval_f$ is the function that takes as input $\mathsf{FHE.Enc}_{\mathsf{pk}}(x)$ and outputs $\mathsf{FHE.Enc}_{\mathsf{pk}}(f(x))$. Given $\mathsf{pk}, \mathsf{evk}$ this function is efficiently computable.
- Set $PK_P = (PK, \mathsf{pk}, \mathsf{evk})$ and $SK_P = (PK_P, SK, \mathsf{dk})$.

$\mathbf{PrProbGen}_{SK_P}(x) \to (\sigma_x, \tau_x)$:
- Compute $C_x = \mathsf{FHE.Enc}_{\mathsf{pk}}(x)$,
- Run $\mathbf{ProbGen}_{SK}(C_x)$ to get (σ_x, τ_x).

$\mathbf{PrCompute}_{PK_P}(\sigma_x) \to \sigma_y$: Run $\mathbf{Compute}_{PK}(\sigma_x)$ to compute σ_y. Note that σ_y is an encoding of $C_y = \mathsf{FHE.Eval}_{\mathsf{evk}}(f, C_x)$.

$\mathbf{PrVerify}_{SK_P}(\tau_x, \sigma_y) \to (acc, y)$: Run $\mathbf{Verify}_{SK}(\tau_x, \sigma_y)$ to get (acc, C). If $acc = 0$, reject. If $acc = 1$, decrypt $y = \mathsf{FHE.Dec}_{\mathsf{dk}}(C)$.

Theorem 1. *If* FHE *is a semantically secure FHE, and* $\mathcal{VC}$ *is a correct, secure (resp. adaptively secure), and outsourceable VC scheme, then* $\mathcal{PVC}$ *is a correct, secure (resp. adaptively secure), outsourceable, and* private *VC scheme.*

Proof. Correctness of $\mathcal{PVC}$ follows from the correctness of FHE and $\mathcal{VC}$. Similarly the fact that $\mathcal{VC}$ is secure (resp. adaptively secure), and outsourceable implies the same properties for $\mathcal{PVC}$.

The one thing to argue then is privacy for $\mathcal{PVC}$, and we prove that from the semantic security of FHE. In other words we show that if an adversary $\mathcal{A}$ can learn any information about the input x to **PrProbGen** in $\mathcal{PVC}$ then we can use $\mathcal{A}$ to break the semantic security of FHE.

Let us assume then that there exists $\mathcal{A}, f$ such that

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{Priv}[\mathcal{VC}, f, \lambda] = 1] \geq \zeta$$

where $\zeta = \zeta(\lambda)$ is non-negligible in λ. We build a simulator S which is allowed to query $\mathcal{A}$ as an oracle and such that

$$\left|\Pr\left[S^{\mathcal{A}}(\lambda, \mathsf{pk}, \mathsf{evk}, c_0) = 1\right] - \Pr\left[S^{\mathcal{A}}(\lambda, \mathsf{pk}, \mathsf{evk}, c_1) = 1\right]\right| \geq \zeta$$

where the probability is over $(\mathsf{pk}, \mathsf{evk}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\lambda), \{c_b \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}}(b)\}_{b\in\{0,1\}}$, and the coins of S.

On input $\mathsf{pk}, \mathsf{evk}, c_b$ the simulator S runs as follows:

1. Run $\mathsf{KeyGen}(eval_f, \lambda)$ to generate the PK, SK for $\mathcal{VC}$ on $eval_f$. It sets $PK_P = (PK, \mathsf{pk}, \mathsf{evk})$.
2. Run $\mathcal{A}$ on PK_P. Remember that in this step $\mathcal{A}$ is allowed two types of queries:
   - Queries to **PProbGen** which S can answer since it knows the public key pk of FHE and the secret key SK of $\mathcal{VC}$
   - Queries to **PVerify** which S can also answer since it knows the secret key SK of $\mathcal{VC}$. Remember that **PVerify** returns only the acceptance/rejection bit *acc* which S can calculate using only SK. The secret key dk of FHE (which S does not have) is not needed to answer these queries.
3. At some point $\mathcal{A}$ outputs two inputs $x_0 \neq x_1$. Let us assume for now that x_0 and x_1 differ in a single bit, for example the first. We will show later how to get rid of this assumption by a standard hybrid argument. So let's assume that x_0 starts with 0 and x_1 starts with 1.
   S will construct C_{x_b} by concatenating c_b with the encryptions of all the other bits (which S can compute using pk).
   S will finish to run **ProbGen** on x_b, to compute σ_{x_b}, τ_{x_b} and returns σ_{x_b} to $\mathcal{A}$. This part requires only knowledge of SK so S can simulate it.
4. $\mathcal{A}$ will continue running and making queries to **PProbGen** and **PVerify** which the simulator will be able to answer as above.
5. Finally $\mathcal{A}$ outputs a bit $\hat{b}$ which is equal to b with probability $1/2 + \zeta$. S outputs the same bit, and therefore S will also be correct with probability $1/2 + \zeta$.

To finish the proof we need to remove the assumption we made on the behavior of $\mathcal{A}$ in step 3. Let us assume that $x_0, x_1 \in \{0,1\}^n$ and set $x_b = [x_{b,1}x_{b,2}\dots x_{b,n}]$. Define the string $x^{(j)} = [x_{1,1}x_{1,2}\dots x_{1,j}x_{0,j+1}\dots x_{0,n}]$, so $x^{(0)} = x_0$ and $x^{(n)} = x_1$.

If $\mathcal{A}$ distinguishes between x_0 and x_1 with advantage ζ then by a standard hybrid argument there must exists a j such that $\mathcal{A}$ distinguishes between $x^{(j-1)}$ and $x^{(j)}$ with advantage at least ζ/n. Notice that in order to be the case we must have $x_{0,j} \neq x_{1,j}$. The proof then continues with S guessing the bit j and placing the challenge ciphertext c_b in position j of the ciphertext C_{x_b} sent to $\mathcal{A}$. S will still guess the correct bit with non-negligible advantage.

Security vs Adaptive Security. Notice that a particular instantiation of our generic scheme is split. For instance, consider the case in which the VC component of our generic solution is implemented by using a SNARK (e.g. any of the protocols in [13, 26, 47, 10]). In those protocols basically τ_x can be empty (as many of those protocols are publicly verifiable). As a consequence our generic protocol in this case is adaptively secure (the encoding of x is just an FHE encryption of x, and the secret key SK is the decryption key of the FHE scheme).

Hiding the Function. We point out that by outsourcing the universal circuit computing functions of a given size, we can hide not only the input data, but also the function being computed, so our scheme can be compiled into one which is also function private, according to definition 4, and without a significant loss in performance.

4 Tools

In this section we describe a collection of tools that we will use to design our efficient verifiable computation protocols for ad-hoc functions. The tools include: an adapted version of the BGV homomorphic encryption scheme, a new notion and realizations of homomorphic hash functions, and amortized closed-form efficient pseudorandom functions.

4.1 The BGV Homomorphic Encryption

In this section we describe a stripped down version of FHE scheme by Brakerski, Gentry, and Vaikuntanathan [18] that we need for our construction. The version below is only somewhat homomorphic, and resembles the less recent scheme by Brakerski and Vaikuntanathan [19]. Also, since we need to evaluate only polynomials of degree at most two, our description does not include the KeySwitch and ModulusSwitch operations.

$\mathsf{BGV.ParamGen}(\lambda)$**.** The message space $\mathcal{M}$ is the ring $R_p := \mathbb{F}_p[X]/\Phi_m(X)$, where $\Phi_m(X)$ is the mth cyclotomic polynomial in $\mathbb{F}_p[X]$, of degree $n = \Phi(m)$. Operations in $\mathcal{M}$ are denoted with $+$ for addition and $\cdot$ for polynomial multiplication modulo $\Phi(X)$. The homomorphic properties of the scheme are over R_p. We choose to represent elements in $\mathcal{M}$ as elements in $\mathbb{Z}^n$ with infinity norm bounded by $p/2$.

The ciphertext space is described as follows: pick a large q which is co-prime to p (the size of q can be determined as it is done in [19], section 2), and define $R_q := \mathbb{Z}/q\mathbb{Z}[X]/\Phi(X)$. Ciphertexts can be thought of as elements in $R_q[Y]$: level 1 ciphertexts (those created by the encryption procedure and eventually manipulated via additive operations only) are of degree 1, while level 0 ciphertexts (ciphertext manipulated via one multiplication and possibly other different homomorphic operations) are of degree 2. Operations on ciphertexts are the corresponding ring operations of $R_q[Y]$.

$$(a_0 + a_1 \cdot Y + a_2 \cdot Y^2) + (b_0 + b_1 \cdot Y + b_2 \cdot Y^2) = (a_0 + b_0 + (a_1 + b_1) \cdot Y + (a_2 + b_2) \cdot Y^2)$$
$$(a_0 + a_1 \cdot Y) \cdot (b_0 + b_1 \cdot Y) = (a_0 \cdot b_0 + (a_1 \cdot b_0 + b_1 \cdot a_0) \cdot Y + a_1 \cdot b_1 \cdot Y^2).$$

Notice we are interested only in multiplication of ciphertexts of level 1.

Finally, the algorithm defines the following probability distributions:

$D_{Z^n,\sigma}$**:** the discrete Gaussian with parameter σ: it is the random variable over $\mathbb{Z}^n$ obtained from sampling $x \in \mathbb{R}^n$ with probability $e^{-\pi \cdot \|x\|_2/\sigma^2}$ and then rounding at the nearest lattice point. Again, we refer to [19] for the specific choice of σ.

ZO_n**:** sample a vector $x = (x_1, \ldots, x_n)$ with $x_i \in \{-1, 0, +1\}$ and $\Pr[x_i = -1] = 1/4$; $\Pr[x_i = 1] = 1/4$; $\Pr[x_i = 0] = 1/2$.

In the following, we assume that the parameters generated here are inputs of any subsequent algorithm.

$\mathsf{BGV.KeyGen}() \to (\mathsf{pk}, \mathsf{dk})$**.** Sample $a \xleftarrow{\$} R_q$, and $s, e \xleftarrow{\$} D_{\mathbb{Z}^n,\sigma}$. Considering s and e as elements in R_q, compute $b \leftarrow a \cdot s + p \cdot e$, and set $\mathsf{dk} \leftarrow s$ and $\mathsf{pk} \leftarrow (a, b)$.

$\mathsf{BGV.Enc}_{\mathsf{pk}}(m, r) \to (c_0, c_1, c_2 = 0)$**.** Given $m \in R_p$, and $r \xleftarrow{\$} (ZO_n, D_{\mathbb{Z}^n,\sigma}, D_{\mathbb{Z}^n,\sigma})$, the message m is parsed as an element in R_q with infinity norm bounded by $p/2$, and the randomness r is parsed as $r = (u, v, w) \in R_q^3$. The output is $c = c_0 + c_1 \cdot Y \in R_q[Y]$, where $c_0 \leftarrow b \cdot v + p \cdot w + m$ and $c_1 \leftarrow a \cdot v + p \cdot u$.

$\mathsf{BGV.Dec}_{\mathsf{dk}}(c) \to t \bmod p$**.** Compute $t \in R_q$ as $t \leftarrow c_0 - s \cdot c_1 - s^2 \cdot c_2$. The output is then $t \bmod p$, which is interpreted as an element in R_p.

Lemma 1 ([19], Lemma 4; and [19], Theorem 2).
For $D = 1, 2$, the public-key encryption scheme $(\mathsf{BGV.ParamGen}, \mathsf{BGV.KeyGen}, \mathsf{BGV.Enc}, \mathsf{BGV.Dec})$*, specified above, is semantically secure under the PLWE assumption, and it allows the computation of any polynomial f of degree D such that*

$$\|f\|_\infty \cdot \left(p \cdot \sigma \cdot n^{1.5}\right)^D \leq q/2.$$

Batching. A nice capability of the BGV encryption scheme, which will be used extensively in our implementations, is the ability to encrypt many "small" plaintexts (e.g. integers modulo a 32-bit prime) into the same ciphertext: each small plaintext will then reside in its dedicated "slot" in the ciphertext. This feature was firstly introduced by Smart and Vercauteren [54], and it is achieved as follows (see [18, 54] for more details).

Two Lines about Number Theory. Let $F \in \mathbb{F}_p[X]$ be a monic polynomial of degree n such that

$$F(X) = \prod_{i=1}^{s} F_i(X),$$

where F_i is irreducible over $\mathbb{F}_p$ and $\deg(F_i) = n/s =: d$ for all i. Let $A_p := \mathbb{F}_p[X]/F$. Notice that:

$$A_p = \mathbb{F}_p[X]/F = \mathbb{F}_p[X]/(F_1, \ldots, F_s) \cong \mathbb{F}_p[X]/(F_1) \times \cdots \times \mathbb{F}_p[X]/(F_s) \cong \mathbb{F}_{p^d} \times \cdots \times \mathbb{F}_{p^d} = \left(\mathbb{F}_{p^d}\right)^s.$$

In detail, the ring operations over A_p are mapped to coordinate-wise ring operations over $(\mathbb{F}_{p^d})^s$.

Applicability to the Encryption Scheme. Let z be an integer, $d = 2^z$, $m = 2 \cdot d$, and p be a prime such that $p = 1 \bmod m$. With this choice of parameters, the native plaintext space for the BGV encryption scheme becomes $\mathcal{M} = R_p = \mathbb{F}_p[X]/\Phi_m(X) = \mathbb{F}_p[X]/(X^d + 1)$. Moreover, $X^d + 1$ is monic, and it splits over $\mathbb{F}_p$, so $R_p \cong (\mathbb{F}_p)^d$, by the above observation. This means that R_p can be thought of as an intermediary domain that homomorphically encodes d values in $\mathbb{F}_p$. This allows to encrypt and perform homomorphic operations on d "small" plaintexts over $\mathbb{F}_p$, by first encoding the d "small' plaintexts into R_p and then applying the regular BGV encryption. Visually:

$$(\mathbb{F}_p)^d \xrightarrow{\cong} R_p \xrightarrow{\mathsf{BGV.Enc}_{\mathsf{pk}}} R_q[Y] \xrightarrow{f} R_q[Y] \xrightarrow{\mathsf{BGV.Dec}_{\mathsf{sk}}} R_p \xrightarrow{\cong} (\mathbb{F}_p)^d$$

$$(m_1, \ldots, m_d) \longmapsto m \longmapsto \mathsf{BGV.Enc}_{\mathsf{pk}}(m) \longmapsto f(\mathsf{BGV.Enc}_{\mathsf{pk}}(m)) \longmapsto f(m) \longmapsto (f(m_1), \ldots, f(m_d)).$$

In the diagram above, m_i is said to reside in the ith plaintext *slot* of the cryptosystem, and d is the total number of admissible slots.

4.2 Homomorphic Hash Functions

In this section we introduce the notion of homomorphic hash functions and we propose two realizations that will be used throughout the paper. Informally, a family of keyed homomorphic hash functions H with domain $\mathcal{X}$ and range $\mathcal{Y}$ consists of a tuple of algorithms $(\mathsf{H.KeyGen}, \mathsf{H}, \mathsf{H.Eval})$ such that:

- $\mathsf{H.KeyGen}$ generates the description of a function H_K,
- H computes the function,
- $\mathsf{H.Eval}$ allows to compute over $\mathcal{Y}$.

In our case we are interested in computations of arithmetic circuits, and thus $\mathsf{H.Eval}$ allows to compute additions and multiplications over $\mathcal{Y}$.

In the following we propose a homomorphic hash whose key feature is that it allows to "compress" a BGV ciphertext $\mu \in R_q[Y]$ into a single entry $\nu \in \mathbb{Z}/q\mathbb{Z}$ in such a way that H is a ring homomorphism, hence $\mathsf{H.Eval}(f, (\mathsf{H}(\mu_1), \ldots, \mathsf{H}(\mu_t)) = \mathsf{H}(f(\mu_1, \ldots, \mu_t))$. Turning our attention to security, we show that this first construction is universal one-way. Next, we will show a variant of this construction that maps into bilinear groups and can be proven collision-resistant.

A Universal One-Way Homomorphic Hash. Let q be a prime and $R_q[Y] = \mathbb{Z}/q\mathbb{Z}[X,Y]/\Phi_m(X)$ be as in BGV. The family of hash functions $(\mathsf{H.KeyGen}, \mathsf{H}, \mathsf{H.Eval})$ with domain $R_q[Y]$ and range $\mathbb{Z}/q\mathbb{Z}$ is defined as follows:

$\mathsf{H.KeyGen}$**:** Pick $\alpha \xleftarrow{\$} R_q$ and $\beta \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$, and set $\kappa = (\alpha, \beta)$.

$\mathsf{H}_\kappa(\mu)$**:** On input $\mu \in R_q[Y]$, the function $\mathsf{H}_{\alpha,\beta}$ evaluates μ at $Y = \alpha$ and then evaluates $\mu(\alpha) \in R_q = \mathbb{Z}/q\mathbb{Z}[X]/\Phi(X)$ at $\beta \in \mathbb{Z}/q\mathbb{Z}$. More explicitly, $\mathsf{H}_{\alpha,\beta} = ev_\beta \circ ev_\alpha$:

$$\begin{array}{ccccc} & & \overset{\mathsf{H}_{\alpha,\beta}}{\frown} & & \\ R_q[Y] = \mathbb{Z}/q\mathbb{Z}[X,Y]/\Phi_m(X) & \xrightarrow{ev_\alpha} & \mathbb{Z}/q\mathbb{Z}[X]/\Phi_m(X) & \xrightarrow{ev_\beta} & \mathbb{Z}/q\mathbb{Z} \\ \mu = \sum_{j=0}^{2} \mu_j Y^j & \longmapsto & \sum_{j=0}^{2} \mu_j \alpha^j = \sum_{j=0}^{2} \sum_{i=0}^{n-1} \left(\mu_j \alpha^j\right)_i X^i & \longmapsto & \sum_{j=0}^{2} \sum_{i=0}^{n-1} \left(\mu_j \alpha^j\right)_i \beta^i. \end{array}$$

$\mathsf{H.Eval}(f_g, \nu_1, \nu_2)$**:** on input two values $\nu_1, \nu_2 \in \mathbb{Z}/q\mathbb{Z}$ and an operation f_g which is addition $+$ or multiplication $\times$, compute $f_g(\nu_1, \nu_2)$.

Theorem 2. *The family of functions* H *defined above is homomorphic and universal one-way, i.e., for all* $\mu \neq \mu'$*:*

$$\Pr[\mathsf{H}_{\alpha,\beta}(\mu) = \mathsf{H}_{\alpha,\beta}(\mu') : \alpha \xleftarrow{\$} R_q, \beta \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}] \leq 2/q^{2n} + n/q$$

which is negligible for an appropriate choice of q*.*

Proof. To see the homomorphic property note that since ev is a ring homomorphism and H is a composition of ev_α and ev_β, then H is a ring homomorphism.

For security, note that from the homomorphic property of H, $\mathsf{H}_{\alpha,\beta}(\mu) = \mathsf{H}_{\alpha,\beta}(\mu')$ implies $\mathsf{H}_{\alpha,\beta}(\mu - \mu') = 0$. Also, since $\mu \neq \mu'$, $\mu - \mu'$ is a non trivial element in the kernel of $\mathsf{H}_{\alpha,\beta}$. Namely, we obtain a non-zero polynomial $\varphi(X,Y)$ that evaluates to zero for $Y = \alpha$ and $X = \beta$. Over the random choice of $\alpha \in R_q$ and $\beta \in \mathbb{Z}/q\mathbb{Z}$, the probability of this event, by a standard argument in algebra, is bounded by $\deg_Y(\varphi)/|R_q| + \deg_X(\varphi)/|\mathbb{Z}/q\mathbb{Z}| \leq 2/q^{2n} + n/q$, which is negligible. □

A Collision-Resistant Homomorphic Hash. Notice that the function $\mathsf{H}_{\alpha,\beta}$ is secure only if the key α, β is kept secret and the function is used only one time (otherwise information on α and β is leaked). Below, we show how to obtain a slightly different version of $\mathsf{H}_{\alpha,\beta}$ which can be proven collision-resistant at the price of being "somewhat" homomorphic – the homomorphic property holds only for degree-2 functions. Moreover, in this construction we restrict to the case in which q is a prime.

Let $\mathsf{bgpp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ be some bilinear group parameters as described before, and let $R_q[Y]$ be as in BGV.

The family of hash functions $(\hat{\mathsf{H}}.\mathsf{KeyGen}, \hat{\mathsf{H}}, \hat{\mathsf{H}}.\mathsf{Eval})$ with domain $R_q[Y]$ and range $\mathbb{G}_1 \times \mathbb{G}_2$ (or $\mathbb{G}_T$) is as follows:

$\hat{\mathsf{H}}.\mathsf{KeyGen}$**:** To sample a member of the family, choose random $\alpha \xleftarrow{\$} R_q$ and $\beta \xleftarrow{\$} \mathbb{F}_q$. Parse α as $(\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_q^n$. Next, for $i, j = 1, \ldots, n$ $k = 0, \ldots, n-1$, compute g^{α_i}, h^{α_i}, $g^{\alpha_i \alpha_j}$, $h^{\alpha_i \alpha_j}$, $g^{\alpha_i \alpha_j \beta^k}$, $h^{\alpha_i \alpha_j \beta^k}$, and include them in K.

Output K and $\kappa = (\alpha, \beta)$.

$\hat{\mathsf{H}}(\mu)$**.** On input $\mu \in R_q[Y]$ such that $\deg_Y(\mu) \leq 2$, the function $\hat{\mathsf{H}}(\mu)$ works as follows.

Let $\mathsf{H}_{\alpha,\beta}$ be the function defined above. If $\deg_Y(\mu) \leq 1$, then compute $(T, U) \leftarrow (g^{\mathsf{H}_{\alpha,\beta}(\mu)}, h^{\mathsf{H}_{\alpha,\beta}(\mu)}) \in \mathbb{G}_1 \times \mathbb{G}_2$. If $\deg_Y(\mu) = 2$, compute $e(g,h)^{\mathsf{H}_{\alpha,\beta}(\mu)} \in \mathbb{G}_T$.

Note that $\hat{\mathsf{H}}$ can be computed in two different ways (the first one being computationally more efficient): (1) by using the secret key κ, or (2) by using the values in the public key K.

$\hat{\mathsf{H}}.\mathsf{Eval}(f_g, \nu_1, \nu_2)$ We show how to compute degree-2 functions on the outputs of $\hat{\mathsf{H}}$ in a homomorphic way. Intuitively speaking, we want to compute degree-2 polynomials f over $\mathbb{F}_q$ "in the exponent". To this end we rely on that the bilinear groups are isomorphic to $\mathbb{F}_q$ and simulate additions via the group operation and multiplications by using the bilinear pairing.

More precisely, given (T_1, U_1), (T_2, U_2) (resp. $\hat{T}_1, \hat{T}_2 \in \mathbb{G}_T$):

- Addition (in the exponent) is performed via (component-wise) group operation, i.e., $(T \leftarrow T_1 \cdot T_2, U \leftarrow U_1 \cdot U_2)$ (resp. $\hat{T} \leftarrow \hat{T}_1 \cdot \hat{T}_2$).
- Multiplication by a constant $c \in \mathbb{F}_q$ is performed as (T^c, U^c) (resp. $\hat{T}^c$).
- Multiplication of two values, is performed with the use of the bilinear pairing: $\hat{T} \leftarrow e(T_1, U_2) \in \mathbb{G}_T$.

In the following theorem we show that the function $\hat{\mathsf{H}}$ described above is homomorphic, and it is collision-resistant under the ℓ-BDHI assumption which we recall below.

Definition 8 (ℓ-BDHI Assumption [14]). *Let $\mathcal{G}$ be a bilinear group generator, and let $\mathsf{bgpp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h) \xleftarrow{\$} \mathcal{G}(1^\lambda)$. Let $z \xleftarrow{\$} \mathbb{F}_q$ be chosen uniformly at random. We say that the ℓ-BDHI assumption holds for $\mathcal{G}$ if for every PPT adversary $\mathcal{A}$ and any $\ell = \mathsf{poly}(\lambda)$ the probability*

$$\Pr[\mathcal{A}(\mathsf{bgpp}, g, h, g^z, h^z, g^{z^2}, h^{z^2}, \ldots, g^{z^\ell}, h^{z^\ell}) = e(g,h)^{1/z}] = \mathsf{negl}(\lambda).$$

Theorem 3. *The function $\hat{\mathsf{H}}$ described above is homomorphic. Furthermore, if the $(n+1)$-BDHI assumption holds for $\mathcal{G}$, then $\hat{\mathsf{H}}$ is collision-resistant, i.e., for $(K, \kappa) \xleftarrow{\$} \hat{\mathsf{H}}.\mathsf{KeyGen}$*

$$\Pr[\hat{\mathsf{H}}(\mu) = \hat{\mathsf{H}}(\mu') \wedge \mu \neq \mu' \mid (\mu, \mu') \leftarrow \mathcal{A}(K)] = \mathsf{negl}(\lambda).$$

Proof. The homomorphic property easily follows by observing that $\hat{\mathsf{H}}.\mathsf{Eval}$ computes the function $\mathsf{H}_{\alpha,\beta}$ in the exponent.

For security, assume for the sake of contradiction that there exists a PPT adversary $\mathcal{A}$ such that $\Pr[\hat{\mathsf{H}}(\mu) = \hat{\mathsf{H}}(\mu') : \mu \neq \mu', (\mu, \mu') \leftarrow \mathcal{A}(K), (K, \kappa) \xleftarrow{\$} \hat{\mathsf{H}}.\mathsf{KeyGen}]$ is non-negligible. We show

how to build a PPT algorithm $\mathcal{B}$ that uses such $\mathcal{A}$ to break the $(n+1)$-BDHI assumption with non-negligible probability.

$\mathcal{B}$ takes as input the tuple $\{(g^{z^i}, h^{z^i})\}_{i=1}^{n+1}$, and its goal is to compute $e(g,h)^{1/z} \in \mathbb{G}_T$.

First, $\mathcal{B}$ chooses random $s, \gamma_1, r_1, \ldots, \gamma_n, r_n \xleftarrow{\$} \mathbb{F}_q$ and simulates K by computing the following values (for $i, j = 1, \ldots, n$, $k = 0, \ldots, n-1$):

- $g^{\alpha_i} = (g^z)^{\gamma_i} g^{r_i}$,
- $g^{\alpha_i \alpha_j} = (g^{z^2})^{\gamma_i \gamma_j} (g^z)^{\gamma_i r_j + \gamma_j r_i} g^{r_i r_j}$,
- $g^{\alpha_i \alpha_j \beta^k} = [(g^{z^{k+2}})^{\gamma_i \gamma_j} (g^{z^{k+1}})^{\gamma_i r_j + \gamma_j r_i} (g^{z^k})^{r_i r_j}]^{s^k}$,
- $g^{\beta^k} = (g^{z^k})^{s^k}$.

Respectively, $\mathcal{B}$ computes the values $h^{\alpha_i}, h^{\alpha_i \alpha_j}$ and $h^{\alpha_i \alpha_j \beta^k}$ from the given $\{h^{z^i}\}_i$. This way $\mathcal{B}$ implicitly sets $\kappa = (\alpha, \beta)$, with

$$\alpha_i = z\gamma_i + r_i \text{ and } \beta = sz. \tag{1}$$

Note that from the adversary's view K, the vector γ, is information-theoretically hidden, i.e., γ is uniformly distributed over $\mathbb{F}_q^n$.

Next, $\mathcal{B}$ runs $\mathcal{A}$ on input K and with non-negligible probability obtains $\mu, \mu' \in R_q[Y]$ such that $\mu \neq \mu'$ and $\hat{\mathsf{H}}(\mu) = \hat{\mathsf{H}}(\mu')$. Without loss of generality, assume that $\deg_Y(\mu) = \deg_Y(\mu') = 2$. By letting $\delta = \mu - \mu' \in R_q[Y]$ the above collision means that $\hat{\mathsf{H}}(\delta) = e(g,h)^{\mathsf{H}_{\alpha,\beta}(\delta)} = 1 \in \mathbb{G}_T$, i.e., $\mathsf{H}_{\alpha,\beta}(\delta) = 0 \mod q$. In other words, the (non-zero) polynomial δ evaluates to zero at $Y = \alpha$, $X = \beta$. Consider the functions $\alpha(Z) = Z \cdot \gamma + r$, and $\beta(Z) = s \cdot Z$. By equation 1, we get $\alpha(z) = \alpha$ and $\beta(z) = \beta$, therefore the following equalities hold:

$$0 = \mathsf{H}_{\alpha,\beta}(\delta) = \delta(X,Y)_{|Y=\alpha, X=\beta} = \delta(X,Y)_{|Y=\alpha(Z), X=\beta(Z), Z=z} = \delta(\alpha(Z), \beta(Z))_{|Z=z}. \tag{2}$$

Define $\omega(Z) = \delta(\alpha(Z), \beta(Z))$. First, notice that ω is a non-zero polynomial (over $\mathbb{F}_q$) with overwhelming probability, since δ is non-zero and γ, r, s are uniform, and second, notice that the coefficients of ω are efficiently computable from γ, r, s, and δ.

Write $\omega(Z) = Z^j(\omega_j + Z \cdot \omega'(Z))$, for some j, where $\omega_j \neq 0$, $\omega'(Z) = \sum_{i=j}^{\deg_Z(\omega)} \omega_i \cdot Z^{i-j}$, and $\deg_Z(\omega) \leq n+1$. Now, since $\omega(z) = 0$ (by construction and by equation 2), and since $z \neq 0$, we have $\omega_j = -z \cdot \omega'(z)$. Therefore, $\mathcal{B}$ can compute

$$e\left(\prod_{i=j}^{\deg_Z(\omega)} (g^{z^{i-j}})^{\omega_i}, h\right)^{-1/\omega_j} = e(g,h)^{-\omega'(z)/\omega_j} = e(g,h)^{1/z},$$

which concludes the proof. □

4.3 Amortized closed-form Efficient Pseudorandom Functions

Here we recall the notion of pseudorandom functions with amortized closed-form efficiency [6] which extend closed-form-efficient PRFs [11].

A PRF consists of two algorithms $(\mathsf{F.KG}, \mathsf{F})$ such that the key generation $\mathsf{F.KG}$ 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 $\mathcal{R}$ of the function, and the function $\mathsf{F}_K(x)$ takes input $x \in \mathcal{X}$ and uses the secret key K to compute a value $R \in \mathcal{R}$. As usual, a PRF must satisfy the pseudorandomness property. Namely, we say that $(\mathsf{F.KG}, \mathsf{F})$ is *secure* if for every PPT adversary $\mathcal{A}$ we have that:

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

where $\epsilon(\lambda)$ is negligible, $(K, \mathsf{pp}) \xleftarrow{\$} \mathsf{F.KG}(1^\lambda)$, and $\Phi : \mathcal{X} \to \mathcal{R}$ is a random function.

For any PRF $(\mathsf{F.KG}, \mathsf{F})$ *amortized closed-form efficiency* is defined as follows.

Definition 9 (Amortized Closed-Form Efficiency [6]). *Consider a computation* Comp *that takes as input* n *random values* $R_1, \ldots, R_n \in \mathcal{R}$ *and a vector of* m *arbitrary values* $\boldsymbol{z} = (z_1, \ldots, z_m)$*, and assume that the computation of* $\mathsf{Comp}(R_1, \ldots, R_n, z_1, \ldots, z_m)$ *requires time* $t(n, m)$*.*

Let $\mathbf{L} = (\mathsf{L}_1, \ldots, \mathsf{L}_n)$ *be arbitrary values in the domain* $\mathcal{X}$ *of* F *such that each can be interpreted as* $\mathsf{L}_i = (\Delta, \tau_i)$*. We say that a PRF* $(\mathsf{F.KG}, \mathsf{F})$ *satisfies* amortized closed-form efficiency *for* $(\mathsf{Comp}, \mathbf{L})$ *if there exist algorithms* $\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{Comp},\boldsymbol{\tau}}$ *and* $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{Comp},\Delta}$ *such that:*

1. *Given* $\omega \leftarrow \mathsf{CFEval}^{\mathsf{off}}_{\mathsf{Comp},\boldsymbol{\tau}}(K, \boldsymbol{z})$*, we have that*

$$\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{Comp},\Delta}(K, \omega) = \mathsf{Comp}(\mathsf{F}_K(\Delta, \tau_1), \ldots, \mathsf{F}_K(\Delta, \tau_n), z_1, \ldots, z_m)$$

2. *the running time of* $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{Comp},\Delta}(K, \omega)$ *is* $o(t)$*.*

A Realization Based on Decision Linear in Bilinear Groups. Below we show a realization of amortized closed-form efficient PRFs based on the decision linear assumption. The scheme is obtained by adapting the one of [6] to work with asymmetric bilinear groups. This function will be crucial to achieve efficiency for our schemes for quadratic multi-variate polynomials.

Let $f : \mathbb{F}_q^t \to \mathbb{F}_q$ be a degree-2 arithmetic circuit, and without loss of generality, parse

$$f(Z_1, \ldots, Z_t) = \sum_{i,j=1}^{t} \eta_{i,j} \cdot Z_i \cdot Z_j + \sum_{k=1}^{t} \eta_k \cdot Z_k, \tag{3}$$

for some $\eta_{i,j}, \eta_k \in \mathbb{F}_q$. We define $\hat{f} : (\mathbb{G}_1 \times \mathbb{G}_2)^t \to \mathbb{G}_T$ as the compilation of f on group elements as:

$$\hat{f}(A_1, B_1, \ldots, A_t, B_t) = \prod_{i,j=1}^{t} e(A_i, B_j)^{\eta_{i,j}} \cdot \prod_{k=1}^{t} e(A_k, h)^{\eta_k}. \tag{4}$$

Below, we describe the PRF with amortized closed-form efficiency for $\mathsf{Comp}(R_1, S_1, \ldots, R_t, S_t, f) = \hat{f}(R_1, S_1, \ldots, R_t, S_t)$:

$\mathsf{F.KG}(1^\lambda)$**.** Let bgpp be some bilinear group parameters, where $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$ are generators.

- Choose two seeds K_1, K_2 for a family of PRFs $\mathsf{F}'_{K_{1,2}} : \{0,1\}^* \to \mathbb{F}_q^2$.
- Output $K = (K_1, K_2)$. The parameters define a function F with domain $\mathcal{X} = \{0,1\}^* \times \{0,1\}^*$ and range $\mathcal{R} = \mathbb{G}_1 \times \mathbb{G}_2$, as described below.

$\mathsf{F}_K(\Delta, \tau)$**.**

- Generate values $(u, v) \leftarrow \mathsf{F}'_{K_1}(\tau)$ and $(a, b) \leftarrow \mathsf{F}'_{K_2}(\Delta)$.
- Output $R = g^{ua+vb}, S = h^{ua+vb}$.

$\mathsf{CFEval}^{\mathsf{off}}_{\boldsymbol{\tau}}(K, f)$**.** Parse $K = (K_1, K_2)$ as a secret key for the PRF, and f as a function $\mathbb{F}_q^t \to \mathbb{F}_q$.

- For $i = 1$ to t, compute $(u_i, v_i) \leftarrow \mathsf{F}'_{K_1}(\tau_i)$, and interpret (u_i, v_i) as a linear form ρ_i that maps (z_1, z_2) to $\rho_i(z_i, z_2) = u_i \cdot z_1 + v_i \cdot z_2$.
- Next, run $\rho \leftarrow f(\rho_1, \ldots, \rho_t)$, i.e. compute a (possibly quadratic) form ρ such that for all $z_1, z_2 \in \mathbb{F}_q$

$$\rho(z_1, z_2) = f(\rho_1(z_1, z_2), \ldots, \rho_t(z_1, z_2)).$$

- Finally, output $\omega_f = \rho$.

$\mathsf{CFEval}^{\mathsf{on}}_{\Delta}(K, \omega_f)$**.** Parse $K = (K_1, K_2)$ as a secret key and $\omega_f = \rho$ as in the previous algorithm. The online evaluation algorithm does the following:

- Generate $(a, b) \leftarrow \mathsf{F}'_{K_2}(\Delta)$.
- Compute $w = \rho(a, b)$.
- Output $W = e(g, h)^w$.

The function above is secure under the decision linear assumption in asymmetric bilinear groups, that we recall below.

Definition 10 (Decision Linear [15]). *Let $\mathcal{G}$ be a bilinear group generator, and let* $\mathsf{bgpp} \xleftarrow{\$} \mathcal{G}(1^\lambda)$*. Let $r_0, r_1, r_2, x_1, x_2 \xleftarrow{\$} \mathbb{F}_q$ be chosen uniformly at random.*
Let $T = (g, h, g^{x_1}, g^{x_2}, g^{x_1 r_1}, g^{x_2 r_2}, h^{x_1}, h^{x_2}, h^{x_1 r_1}, h^{x_2 r_2})$.
We define the advantage of an adversary $\mathcal{A}$ in solving the decision linear *problem as*

$$\mathbf{Adv}^{dlin}_{\mathcal{A}}(\lambda) = |\Pr[\mathcal{A}(\mathsf{bgpp}, T, g^{r_1+r_2}, h^{r_1+r_2}) = 1] - \Pr[\mathcal{A}(\mathsf{bgpp}, T, g^{r_0}, h^{r_0}) = 1]|$$

We say that the decision linear assumption holds for $\mathcal{G}$ if for every PPT algorithm $\mathcal{A}$, $\mathbf{Adv}^{dlin}_{\mathcal{A}}(\lambda)$ is negligible.

Theorem 4. *If the Decision Linear assumption holds for $\mathcal{G}$, and F' is a family of pseudorandom functions, then the function F described above is a pseudorandom function with amortized closed-form efficiency for $\mathsf{Comp} = \hat{f}$ as defined above.*

The proof of theorem 4 follows the one of [6], theorem 2.

A Realization Based on DDH over 2^k-Residues in $\mathbb{Z}^*_N$. here we propose another amortized closed-form efficient PRF based on the DDH assumption in the subgroup of 2^k-residues of $\mathbb{Z}^*_N$.

Let $f : \mathbb{Z}^t_{2^k} \to \mathbb{Z}_{2^k}$ be a linear function $f(Z_1, \ldots, Z_t) = \sum_{i=1}^t f_i \cdot Z_i$. We define $\hat{f} : (\mathbb{Z}^*_N)^t \to \mathbb{Z}^*_N$ as the compilation of f on $\mathbb{Z}^*_N$ elements as:

$$\hat{f}(A_1, \ldots, A_t) = \prod_{i=1}^{t} A_i^{f_i} \mod N \tag{5}$$

Below we describe the PRF with amortized closed-form efficiency for $\mathsf{Comp}(R_1, \ldots, R_t, f) = \hat{f}(R_1, \ldots, R_t)$:

$\mathsf{F.KG}(1^\lambda)$**.** Let $N = pq$ be the product of two quasi-safe primes $p = 2^k p' + 1$ and $q = 2^k q' + 1$. Let $\mathcal{R}_k$ be the following subgroup of $\mathbb{Z}^*_N$, $\mathcal{R}_k = \{x^{2^k} : x \in \mathbb{Z}^*_N\}$, and let $g \in \mathcal{R}_k$ be a generator.

- Choose two seeds K_1, K_2 for a family of PRFs $\mathsf{F}'_{K_{1,2}} : \{0,1\}^* \to \mathbb{Z}_{p'q'}$.
- Output $K = (K_1, K_2)$ and $\mathsf{pp} = (N, k)$. These parameters define a function F with domain $\mathcal{X} = \{0,1\}^* \times \{0,1\}^*$ and range $\mathbb{Z}^*_N$, as described below.

$\mathsf{F}_K(\Delta, \tau)$**.**

- Generate values $v \leftarrow \mathsf{F}'_{K_1}(\tau)$ and $b \leftarrow \mathsf{F}'_{K_2}(\Delta)$.
- Output $R = g^{vb} \mod N$.

$\mathsf{CFEval}^{\mathsf{off}}_{\tau}(K, f)$**.** Parse $K = (K_1, K_2)$ as a secret key for the PRF, and f as a function $(\mathbb{Z}_{2^k})^t \to \mathbb{Z}_{2^k}$.

- For $i = 1$ to t, compute $v_i \leftarrow \mathsf{F}'_{K_1}(\tau_i)$.
- Next, compute $\rho \leftarrow f(v_1, \ldots, v_t) \mod p'q'$.
- Finally, output $\omega_f = \rho$.

$\mathsf{CFEval}^{\mathsf{on}}_{\Delta}(K, \omega_f)$**.** Parse $K = (K_1, K_2)$ as a secret key and $\omega_f = \rho$ as in the previous algorithm. The online evaluation algorithm does the following:

– Generate $b \leftarrow \mathsf{F}'_{K_2}(\Delta)$.
– Output $W = g^{\rho b} \mod N$.

The function above can be proven secure from the DDH assumption in $\mathcal{R}_k$, that we recall below.

Definition 11 (DDH). *Let $N = pq$ be the product of two quasi-safe primes $p = 2^k p' + 1$ and $q = 2^k q' + 1$. Define $\mathcal{R}_k = \{x^{2^k} : x \in \mathbb{Z}_N^*\}$, and let $g \in \mathcal{R}_k$ be a generator. Let $a, b, c \xleftarrow{\$} \mathbb{Z}_{p'q'}$ be chosen uniformly at random. We define the advantage of an adversary $\mathcal{A}$ in solving the DDH problem as*

$$\mathbf{Adv}^{ddh}_{\mathcal{A}}(\lambda) = |\Pr[\mathcal{A}(N, k, g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(N, k, g, g^a, g^b, g^c) = 1]|$$

We say that the DDH assumption holds in $\mathcal{R}_k$ if for every PPT algorithm $\mathcal{A}$ we have that $\mathbf{Adv}^{ddh}_{\mathcal{A}}(\lambda)$ is negligible.

Theorem 5. *If the DDH assumption holds for $\mathcal{R}_k$, and F' is a family of pseudorandom functions, then the function F described above is a pseudorandom function with amortized closed-form efficiency for $\mathsf{Comp} = \hat{f}$ as defined above.*

Proof. The pseudo randomness property is immediate from the definition of the function and by the random self reducibility of DDH.

To see the amortized closed-form efficiency, note that

$$W = g^{\rho b} = g^{b\sum_{i=1}^{t} f_i v_i} = g^{\sum_{i=1}^{t} f_i (b v_i)} = \prod_{i=1}^{t} (g^{b v_i})^{f_i}$$

$$= \prod_{i=1}^{t} \mathsf{F}_K(\Delta, i)^{f_i} \mod N = \hat{f}(\mathsf{F}_K(\Delta, 1), \ldots, \mathsf{F}_K(\Delta, t))$$

□

5 Computing Multi-Variate Polynomials of Degree 2

In this section we propose an efficient VC scheme for the case of multi-variate polynomials of degree 2. The basic idea of the construction is to apply the homomorphic MAC scheme of [6] to the BGV homomorphic encryption, where (in this section) q is choosen to be prime. Such homomorphic MAC indeed allows to authenticate degree-2 arithmetic computations over $\mathbb{F}_q$ and achieves amortized efficient verification, i.e., after a pre-computation phase whose cost is the same as running f, every output of f can be verified in constant time. However, a straightforward application of the scheme [6] on top of BGV ciphertexts would require to: (1) authenticate each of the $2n$ $\mathbb{F}_q$-components of a BGV ciphertext, and (2) authenticate the BGV evaluation circuit $\hat{f} : \mathbb{F}_q^{2nt} \to \mathbb{F}_q^{3n}$ instead of $f : \mathbb{F}_q^t \to \mathbb{F}_q$. Essentially, this would incur into a $O(n)$ blowup in all algorithms.

In contrast, we significantly improve over this approach in two main ways. First, we apply our collision-resistant homomorphic hash function $\hat{\mathsf{H}}$ which allows to compress a BGV ciphertext into a pair of group elements (T, U), yet it is a ring homomorphism for $R_q[Y]$. Second, we modify the homomorphic MAC scheme of [6] so that it can authenticate group elements (instead of $\mathbb{F}_q$ values). Combining these two ideas allows us to avoid the $O(n)$ blow-up (see section 10 for a concrete comparison).

Our scheme $\mathcal{VC}_{\mathsf{quad}}$ is specified as follows:

KeyGen$(f, \lambda) \to (PK, SK)$:

- Run $\mathsf{BGV.ParamGen}(\lambda)$ to get the description of the parameters for the BGV encryption scheme; run $(\mathsf{pk}, \mathsf{dk}) \xleftarrow{\$} \mathsf{BGV.KeyGen}()$. Let $R_q := \mathbb{F}_q[X]/\Phi_m(X)$ be the polynomial ring where $\Phi_m(X)$ is the mth cyclotomic polynomial in $\mathbb{F}_q[X]$ of degree $n = \Phi(m)$. The message space $\mathcal{M}$ is the ring $R_q[Y]$.
- Run $\mathsf{bgpp} \xleftarrow{\$} \mathcal{G}(1^\lambda)$ to generate the description of asymmetric bilinear groups $\mathsf{bgpp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$, where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of the same prime order q, $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$ are two generators, and $e : \mathbb{G}_1 \times \mathbb{G}_2$ is a non-degenerate bilinear map.
- Choose a random member of the hash function family $\hat{\mathsf{H}} : R_q[Y] \to \mathbb{G}_1 \times \mathbb{G}_2$ by running $(\kappa, \hat{K}) \xleftarrow{\$} \hat{\mathsf{H}}.\mathsf{KeyGen}$. For the convenience of our scheme we do not use the public key of $\hat{\mathsf{H}}$.
- Choose a random value $a \xleftarrow{\$} \mathbb{F}_q$, and run $(K, \mathsf{pp}) \xleftarrow{\$} \mathsf{PRF.KeyGen}(1^\lambda)$ to obtain the seed K of a function $\mathsf{F}_K : \{0,1\}^* \times \{0,1\}^* \to \mathbb{G}_1 \times \mathbb{G}_2$. In particular, F_K is supposed to be computationally indistinguishable from a function that outputs $(R, S) \in \mathbb{G}_1 \times \mathbb{G}_2$ such that $DLog_g(R) = DLog_h(S)$ is uniform over $\mathbb{F}_q$ (i.e., $e(R, h) = e(g, S)$).
- Compute a concise verification information for f by using the offline closed-form efficient algorithm of F, i.e., $\omega_f \leftarrow \mathsf{CFEval}^{\mathsf{off}}_{\boldsymbol{\tau}}(K, f)$.
- Output the secret key $SK = (\mathsf{pk}, \mathsf{dk}, \kappa, a, K, \omega_f)$, and the public evaluation key $PK = (\mathsf{pk}, \mathsf{pp}, f)$.

ProbGen$_{SK}(\boldsymbol{x} = (x_1, \ldots, x_t)) \to \sigma_x, \tau_x$:

- Choose an arbitrary string $\Delta \in \{0,1\}^\lambda$ as an identifier for the input vector $\boldsymbol{x}$.
- For $i = 1$ to t: first run $\mu_i \xleftarrow{\$} \mathsf{BGV.Enc}_{\mathsf{pk}}(x_i)$ to obtain a BGV ciphertext $\mu_i \in R_q^2$, and compute its hash value $(T_i, U_i) \leftarrow \hat{\mathsf{H}}_\kappa(\mu_i)$. Next, run $(R_i, S_i) \leftarrow \mathsf{F}_K(\Delta, i)$, compute

$$X_i \leftarrow (R_i \cdot T_i^{-1})^{1/a} \in \mathbb{G}_1, \; Y_i \leftarrow (S_i \cdot U_i^{-1})^{1/a} \in \mathbb{G}_2,$$

  and set $\sigma_i = (T_i, U_i, X_i, Y_i, \Lambda_i = 1) \in (\mathbb{G}_1 \times \mathbb{G}_2)^2 \times \mathbb{G}_T$.
- Output $\sigma_x = (\Delta, \mu_1, \sigma_1, \ldots, \mu_t, \sigma_t)$, and $\tau_x = \perp$.

Note that **ProbGen** can work in the streaming model in which every input item x_i can be processed separately.

Compute$_{PK}(\sigma_x) \to \sigma_y$:

- Let f be an admissible circuit and $\sigma_x = (\Delta, \mu_1, \sigma_1, \ldots, \mu_t, \sigma_t)$.
- First, run $\mu \leftarrow \mathsf{BGV.Eval}_{\mathsf{pk}}(f, \mu_1, \ldots, \mu_t)$ to homomorphically evaluate f over the ciphertexts (μ_i).
- Next, homomorphically evaluate f over the authentication tags $(\sigma_1, \ldots, \sigma_t)$). To do so, proceed gate-by-gate over f as described below.

  $\mathsf{GateEval}(f_g, \sigma^{(1)}, \sigma^{(2)}) \to \sigma$. Parse $\sigma^{(i)} = (T^{(i)}, U^{(i)}, X^{(i)}, Y^{(i)}, \Lambda^{(i)}) \in (\mathbb{G}_1 \times \mathbb{G}_2)^2 \times \mathbb{G}_T$ for $i = 1, 2$, where f_g equals $+$ or $\times$. Compute $\sigma = (T, U, X, Y, \Lambda)$ as:

  Addition. If $f_g = +$:

$$T \leftarrow T^{(1)} \cdot T^{(2)}, \; U \leftarrow U^{(1)} \cdot U^{(2)}, \; X \leftarrow X^{(1)} \cdot X^{(2)}, \; Y \leftarrow Y^{(1)} \cdot Y^{(2)}, \; \Lambda \leftarrow \Lambda^{(1)} \cdot \Lambda^{(2)}.$$

  Multiplication by constant. If $f_g = \times$ and one of the two inputs, say $\sigma^{(2)}$, is a constant $c \in \mathbb{F}_q$:

$$T \leftarrow (T^{(1)})^c, \quad U \leftarrow (U^{(1)})^c, \quad X \leftarrow (X^{(1)})^c, \quad Y \leftarrow (Y^{(1)})^c, \quad \Lambda \leftarrow (\Lambda^{(1)})^c.$$

Multiplication. If $f_g = \times$: if $\deg(\mu^{(1)}) > 1$ or $\deg(\mu^{(2)}) > 1$, reject; else:

$$T \leftarrow e(T^{(1)}, U^{(2)}), \quad U \leftarrow e(T^{(2)}, U^{(1)}), \quad \Lambda \leftarrow e(X^{(1)}, Y^{(2)}),$$
$$X \leftarrow e(X^{(1)}, U^{(2)})e(X^{(2)}, U^{(2)}), \quad Y \leftarrow e(T^{(1)}, Y^{(2)})e(T^{(2)}, Y^{(2)}).$$

Note that after a multiplication, it is not necessary to keep U and Y, as since (for honestly computed tags) $T = U$ and $X = Y$. We keep them only for ease of description.

- Let σ be the authentication tag obtained after evaluating the last gate of f.
  Output $\sigma_y = (\Delta, \mu, \sigma)$.

Verify$_{SK}(\sigma_y = (\Delta, \mu, \sigma), \tau_x) \to (acc, x')$:

- Parse $SK = (\mathsf{pk}, \mathsf{dk}, \kappa, a, K, \omega_f)$ as the secret key where ω_f is the concise verification information for f. Let $\sigma = (T, U, X, Y, \Lambda)$ be the purported authentication tag for ciphertext μ.
- First, compute $\hat{\nu} \leftarrow \hat{\mathsf{H}}_\kappa(\mu)$.
- Next, run the online closed-form efficient algorithm of F on Δ, to compute $W \leftarrow \mathsf{CFEval}^{\mathsf{on}}_{\Delta}(K, \omega_f)$.
- Finally, if $\deg(f) = 2$, check the following equations:

$$T = U = \hat{\nu} \tag{6}$$
$$X = Y \tag{7}$$
$$W = T^{-1} \cdot (X)^a \cdot (\Lambda)^{a^2} \tag{8}$$

  If $\deg(f) = 1$, replace equation (6) with $e(T, h) = e(g, U) \wedge (T, U) = \hat{\mathsf{H}}(\mu)$, equation (7) with $e(X, h) = e(g, Y)$, and equation (8) with $W = \hat{H}(\mu) \cdot e(X, h)^a$.
  If all equations are satisfied set $acc = 1$ (accept). Otherwise, set $acc = 0$ (reject).
- If $acc = 1$, then set $x' \leftarrow \mathsf{BV.Dec}_{\mathsf{dk}}(\mu)$. Otherwise set $x' = \perp$. Finally, return (acc, x').

Theorem 6. *If* F *is a pseudorandom function,* $\hat{\mathsf{H}}$ *is a collision-resistant homomorphic hash function and BGV is a semantically secure homomorphic encryption scheme, then* $\mathcal{VC}_{\mathsf{quad}}$ *is correct, adaptively secure and input private.*

Proof. For correctness, if both the client and the server are honest, then we show that all three verification equations are satisfied.

Let $\sigma_x = (\Delta, \mu, \sigma_1, \ldots, \mu_t, \sigma_t)$ be an input encoding as generated by **ProbGen** on $\boldsymbol{x} = (x_1, \ldots, x_t)$. and let $f : \mathbb{F}_q^t \to \mathbb{F}_q$ be an arithmetic circuit of degree-2. Without loss of generality, parse

$$f(Z_1, \ldots, Z_t) = \sum_{i,j=1}^{t} \eta_{i,j} \cdot Z_i \cdot Z_j + \sum_{k=1}^{t} \eta_k \cdot Z_k, \tag{9}$$

for some $\eta_{i,j}, \eta_k \in \mathbb{F}_q$. Define $\hat{f} : (\mathbb{G}_1 \times \mathbb{G}_2)^t \to \mathbb{G}_T$ as the compilation of f on group elements as:

$$\hat{f}(A_1, B_1, \ldots, A_t, B_t) = \prod_{i,j=1}^{t} e(A_i, B_j)^{\eta_{i,j}} \cdot \prod_{k=1}^{t} e(A_k, h)^{\eta_k}, \tag{10}$$

Correctness of verification equation (6) is as follows

$$\hat{\mathsf{H}}_{\alpha,\beta}(\mu) = e(g,h)^{\mathsf{H}_\kappa(\mathsf{BGV.Eval}(\mathsf{ek}, f, \mu_1, \ldots, \mu_t))} = e(g,h)^{f(H_{\alpha,\beta}(\mu_1), \ldots, H_{\alpha,\beta}(\mu_t))} = \hat{f}(T_1, U_1, \ldots, T_t, U_t) = T,$$

where the equality in the first line holds by the fact that $\hat{\mathsf{H}}$ is homomorphic; the first equality in the second line holds by equation (10), by construction of T_i and U_i in **ProbGen** (i.e., from that $e(T_i, h) = e(g, U_i)$), and by the fact that $\mathbb{F}_q$ is isomorphic to the bilinear groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$; the second equality in the second line holds by the definition of $\mathsf{GateEval}$ over the (T_i, U_i) pairs. Also, the equality $T = U$ (over $\mathbb{G}_T$) holds in equation (6) by construction of T_i and U_i in **ProbGen**.

To see correctness of equation (7) we first observe that by construction of **ProbGen** we have that $DLog_g(X_i) = DLog_h(Y_i)$ and $DLog_g(T_i) = DLog_h(U_i)$. Second, we observe that such invariant is preserved after every execution of $\mathsf{GateEval}$.

To see correctness of equation (8) we proceed in an inductive way. Observe that one can see the function $f(z_1, \ldots, z_t)$ as the composition of two functions $f_g(f_1(z_1, \ldots, z_t), f_2(z_1, \ldots, z_t))$ in the last gate f_g of f. More generally, one can see every gate f_g of f as computing the function f_g (addition or multiplication) of the two input functions.

So, we first note that by construction every $\sigma_i = (T_i, U_i, X_i, Y_i, \Lambda_i)$ satisfies equation (8) with respect to the projection function $\pi_i(z_1, \ldots, z_t) = z_i$:

$$W_i = \hat{\pi}_i(R_1, S_1, \ldots, R_t, S_t) = e(R_i, h) = e(T_i, h)e(X_i, h)^a = e(T_i, h)e(R_i T_i^{-1}, h)$$

Next, assume that for $i = 1, 2$ $\sigma_i = (T_i, U_i, X_i, Y_i, \Lambda_i)$ satisfies equation (8) with respect to $f_i(z_1, \ldots, z_t)$. We show below that $\sigma = \mathsf{GateEval}(f_g, \sigma_1, \sigma_2)$ satisfies equation (8) with respect to $f = f_g(f_1, f_2)$. For $i = 1, 2$ let $W_i = \hat{f}_i(R_1, S_1, \ldots, R_t, S_t)$ and let $\hat{W} = \hat{f}_g(W_1, W_2)$. In the case where f_g is an addition or a multiplication by constant, it is easy to verify that

$$T^{-1} \cdot (X)^a \cdot (\Lambda)^{a^2} = \hat{f}_g(T_1, T_2)^{-1} \cdot (\hat{f}_g(X_1, X_2))^a \cdot (\hat{f}_g(\Lambda_1, \Lambda_2))^{a^2} = \hat{f}_g(W_1, W_2)$$

If f_g is a multiplication, the inductive assumption essentially says that $X_i^a = W_i T_i^{-1}$ and $Y_i^a = \tilde{W}_i U_i^{-1}$ where $W_i \in \mathbb{G}_1$ and $\tilde{W}_i \in \mathbb{G}_2$ is such that $e(W_i, h) = e(g, \tilde{W}_i)$. Therefore, we have:

$$\begin{aligned} T^{-1} \cdot (X)^a \cdot (\Lambda)^{a^2} &= e(T_1, U_2)^{-1} \cdot [e(X_1, U_2)e(U_1, X_2)]^a \cdot [e(X_1, Y_2)]^{a^2} \\ &= e(T_1, U_2)^{-1} \cdot [e(W_1 T_1^{-1}, U_2)e(U_1, \tilde{W}_2 T_2^{-1})]e(W_1 T_1^{-1}, \tilde{W}_2 U_2^{-1}) \\ &= e(W_1, \tilde{W}_2) = W \end{aligned}$$

To prove security we show the following sequence of games.

Game 0: this is the experiment $\mathbf{Exp}_{\mathcal{A}}^{Adap-Verif}[\mathcal{VC}_{\mathsf{quad}}, \lambda]$.

Game 1: this is like Game 0, except that when answering verification queries W is computed as $\hat{f}(R_1, S_1, \ldots, R_t, S_t)$, instead of using the closed-form efficient algorithm. By the correctness of the closed-form efficient PRF Game 1 is identically distributed to Game 0.

Game 2: this game is the same as Game 1, except that the PRF is replaced by the function $\mathcal{R}(\Delta, i) = (g^r, h^r)$ in which r is the output of a truly random function $\mathcal{R}' : \{0,1\}^* \times [1, t] \to \mathbb{F}_q$. Via a simple reduction to the security of the PRF it is possible to show that Game 2 is computationally indistinguishable from Game 1.

Game 3: this is like Game 2 except that the challenger answers with 0 (reject) every verification query $\sigma_y = (\Delta, \mu, \sigma)$ in which Δ was never chosen in a **ProbGen** query during the experiment. It is possible to show that Game 3 is statistically close to Game 2 information theoretically. To see this, observe that when verifying such a query the value W is $= \hat{f}(R_1, S_1, \ldots, R_t, S_t)$ which can also be written as $W = e(g, h)^{f(r_1, \ldots, r_t)}$. Since Δ was never queried before, every $r_i = \mathcal{R}'(\Delta, i)$ is freshly random. Hence, by Schwartz-Zippel Lemma we have that the probability of $f(r_1, \ldots, r_t)$ hitting any given value of $\mathbb{F}_q$ is bounded by $2/q$.

Game 4: this game is the same as Game 3 except for the following changes. First, on every verification query $\sigma_y = (\Delta, \mu, \sigma)$ the challenger immediately rejects if equations (6) or (7) are not satisfied. Otherwise, assume that Δ was previously generated in a **ProbGen** query (otherwise the query is rejected as well by the modification in game 3), and let $\tilde{\mu}_1, \tilde{\sigma}_1, \ldots, \tilde{\mu}_t, \tilde{\sigma}_t$ be the corresponding values obtained in that **ProbGen** query. From such values compute $\tilde{\sigma} = (\tilde{T}, \tilde{U}, \tilde{X}, \tilde{Y}, \tilde{\Lambda})$ and $\tilde{\mu} = \mathsf{BGV.Eval}(\mathsf{ek}, \tilde{\mu}_1, \ldots, \tilde{\mu}_t)$ as in the **Compute** algorithm. Next, if $\sigma = \tilde{\sigma}$ (component-wise) the challenger accepts. Otherwise, if $\sigma \neq \tilde{\sigma}$ it checks if

$$T \cdot (X)^a \cdot \Lambda^{a^2} = \tilde{T} \cdot (\tilde{X})^a \cdot (\tilde{\Lambda})^{a^2} \tag{11}$$

and accepts if this equation is satisfied. Otherwise, it rejects.

By carefully analyzing the changes introduced in Game 3, we note that these are only syntactic modifications. In particular, by the correctness of **Compute** replacing the check of equation (8) with the equation (11) above leads to identically distributed answers: by correctness $\tilde{T} \cdot (\tilde{X})^a \cdot (\tilde{\Lambda})^{a^2}$ is equal to the same W used to verify σ_y. Therefore, Game 4 is identically distributed as Game 3.

Game 5: this game is like Game 4, except that if $\sigma \neq \tilde{\sigma}$ and equation (11) is satisfied, then Game 5 answers the query with 0 (reject), and sets an (initially false) flag bad to true.

It is easy to see that Game 5 is identical to Game 4 up to the event that the flag bad is set to true. Let Bad_5 be such event. We show in Claim 1 that $\Pr[\mathsf{Bad}_5]$ is negligible (information-theoretically).

Game 6: this game is the same as Game 5 except for a change in answering the following verification queries. Let $\sigma_y = (\Delta, \mu, \sigma)$ be a query such that: Δ was previously obtained from **ProbGen**, $\sigma = \tilde{\sigma}$ (as described in Game 4), and bad was not true. Let T, U be the values in σ, and without loss of generality, assume $T = U \in \mathbb{G}_T$. Since σ is verified correctly, by equation (6) we have $T = \hat{\mathsf{H}}_\kappa(\mu)$.

Let $\tilde{\sigma}$ and $\tilde{\mu}$ be the values computed as in Game 4. Since $\mathsf{bad} \neq \mathsf{true}$, we have that $\tilde{T} = T$ and by correctness we have that $\tilde{T} = \hat{\mathsf{H}}_\kappa(\tilde{\mu})$. At this point, if the ciphertext μ provided by the adversary is such that $\mu \neq \tilde{\mu}$, then the challenger answers 0 (reject) and sets an initially false flag $\mathsf{bad}' \leftarrow \mathsf{true}$.

It is easy to see that Game 6 is identical to Game 5 up to the event that the flag bad' is set to true. If we call Bad_6 such event, we observe that occurrence of Bad_6 immediately implies a collision in the function $\hat{\mathsf{H}}$. Therefore, it is possible to show that the probability $\Pr[\mathsf{Bad}_6]$ is negligible under the collision resistance of the function $\hat{\mathsf{H}}$. The reduction is straightforward once observed that one can simulate the computations of $\hat{\mathsf{H}}$ in Game 6 (for both **ProbGen** and **Verify** queries) by using the public key of $\hat{\mathsf{H}}$.

To complete the proof of the theorem below in the following Claim we show that $\Pr[\mathsf{Bad}_5]$ is negligible.

Claim 1 *For any PPT adversary making at most Q verification queries in Game 5 it holds* $\Pr[\mathsf{Bad}_5] \leq \frac{2Q}{q-2Q}$.

Proof. The basic idea of the proof is that in Game 5 the secret value a remains information-theoretically hidden to the adversary. For every new authentication query, indeed notice that the pairs (X_i, Y_i) generated in **ProbGen** (in Game 5) are uniformly distributed over the set of pairs (X, Y) such that $e(X, h) = e(g, Y)$. Hence, a is only used to answer verification queries and more precisely to check equation (11).

For every verification query $i = 1$ to Q, let B_i be the event that bad is set to true in the i-th query but not before. By definition of B_i and by a union bound we have that $\Pr[\mathsf{Bad}_5] \leq \sum_{i=1}^{Q} \Pr[B_i | \neg B_1 \wedge \cdots \wedge \neg B_{i-1}]$. Now, observe that at the first verification query a is uniformly distributed over $\mathbb{F}_q$, and thus $\Pr[B_1] \leq 2/q$. Then, at every query, if bad is not set true it means that equation (11) was not satisfied. Namely, one can exclude at most two possible values of a, and assume to add such values to a set A'. At the i-th query, if we condition on that B_j never occurred in the previous queries we have that a is uniformly distributed over $\mathbb{F}_q \setminus A'$, and A' contains at most $2(i-1)$ elements. Hence, we have that $\Pr[B_i | \neg B_1 \wedge \cdots \wedge \neg B_{i-1}] \leq \frac{2}{q-2(i-1)}$. Therefore, we finally obtain that $\Pr[\mathsf{Bad}_5] \leq \frac{2Q}{q-2Q}$. □

Finally, input privacy follows by defining a series of hybrid games in which we progressively modify the output of **ProbGen** queries in such a way that every ciphertext μ_i returned by the **ProbGen** oracle in the privacy game is replaced by an encryption of 0. By the semantic security of the BGV encryption scheme, every two consecutive pairs of games are indistinguishable to a PPT adversary. Moreover, note that in this proof the verification oracle can be easily simulated without the knowledge of the BGV decryption key. □

A Variant with Function Privacy. In this section we show that the previous scheme for multivariate quadratic polynomialscan be modified in order to achieve function privacy. Precisely, if we see the polynomial $f : \mathbb{F}_p^t \to \mathbb{F}_p$ as an arithmetic circuit, we can hide all the constants c occurring in multiplication-by-constant gates that take inputs of degree 1.

To obtain this construction, the scheme $\mathcal{VC}_{\mathsf{quad}}$ is modified into $\mathcal{VC}^*_{\mathsf{quad}}$ as follows. Let $f : \mathbb{F}_p^t \to \mathbb{F}_p$ be a degree-2 arithmetic circuit. For every gate f_g in f which is a multiplication by a constant $c \in \mathbb{F}_p$, compute $\gamma \xleftarrow{\$} \mathsf{BGV.Enc}_{\mathsf{pk}}(c)$, $\nu_c \leftarrow \mathsf{H}_\kappa(\gamma)$ and $(T_c, U_c) \leftarrow \hat{\mathsf{H}}_K(\gamma)$. Then we define the function $f_{pri} : \mathbb{F}_q^t \to \mathbb{F}_q$ as the same as f except that every multiplication by a constant c is replaced by a multiplication by the corresponding $\nu_c \in \mathbb{F}_q$ computed as described above. $f_{pub} : \mathbb{F}_q^t \to \mathbb{F}_q$ is the same as f except that every multiplication by a constant c is replaced by the corresponding (T_c, U_c) computed as above.

In $\mathcal{VC}^*_{\mathsf{quad}}$, $\mathbf{KeyGen}(f, \lambda)$ algorithm proceeds as before except that the value ω_f is computed from f_{pri}, i.e., $\omega_f \leftarrow \mathsf{CFEval}^{\mathsf{off}}_{\boldsymbol{\tau}}(K, f_{pri})$, and EK contains f_{pub} instead of f. In $\mathbf{Compute}_{PK}(\sigma_x)$ the difference is that we slightly modify $\mathsf{GateEval}$ for multiplication-by-constant gates as follows:

Multiplication by constant. Let $\sigma = (T^{(1)}, U^{(1)}, X^{(1)}, Y^{(1)}, \Lambda^{(1)}) \in (\mathbb{G}_1 \times \mathbb{G}_2)^2 \times \mathbb{G}_T$ with $\Lambda^{(1)} = 1$, and recall that the constant $c \in \mathbb{F}_p$ is (publicly) encoded as $(T_c, U_c) \in \mathbb{G}_1 \times \mathbb{G}_2$. We compute $\sigma = (T, U, X, Y, \Lambda)$ as:

$$T = e(T^{(1)}, U_c),\ U = e(T_c, U^{(1)}),\ X = e(X^{(1)}, U_c),\ Y = e(T_c, Y^{(1)})$$

Theorem 7. *Let $\mathcal{F}$ be the class of arithmetic circuits in which multiplication by constants occur only for inputs of degree 1. If F is a pseudorandom function, $\hat{\mathsf{H}}$ is a collision-resistant homomorphic hash function and BGV is a semantically secure homomorphic encryption scheme, then $\mathcal{VC}^*_{\mathsf{quad}}$ is correct, adaptively secure, input private and function private with respect to $\mathcal{F}$.*

The proof of correctness, adaptive security and input privacy is essentially the same as in Theorem 6. Function privacy easily follows by defining a series of hybrid games in which in **KeyGen** we progressively replace every ciphertext γ encrypting a constant c by an encryption of 0. By the semantic security of the BGV encryption scheme, every two consecutive pairs of games are indistinguishable to a PPT adversary.

6 Computing Polynomials of Large Degree

We now look at the simpler task of delegation of univariate polynomials of large degree t, evaluated on values $x \in \mathbb{F}_p$ that do not need to be private.

We still use BGV [18] for q prime, but without using its full power: we only need its additive homomorphic property. We then apply the technique of [11] on the hashing $\mathsf{H}_{\alpha,\beta}(\gamma_i)$, where γ_i is the BGV encryption of g_i, and $g(x) = \sum_{i=0}^{t} g_i \cdot x^i$ is the function to compute. Again, acceptance or rejection by the client depends *only* on the correct execution of the computation over the ciphertexts. No useful decryption query is performed.

More in detail, we assume that, as in [11], the client and the server agreed on a group G of prime order q in which the discrete logarithm problem is hard, and on a generator g for G.

Before the specifications of the scheme, we shall make a final observation: due to the noise growth of the BGV scheme summarized in lemma 1 (for $D = 1$), in order to achieve correctness of the result to be decoded by the client, we need q to be larger than $2 \cdot p \cdot x^t \cdot \sigma \cdot n^{1.5}$, where x is the evaluation point of the scheme. In the worst case, $x = p$, but we prefer to separate x and p, to state a more general result, that can fit better for applications in which x is smaller than p.

The scheme $\mathcal{VC}_{\mathsf{poly}}$ is specified as follows:

KeyGen$(g_0, \ldots, g_t, \lambda) \to (PK, SK)$:
- Run $\mathsf{BGV.ParamGen}(\lambda)$, $(\mathsf{pk}, \mathsf{dk}) \leftarrow \mathsf{BGV.KeyGen}()$.
- Specify a group $(G, \cdot)$ of order q and a generator g.
- Sample a uniform MAC key $c \leftarrow \mathbb{F}_q$, uniform $k_0, k \in \mathbb{F}_q$,
- Sample uniform $\alpha \xleftarrow{\$} R_q$, $\beta \xleftarrow{\$} \mathbb{F}_q$,
- Compute $\gamma_i \leftarrow \mathsf{BV.Enc}_{\mathsf{pk}}(g_i)$, $T_i \leftarrow c \cdot \mathsf{H}_{\alpha,\beta}(\gamma_i) + k^i \cdot k_0$, $G_{T,i} \leftarrow g^{T_i}$,
- Set $PK = (\mathsf{pk}, G, g, \gamma_0, G_{T,0}, \ldots, \gamma_d, G_{T,t})$ and $SK = (\mathsf{pk}, G, g, \mathsf{dk}, c, \alpha, \beta, k, k_0)$.

ProbGen$_{SK}(x) \to \sigma_x, \tau_x$:
- Set $\sigma_x = x$, and $\tau_x = x$.

Compute$_{PK}(\sigma_x = x) \to \sigma_y$:
- Compute $\gamma \leftarrow \sum_{i=0}^{t} x^i \cdot \gamma_i$, and $G_T \leftarrow \prod_{i=0}^{t} (G_{T,i})^{x^i}$,
- Set $\sigma_y = \gamma, G_t$

Verify$_{SK}(\sigma_y = \gamma, G_T, \tau_x = x) \to (acc, a')$:
- Compute the element X in $\mathbb{F}_q$ such that:

$$X = \left((x \cdot k)^{t+1} - 1\right) \cdot (x \cdot k - 1)^{-1},$$

- If $G_T \neq \left(g^{\mathsf{H}_{\alpha,\beta}(\gamma)}\right)^c \cdot \left(g^{k_0}\right)^X$, reject. Otherwise, accept and compute $a' \leftarrow \mathsf{BV.Dec}_{\mathsf{dk}}(\gamma)$.

Theorem 8. *The scheme $\mathcal{VC}_{\mathsf{poly}}$ is correct, adaptively secure, and function private.*

Proof. For correctness, when both the client and the server are honest the verification step passes, because of the following reasoning. Notice that:

$$G_T = \prod_{i=0}^{t} (G_{T,i})^{x^i} = \prod_{i=0}^{t} \left(g^{T_i}\right)^{x^i} = g^{\sum_{i=0}^{t} T_i \cdot x^i} = g^{\sum_{i=0}^{t} \left(c \cdot \mathsf{H}_{\alpha,\beta}(\gamma_i) + k^i \cdot k_0\right) \cdot x^i} =$$
$$= g^{c \cdot \sum_{i=0}^{t} \mathsf{H}_{\alpha,\beta}(\gamma_i) \cdot x^i + \sum_{i=0}^{t} k^i \cdot k_0 \cdot x^i} = g^{c \cdot \mathsf{H}_{\alpha,\beta}(\gamma)} \cdot g^{k_0 \cdot \sum_{i=0}^{t} (k \cdot x)^i},$$

by construction and using the fact that $\mathsf{H}_{\alpha,\beta}$ is an homomorphism of rings (last step). Notice now that:

$$\sum_{i=0}^{t} (k \cdot x)^i = \left((x \cdot k)^{t+1} - 1\right) \cdot (x \cdot k - 1)^{-1} = X,$$

so $G_T = g^{c \cdot \mathsf{H}_{\alpha,\beta}(\gamma)} \cdot g^{k_0 \cdot X} = \left(g^{\mathsf{H}_{\alpha,\beta}(\gamma)}\right)^c \cdot g^{k_0 \cdot X}$, therefore the check in the verification step passes. Moreover, the client retrieves the correct value of the computation, because

$$\mathsf{BGV.Dec}_{\mathsf{dk}}(\gamma) = \mathsf{BGV.Dec}_{\mathsf{dk}}\left(\sum_{i=0}^{t} \gamma_i \cdot x^i\right) = \mathsf{BGV.Dec}_{\mathsf{dk}}\left(\sum_{i=0}^{t} \mathsf{BGV.Enc}_{\mathsf{pk}}(g_i) \cdot x^i\right) = \sum_{i=0}^{t} g_i \cdot x^i.$$

Notice that the bound on x makes the last equality hold, since that bound respects the inequality in Theorem 1 (for $D = 1$).

For adaptive security, notice that $\sigma_x = x$, therefore the scheme is split. Moreover, adaptive security reduces to regular security, as the adversary has full knowledge on the encodings that can be queried in the adaptive game, even before seeing the public key of the verification scheme. Regular security can be proven in a game-based fashion, using a similar approach to the one in [11], Section 5.2:

Game_0**:** The experiment in Definition 2.
Game_1**:** As Game_0, but in the verification the value X is computed via $X \leftarrow \sum_{i=0}^{t} (k \cdot x)^i$.
Game_2**:** As Game_1, but replacing the value $k^i \cdot k_0$ by a uniform $r_i \in \mathbb{F}_q$, and the verification check correspondingly to $G_T \neq \left(g^{\mathsf{H}_{\alpha,\beta}(\gamma)}\right)^c \cdot g^{\sum_{i=0}^{t} r_i \cdot x^i}$, which maintains correctness.

As in [11], the change between Game_1 and Game_0 is merely syntactical, so the advantage of a cheating server is the same. Moreover, the advantage of a cheating prover in Game_2 is negligibly close to the one in Game_1, since the function $\varphi_{k_0,k} : i \mapsto g^{k^i \cdot k_0}$ is a pseudorandom function (based on the Strong Diffie-Helmann assumption), for uniform $k_0, k \in \mathbb{F}_q$.

An adversary Game_2 if it provides γ' and G'_T that pass the verification check, but such that $\mathsf{BGV.Dec}_{\mathsf{dk}}(\gamma') \neq \mathsf{BGV.Dec}_{\mathsf{dk}}(\gamma)$. By the correctness of the scheme, $\gamma' \neq \gamma$. Let $\gamma' = \gamma + \delta$, with $0 \neq \delta \in R_q[Y]$. Moreover, let $G'_T = G_T \cdot g^d$, for $d \in \mathbb{F}_q$. Since the verification check passes, then

$$G_t \cdot g^d = \left(g^{\mathsf{H}_{\alpha,\beta}(\gamma+\delta)}\right)^c \cdot g^{\sum_{i=0}^{t} r_i \cdot x^i}.$$

Notice that the right hand side is equal to $g^{c \cdot \mathsf{H}_{\alpha,\beta}(\gamma+\delta) + \sum_{i=0}^{t} r_i \cdot x^i} = g^{c \cdot \mathsf{H}_{\alpha,\beta}(\gamma) + c \cdot \mathsf{H}_{\alpha,\beta}(\delta) + \sum_{i=0}^{t} r_i \cdot x^i}$, since $\mathsf{H}_{\alpha,\beta}$ is a ring homomorphism. This means that the right hand side equals $G_t \cdot g^{c \cdot \mathsf{H}_{\alpha,\beta}(\delta)}$, by the correctness of the scheme, so the above equality holds if and only if $g^d = g^{c \cdot \mathsf{H}_{\alpha,\beta}(\delta)}$. There are two scenarios in which this happens:

1. $d \neq 0$. In this case the adversary has to provide a polynomial $\gamma' = \gamma + \delta$, where $\mathsf{H}_{\alpha,\beta}(\delta) = d/c$, which happens with negligible probability, since d/c is uniform (because all the information about c given to the adversary is hidden by the r_i).
2. $d = 0$. In this case, the adversary has to provide $\gamma' = \gamma + \delta$, where $\mathsf{H}_{\alpha,\beta}(\delta) = 0$, which is equivalent to breaking the universal one-wayness of H, and happens with negligible probability.

Combining the two scenarios, and the various games, we get that any adversary wins the security game with negligible probability.

Function privacy is guaranteed by a reduction to the semantic security of the scheme, similar to the reduction done in the proof of Theorem 9. □

7 Computing Linear Combinations

In this section we give a specialized construction for a particular setting. Namely, we are interested in getting an input private *and* function private *split* scheme that is not necessarily outsourceable, but that works in the streaming model. Our scheme is focused on the delegation of the computation of secret linear functions on encrypted data – linearity over R_p, i.e. a function g to be computed is described by $g_0, \dots g_d \in R_p$, and on an input $x_0, \dots, x_d \in R_p$, g outputs $\sum_{i=0}^{d} g_i \cdot x_i$. Again, we use BGV, but this time we require its somewhat homomorphic property for polynomials of degree $D = 2$ (see lemma 1 for the parameter choice), and do not require q be prime. The scheme $\mathcal{VC}_{\mathsf{LC}}$ is specified as follows.

KeyGen$(g = g_0, \dots g_t, \lambda) \to (PK, SK)$:
- Run $\mathsf{BGV.ParamGen}(\lambda)$, $(\mathsf{pk}, \mathsf{dk}) \leftarrow \mathsf{BGV.KeyGen}()$.
- Sample a uniform MAC key $c \xleftarrow{\$} R_q$, a key k for PRF, uniform values $\alpha \xleftarrow{\$} R_q$, $\beta \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$, and compute $r_i \leftarrow \mathsf{PRF}_k(i)$.
- Compute $\gamma_i \leftarrow \mathsf{BGV.Enc}_{\mathsf{pk}}(g_i)$, and $T_i \leftarrow c \cdot \mathsf{H}_{\alpha,\beta}(\gamma_i) + r_i \in \mathbb{Z}/q\mathbb{Z}$,
- Set $PK = (\mathsf{pk}, \gamma_0, T_0, \dots, \gamma_t, T_t)$, and $SK = (\mathsf{pk}, \mathsf{dk}, c, k, \alpha, \beta)$.

ProbGen$_{SK}(x = x_0, \dots, x_t) \to \sigma_x, \tau_x$: for $i = 0, \dots, t$
- If σ_x, τ_x are not defined: initialize σ_x as the zero vector in R_q^{t+1}, and $\tau_x \leftarrow 0 \in \mathbb{Z}/q\mathbb{Z}$.
- Compute $\mu_i \leftarrow \mathsf{BGV.Enc}_{\mathsf{pk}}(x_i)$.
- Compute $r_i \leftarrow \mathsf{PRF}_k(i)$, as in the keygen algorithm.
- Set the $i+1$th entry of σ_x as μ_i.
- Update τ_x by adding $r_i \cdot \mathsf{H}_{\alpha,\beta}(\mu_i)$ to its previous value (i.e. $\tau_x += r_i \cdot \mathsf{H}_{\alpha,\beta}(\mu_i)$).

Compute$_{PK}(\sigma_x = \mu_0, \dots, \mu_t) \to \sigma_y$:
- Compute $\gamma \leftarrow \sum_{i=0}^{t} \gamma_i \cdot \mu_i \in R_q[Y]$, and $\tau \leftarrow \sum_{i=0}^{d} T_i \cdot \mu_i$.
- Set $\sigma_y = \gamma, \tau$.

Verify$_{SK}(\tau_x, \sigma_y = \gamma, \tau) \to (acc, y')$: If $\mathsf{H}_{\alpha,\beta}(\tau) \neq c \cdot \mathsf{H}_{\alpha,\beta}(\gamma) + \tau_x$: reject. Otherwise: accept, and compute $y' \leftarrow \mathsf{BGV.Dec}_{\mathsf{dk}}(\gamma)$.

Theorem 9. *The scheme $\mathcal{VC}_{\mathsf{LC}}$ is correct, adaptively secure, function private, input private.*

Proof. For correctness, if both the client and the server are honest, the client accepts:

$$\begin{aligned}
\mathsf{H}_{\alpha,\beta}(\tau) &= \sum_{i=0}^{t} T_i \cdot \mathsf{H}_{\alpha,\beta}(\mu_i) = \sum_{i=0}^{t} (c \cdot \mathsf{H}_{\alpha,\beta}(\gamma_i) + r_i) \cdot \mathsf{H}_{\alpha,\beta}(\mu_i) \\
&= \sum_{i=0}^{t} (c \cdot \mathsf{H}_{\alpha,\beta}(\gamma_i) \cdot \mathsf{H}_{\alpha,\beta}(\mu_i) + r_i \cdot \mathsf{H}_{\alpha,\beta}(\mu_i)) = \sum_{i=0}^{t} (c \cdot \mathsf{H}_{\alpha,\beta}(\gamma_i \cdot \mu_i) + r_i \cdot \mathsf{H}_{\alpha,\beta}(\mu_i)) \\
&= c \cdot \left(\sum_{i=0}^{t} \mathsf{H}_{\alpha,\beta}(\gamma_i \cdot \mu_i) \right) + \sum_{i=0}^{t} r_i \cdot \mathsf{H}_{\alpha,\beta}(\mu_i) = c \cdot \mathsf{H}_{\alpha,\beta} \left(\sum_{i=0}^{t} \gamma_i \cdot \mu_i \right) + \tau_x = c \cdot \mathsf{H}_{\alpha,\beta}(\gamma) + \tau_x.
\end{aligned}$$

Where all the equalities hold by the fact that $\mathsf{H}_{\alpha,\beta}$ is a ring homomorphism and addition and multiplication of polynomials distribute. Moreover, the output y' is the desired one, since it equals:

$$\begin{aligned}
\mathsf{BGV.Dec}_{\mathsf{dk}}(\gamma) &= \mathsf{BGV.Dec}_{\mathsf{dk}} \left(\sum_{i=0}^{t} \gamma_i \cdot \mu_i \right) \\
&= \mathsf{BGV.Dec}_{\mathsf{dk}} \left(\sum_{i=0}^{t} \mathsf{BGV.Enc}_{\mathsf{pk}}(g_i) \cdot \mathsf{BGV.Enc}_{\mathsf{pk}}(x_i) \right) = \sum_{i=0}^{t} g_i \cdot x_i.
\end{aligned}$$

For adaptive security, we show firstly that the scheme is split: define $PK_E = SK_E = \mathsf{pk}$, and notice that σ_x consists of $\mu_i = \mathsf{BV.Enc}_{\mathsf{pk}}(w_i)$, which are independent of f. Therefore, in the security experiment the challenger can answer the encoding queries before the adversary choses the function to attack on. At that point the challenger computes the remaining part of the public and secret key. Secondly, we show that no adversary wins the adaptive security game with non-negligible probability. Notice that the values c, α, β are statistically hidden to the server, because $T_i = c \cdot \mathsf{H}_{\alpha,\beta}(\gamma_i) + r_i$, and r_i is statistically close to uniform by the security property of the PRF. Now, for the sake of contradiction, assume that there is an adversary who can win the adaptive security game with non-negligible probability, i.e. suppose that a server could provide γ', τ' with $\gamma' \neq \gamma$, and such that the verification passes. This would imply that $\mathsf{H}_{\alpha,\beta}(\tau') = c \cdot \mathsf{H}_{\alpha,\beta}(\gamma') + \sum_{i=0}^{t} r_i \cdot \mu_i$, and that $\mathsf{H}_{\alpha,\beta}(\tau') - \mathsf{H}_{\alpha,\beta}(\tau) = c \cdot (\mathsf{H}_{\alpha,\beta}(\gamma') - \mathsf{H}_{\alpha,\beta}(\gamma))$, which happens if and only if $\mathsf{H}_{\alpha,\beta}(\tau' - \tau - c \cdot (\gamma' - \gamma)) = 0$ (by the fact that $\mathsf{H}_{\alpha,\beta}$ is an homomorphism of rings). There are two possible scenarios in which this happens:

1. $\tau' - \tau = c \cdot (\gamma' - \gamma)$. If this were the case, the server could compute c in polynomial time, by dividing any $\mathbb{Z}/q\mathbb{Z}$-coordinate of $\tau' - \tau$ by the corresponding $\mathbb{Z}/q\mathbb{Z}$-coordinate of $\gamma' - \gamma$ (this division in $\mathbb{Z}/q\mathbb{Z}$ is indeed possible, as we are guaranteed that $\tau' - \tau$ is the product of c by $\gamma' - \gamma$). However, computing c in polynomial time is impossible, since c is statistically hidden to the server, as argued earlier. That means that this case happens with negligible probability.
2. $\tau' - \tau \neq c \cdot (\gamma' - \gamma)$, but $\mathsf{H}_{\alpha,\beta}(\tau' - \tau - c \cdot (\gamma' - \gamma)) = 0$. This is equivalent to breaking the one-wayness of H, and happens with negligible probability.

Combining the above, we obtain that the overall probability that an adversary wins the adaptive security game is negligible.

To show function privacy, we create a series of games:

Game_0**:** This is the regular privacy game.
Game_{i+1}**:** Identical to Game_i, but replace g_i with 0 (for $i = 0, \ldots, t$).

Notice that for any game the verification oracle can be provided trivially, without the usage of sk. By the semantic security of the BGV scheme, an adversary who wins Game_i with a given probability p wins Game_{i+1} with probability negligibly close to p (otherwise one could turn the adversary into an effective attacker to the BGV semantic security). However, Game_{t+1} is completely independent of the function, therefore any adversary wins that game with negligible probability. By the series of hybrids, no adversary can win the original function privacy game.

Input privacy can be proven in a similar fashion, and it is guaranteed by the fact that the verification check is independent fo the encoded values. □

On the Outsourceability of the Scheme. As we hinted in the opening paragraph, this scheme is not outsourceable, but it has the desirable property that the input does not have to be processed at once, and does not need to be stored on the client side during the processing phase. More in detail, the x_i are generated in rounds, and at each round **ProbGen** is called. This means that the vector σ_x is formally initialized as the zero vector, and at round i it is updated by appending μ_i but *only* μ_i is sent to the server at round i. Analogously, τ_x is created and stored by the client at round 0, and at round i it is updated by adding $r_i \cdot \mathsf{H}_{\alpha,\beta}(\mu_i)$. This allows the client to work with a short memory: indeed, creating and sending μ_i requires $O(\log(q))$ storage, which is the same memory needed to create, update, and store τ_x.

Communication Complexity. In case of a sparse linear combination, say in which $\widetilde{d}$ among the x_i are non-zero, the client has two choices:

Maintain full input security: If the client wants to completely hide the weights, then it must pay $O(d)$ to send a ciphertext μ_i for each index, as in the standard scheme.

Improve the communication complexity: It can pay $O(\widetilde{d})$ by simply sending the non-zero weights encrypted (and the indices they correspond to); in this case, however, it reveals which indices are zero in the linear combination. Every piece of information on the non-zero indices is preserved (by a simple reduction to the security of the standard scheme).

8 Computing Linear Functions over the ring $\mathbb{Z}_{2^k}$

In this section we show another scheme that allows to compute multi-variate polynomials of degree-1 over the ring $\mathbb{Z}_{2^k}$. The scheme has properties similar to the scheme in section 5: it is input private, and it allows for constant-time verification. The basic idea is to combine the linearly-homomorphic encryption scheme recently proposed by Joye and Libert [40] with a linearly homomorphic MAC with efficient verification. In particular, for the MAC we use the ideas in [6] to design a MAC that allows us to authenticate linear computations over the group $\mathbb{Z}_N^*$ which is the ciphertext space of the encryption scheme [40]. In particular, a crucial tool to achieve efficiency is a new pseudorandom function with amortized closed-form efficiency whose security relied on the DDH assumption in the subgroup of 2^k-residues of $\mathbb{Z}_N^*$.

A full description of the scheme $\mathcal{VC}_{lin}$ follows:

KeyGen$(f, \lambda) \rightarrow (PK, SK)$:

- Sample two large-enough quasi-safe primes p, q such that $p = 2^k p' + 1$ and $q = 2^k q' + 1$. Set $N = pq$, and sample $y \xleftarrow{\$} J_N$. Let us call $\mathcal{R}_k$ the subgroup of $\mathbb{Z}_N^*$: $\mathcal{R}_k = \{x^{2^k} : x \in \mathbb{Z}_N^*\}$.
- Sample a random $\alpha \xleftarrow{\$} \mathbb{Z}_{\phi(N)}^*$, and run $(K, \mathsf{pp}) \xleftarrow{\$} \mathsf{F.KG}(1^\lambda)$ to obtain the seed and the public parameters of an ACF-efficient PRF $\mathsf{F}_K : \{0,1\}^* \times \{0,1\}^* \rightarrow \mathcal{R}_k$.
- Compute a concise verification information for f by using the offline closed-form efficient algorithm of F, i.e., $\omega_f \leftarrow \mathsf{CFEval}_{\boldsymbol{\tau}}^{\mathsf{off}}(K, f)$.
- Output the secret key $SK = (p, \alpha, K, \omega_f)$, and the public evaluation key $EK = (N, y, k, \mathsf{pp}, f)$.

ProbGen$_{SK}(\boldsymbol{x} = (x_1, \ldots, x_t)) \rightarrow \sigma_x, \tau_x$:

- Choose an arbitrary string $\Delta \in \{0,1\}^\lambda$ as an identifier for the input vector $\boldsymbol{x} \in (\mathbb{Z}_{2^k})^t$.
- For $i = 1$ to t: first sample $r_i \xleftarrow{\$} \mathbb{Z}_N^*$ and compute $c_i = y^{x_i} r_i^{2^k} \mod N$.
  Next, compute $R_i \leftarrow \mathsf{F}_K(\Delta, i)$, and compute $\sigma_i = c_i^\alpha \cdot R_i \mod N$.
- Set $\sigma_x = (\Delta, c_1, \sigma_1, \ldots, c_t, \sigma_t)$, and $\tau_x = \perp$.

It is worth noting that **ProbGen** can work in the so-called streaming model in which every input item x_i can be processed separately.

Compute$_{PK}(\sigma_x) \rightarrow \sigma_y$:

- Let $f = (f_1, \ldots, f_t) \in (\mathbb{Z}_{2^k})^t$ be a linear function and let $\sigma_x = (\Delta, c_1, \sigma_1, \ldots, c_t, \sigma_t)$.
- First, compute $c \leftarrow \prod_{i=1}^t c_i^{f_i} \mod N$ to homomorphically evaluate f over the ciphertexts (c_i).
- Second, compute $\sigma = \prod_{i=1}^t \sigma_i^{f_i} \mod N$ to homomorphically evaluate f over the authentication tags (σ_i).
- Output $\sigma_y = (\Delta, c, \sigma)$.

Verify$_{SK}(\sigma_y = (\Delta, c, \sigma), \tau_x) \rightarrow (acc, x')$:

- Parse $SK = (p, \alpha, K, \omega_f)$ as the secret key where ω_f is the concise verification information for f.
- First, run the online closed-form efficient algorithm of F, to compute $W \leftarrow \mathsf{CFEval}_{\Delta}^{\mathsf{on}}(K, \omega_f)$.

– Next, if the following equation is satisfied set $acc = 1$ (accept). Otherwise, set $acc = 0$ (reject).

$$\sigma = c^{\alpha} \cdot W \mod N \tag{12}$$

– If $acc = 1$, then compute $\left(\frac{c}{p}\right)_{2^k} = c^{p'} \mod N$, and find $x' \in \{0,1\}^k$ such that $[\left(\frac{y}{p}\right)_{2^k}]^{x'} = z$ (see [40] for details).
If $acc = 0$ set $x' = \perp$. Finally, return (acc, x').

Definition 12 (Gap-2^k-Residuosity assumption [40]). *Let $N = pq$ be the product of two quasi-safe primes $p = 2^k p' + 1$ and $q = 2^k q' + 1$. Define $\mathcal{R}_k = \{x^{2^k} : x \in \mathbb{Z}_N^*\}$, and let J_N the subgroup of $\mathbb{Z}_N^*$ of elements with Jacobi symbol 1, and let $QR_N \subset J_N$ be the subgroup of quadratic residues of $\mathbb{Z}_N^*$. Let $x \xleftarrow{\$} \mathcal{R}_k$ and $y \xleftarrow{\$} J_N \setminus QR_N$. We say that the Gap-$2^k$-Residuosity assumption holds if for every PPT adversary $\mathcal{A}$ the following advantage is negligible:*

$$\mathbf{Adv}_{\mathcal{A}}^{Gap-2^k-res}(\lambda) = |\Pr[\mathcal{A}(N,k,x) = 1] - \Pr[\mathcal{A}(N,k,y) = 1]|$$

In [40] it is shown that the Gap-2^k-Residuosity assumption is implied by the more natural quadratic residuosity and special Jacobi symbol assumptions.

Theorem 10. *If F is a pseudorandom function and the Gap-2^k-Residuosity assumption holds, then $\mathcal{VC}_{lin}$ is correct, adaptively secure and input private.*

Proof. For correctness observe that:

$$c^{\alpha} \cdot W \mod N = \left(\prod_{i=1}^{t} c_i^{f_i}\right)^{\alpha} \cdot \left(\prod_{i=1}^{t} \mathsf{F}_K(\Delta, i)^{f_i}\right) = \prod_{i=1}^{t} (c_i^{\alpha} \mathsf{F}_K(\Delta, i))^{f_i} = \prod_{i=1}^{t} \sigma_i^{f_i} = \sigma.$$

Below is a proof sketch for the security property.

Game 0: this is the experiment $\mathbf{Exp}_{\mathcal{A}}^{Adap-Verif}[\mathcal{VC}_{lin}, \lambda]$.

Game 1: this is like Game 0, except that when answering verification queries W is computed as $\prod_{i=1}^{t} R_i^{f_i} \mod N$ with $R_i \leftarrow \mathsf{F}_K(\Delta, i)$, instead of using the online closed-form efficient algorithm. By the correctness of the closed-form efficient PRF, Game 1 is identically distributed to Game 0.

Game 2: this game is the same as Game 1, except that the PRF is replaced by a truly random function $\mathcal{R} : \{0,1\}^* \times [1,t] \to \mathcal{R}_k$. Via a simple reduction to the security of the PRF it is possible to show that Game 2 is computationally indistinguishable from Game 1.

Game 3: this is the same as Game 2 except for changing the distribution of the public key by sampling $y \xleftarrow{\$} \mathcal{R}_k$. Note that at this point the encryption scheme becomes lossy. Also, by the Gap-2^k-Residuosity assumption Game 3 is computationally indistinguishable from Game 2.

Game 4: this is like Game 3 except that the challenger answers with 0 (reject) every verification query $\sigma_y = (\Delta, c, \sigma)$ in which Δ was never chosen in a **ProbGen** query during the experiment (and of course the function f is non-zero).
It is possible to show that, information theoretically, Game 4 is statistically close to Game 3.

Game 5: Change as follows the way to check verification queries (Δ, c, σ) in which Δ was previously generated in a **ProbGen** query (otherwise the query is rejected as well by the modification in the previous game). Let $\tilde{c}_1, \tilde{\sigma}_1, \ldots, \tilde{c}_t, \tilde{\sigma}_t$ be the corresponding values obtained in that

ProbGen query. From such values compute $\tilde{\sigma} = \prod_{i=1}^{t} \tilde{\sigma}_i^{f_i}$ and $\tilde{c} = \prod_{i=1}^{t} \tilde{c}_i^{f_i}$ as in the **Compute** algorithm. Check if

$$\sigma/\tilde{\sigma} = (c/\tilde{c})^{\alpha}$$

and if so accept, otherwise reject.

Note that by correctness, the above check is equivalent to the real verification. Hence Game 5 is identically distributed to Game 4.

Game 6: Simulate all **ProbGen** queries without using α, i.e., sample directly $\sigma_i \xleftarrow{\$} \mathcal{R}_k$. Note that from Game 3 the ciphertexts c_i are in $\mathcal{R}_k$ (since $y \in \mathcal{R}_k$. Hence, Game 6 is identically distributed to Game 5.

It is worth noting that now α is used only for verification. If we imagine to sample α at the time of the first verification query, then at that point α is uniformly distributed over $\mathbb{Z}^*_{\phi(N)}$.

Game 7: All verification queries where $(c, \sigma) = (\tilde{c}, \tilde{\sigma})$ are directly answered with 1, i.e., without using α. If $c \neq \tilde{c}$, then answer with reject. We claim that Game 7 is statistically close to Game 6.

In particular, if we change only the answer to the first query, the distance is $\epsilon = \Pr[(c/\tilde{c})^{\alpha} = \sigma/\tilde{\sigma}]$ taken over the random choice of α that is $\frac{1}{|\mathbb{Z}^*_{\phi(N)}|} = \frac{1}{2^{k-1}(p'-1)(q'-1)}$, which is negligible. By extending this argument to all possible Q verification queries the distance is $\leq \frac{Q}{|\mathbb{Z}^*_{\phi(N)}|-Q}$.

Input privacy follows by defining a hybrid game in which the public element y is sampled uniformly in $\mathcal{R}_k$. Such change is computationally indistinguishable by the Gap-2^k-Residuosity assumption. Then, it is easy to see that the encryption is lossy, and thus no information on the plain texts is leaked. Moreover, note that in this proof the verification oracle can be easily simulated without the knowledge of the factorization of N. □

9 Applications

9.1 Statistics on Encrypted Data Sets

Consider the problem in which a client stores several large data-sets $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ on a server, and wants to compute a collection of statistics on the outsourced data in a private and verifiable way. By using our scheme for multi-variate quadratic polynomials of section 5, we can provide efficient solutions for the computation of several statistical functions, such as average, variance, standard deviation, RMS, covariance, linear regression, Pearson's and uncentered correlation coefficient. Below we show how these statistical functions can be decomposed into simpler non-rational functions that we can authenticate using our scheme. For vectors $\boldsymbol{x} = (x_1, \ldots, x_t)$, $\boldsymbol{y} = (y_1, \ldots, y_t)$, we define the following basic functions:

$$f_1(\boldsymbol{x}) = \|\boldsymbol{x}\|_1 = \sum_{j=1}^{t} x_j, \qquad f_2(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{j=1}^{t} x_j \cdot y_j,$$

$$f_3(\boldsymbol{x}, \boldsymbol{y}) = f_1(\boldsymbol{x}) \cdot f_1(\boldsymbol{y}) = \left(\sum_{i=1}^{t} x_i \right) \cdot \left(\sum_{j=1}^{t} y_j \right)$$

We now give a description that explains how to use our scheme in section 5 for the following functions:

Average: Authenticate $f_1(\boldsymbol{x})$, since

$$\text{avg}(\boldsymbol{x}) = \frac{\sum_{j=1}^{t} x_j}{t} = \frac{f_1(\boldsymbol{x})}{t}.$$

Variance: Authenticate $f_1(\boldsymbol{x})$ and $f_2(\boldsymbol{x}, \boldsymbol{x})$, since

$$\begin{aligned}\text{var}(\boldsymbol{x}) &= \frac{\sum_{j=1}^{t}(x_j - \text{avg}(\boldsymbol{x}))^2}{t} = \frac{\sum_{j=1}^{t} x_j^2}{t} - \frac{(\sum_{i=1}^{t} x_i)^2}{t^2} \\ &= \frac{f_2(\boldsymbol{x}, \boldsymbol{x})}{t} - \frac{f_1(\boldsymbol{x})^2}{t^2}\end{aligned}$$

Standard deviation: Since $\text{stdev}(\boldsymbol{x}) = \sqrt{\text{var}(\boldsymbol{x})}$, one can simply use the above method for authenticating the variance function.

Root Mean Square: Authenticate $f_2(\boldsymbol{x}, \boldsymbol{x})$, since

$$\text{RMS}(\boldsymbol{x}) = \sqrt{\frac{\sum_{j=1}^{t} x_j^2}{t}}$$

Covariance: Authenticate $f_2(\boldsymbol{x}, \boldsymbol{y})$ and $f_3(\boldsymbol{x}, \boldsymbol{y})$, since

$$\begin{aligned}\text{cov}(\boldsymbol{x}, \boldsymbol{y}) &= \frac{\sum_{j=1}^{t}(x_j - \text{avg}(\boldsymbol{x}))(y_j - \text{avg}(\boldsymbol{y}))}{t} \\ &= \frac{\sum_{j=1}^{t} x_j y_j}{t} - \frac{(\sum_{i=1}^{t} x_i)(\sum_{j=1}^{t} y_j)}{t^2} \\ &= \frac{f_2(\boldsymbol{x}, \boldsymbol{y})}{t} - \frac{f_3(\boldsymbol{x}, \boldsymbol{y})}{t^2}\end{aligned}$$

Linear Regression: Given two sets of observations as two vectors $(\boldsymbol{x}, \boldsymbol{y})$, the linear regression of $\boldsymbol{y}$ as a function of $\boldsymbol{x}$ is defined by two coefficients $\hat{\alpha}, \hat{\beta}$ such that

$$\begin{aligned}\hat{\beta} &= \frac{\sum_{i=1}^{t}(x_i - \text{avg}(\boldsymbol{x}))(y_i - \text{avg}(\boldsymbol{y}))}{\sum_{i=1}^{t}(x_i - \text{avg}(\boldsymbol{x}))^2} = \frac{\text{cov}(\boldsymbol{x}, \boldsymbol{y})}{\text{var}(\boldsymbol{x})} = \frac{f_2(\boldsymbol{x}, \boldsymbol{y}) - f_3(\boldsymbol{x}, \boldsymbol{y})/t}{f_2(\boldsymbol{x}, \boldsymbol{x}) - f_1(\boldsymbol{x})/t} \\ \hat{\alpha} &= \text{avg}(\boldsymbol{y}) - \hat{\beta} \cdot \text{avg}(\boldsymbol{x}) = (f_1(\boldsymbol{y}) - \hat{\beta} f_1(\boldsymbol{x}))/t\end{aligned}$$

Hence, authenticate $f_1(\boldsymbol{x})$, $f_1(\boldsymbol{y})$, $f_2(\boldsymbol{x}, \boldsymbol{y})$, $f_2(\boldsymbol{x}, \boldsymbol{x})$, and $f_3(\boldsymbol{x}, \boldsymbol{y})$.

Sample Pearson's correlation coefficient: Authenticate $f_1(\boldsymbol{x})$, $f_1(\boldsymbol{y})$, $f_2(\boldsymbol{x}, \boldsymbol{y})$, $f_2(\boldsymbol{x}, \boldsymbol{x})$, $f_2(\boldsymbol{y}, \boldsymbol{y})$, $f_3(\boldsymbol{x}, \boldsymbol{y})$, since

$$\begin{aligned}r_{\boldsymbol{x},\boldsymbol{y}} &= \frac{\sum_{i=1}^{t}(x_i - \text{avg}(\boldsymbol{x}))(y_i - \text{avg}(\boldsymbol{y}))}{\sqrt{\sum_{i=1}^{t}(x_i - \text{avg}(\boldsymbol{x}))^2} \cdot \sqrt{\sum_{i=1}^{t}(y_i - \text{avg}(\boldsymbol{y}))^2}} = \frac{\text{cov}(\boldsymbol{x}, \boldsymbol{y})}{\text{stdev}(\boldsymbol{x})\text{stdev}(\boldsymbol{y})} \\ &= \frac{f_2(\boldsymbol{x}, \boldsymbol{y}) - f_3(\boldsymbol{x}, \boldsymbol{y})/t}{t\sqrt{(f_2(\boldsymbol{x}, \boldsymbol{x})/t - f_1(\boldsymbol{x})^2/t^2)(f_2(\boldsymbol{y}, \boldsymbol{y})/t - f_1(\boldsymbol{y})^2/t^2)}} \\ &= \frac{f_2(\boldsymbol{x}, \boldsymbol{y}) - f_3(\boldsymbol{x}, \boldsymbol{y})/t}{\sqrt{f_2(\boldsymbol{x}, \boldsymbol{x}) f_2(\boldsymbol{y}, \boldsymbol{y}) - f_2(\boldsymbol{x}, \boldsymbol{x}) f_1(\boldsymbol{y})^2/t - f_1(\boldsymbol{x})^2 f_2(\boldsymbol{y}, \boldsymbol{y})/t + f_1(\boldsymbol{x})^2 f_1(\boldsymbol{y})^2/t^2)}}\end{aligned}$$

Uncentered correlation coefficient: Authenticate $f_2(\boldsymbol{x}, \boldsymbol{y})$, $f_2(\boldsymbol{x}, \boldsymbol{x})$, $f_2(\boldsymbol{y}, \boldsymbol{y})$, since

$$\begin{aligned}r^u_{\boldsymbol{x},\boldsymbol{y}} &= \frac{\sum_{i=1}^{t} x_i y_i}{\sqrt{\sum_{i=1}^{t} x_i^2} \cdot \sqrt{\sum_{i=1}^{t} y_i^2}} \\ &= \frac{f_2(\boldsymbol{x}, \boldsymbol{y})}{\sqrt{f_2(\boldsymbol{x}, \boldsymbol{x}) f_2(\boldsymbol{y}, \boldsymbol{y})}}\end{aligned}$$

9.2 Distance and Correlation Measures on Encrypted Data Sets

Consider the problem in which a client stores a large matrix $X \in \mathbb{F}_p^{L \times N}$ on a server, and then wants to compute the Euclidean distance between a given vector $\boldsymbol{y} \in \mathbb{F}_p^N$ and each row of the matrix. Namely, on input $\boldsymbol{y}$ from the client, the server computes a vector $\boldsymbol{D} = (D_1, \ldots, D_L)$ where every entry $D_i = \sum_{j=1}^{N}(X_{i,j} - y_j)^2$ is the (square of the) Euclidean distance[3] between $\boldsymbol{y}$ and the i-th row of X. For security, we are interested in a solution that guarantees both integrity and privacy, i.e., results are efficiently verifiable by the clients without having to store the matrix X, and the server does not learn any information about neither X nor the queried vector $\boldsymbol{y}$.

To achieve a solution for this application we can employ the function-private scheme $\mathcal{VC}^*_{\mathsf{quad}}$ for degree-2 polynomials (by explicitly relying on its split version) as follows:

- First, the client generates the secret key for the input-encoding $(PK_E, SK_E) \xleftarrow{\$} \mathbf{KeyGen}^E(\lambda)$.
- In a pre-processing phase, given the matrix X, the client computes $\sigma_{\boldsymbol{x},i} \xleftarrow{\$} \mathbf{ProbGen}(SK_E, \boldsymbol{X}_i)$ for all the rows of X. Precisely, we can use batching to encode s rows in the same $\sigma_{\boldsymbol{x},i}$. The cost of this phase, which is performed only once is $O(L \cdot N)$.
- Given the vector $\boldsymbol{y}$, the client defines the function $f_y(z_1, \ldots, z_N) = \sum_{j=1}^{N}(z_j - y_j)^2$, and runs $(EK_{f_y}, SK_{f_y}) \xleftarrow{\$} \mathbf{KeyGen}^V(f_y, \lambda, PK_E, SK_E)$. Note that f_y is an admissible function for $\mathcal{VC}^*_{\mathsf{quad}}$ as it is of degree-2 and all the constants derived from the y_j's multiply degree-1 terms. Moreover, notice that the client can compactly send f_y to the server by sending $\hat{\mathsf{H}}(y_i)$. The cost of this phase is $O(N)$.
- The server computes $\sigma_{D_i} \leftarrow \mathbf{Compute}(EK_E, EK_{f_y}, \sigma_{\boldsymbol{x},i})$ for every (packed) row of the encrypted matrix, and returns $(\sigma_{D_1}, \ldots, \sigma_{D_L})$ to the client.
- Finally, the client obtains the verified result by running the verification algorithm on every output encoding σ_{D_i}. The cost of verifying each entry is $O(1)$ which sums up to $O(L)$.

To summarize, after the pre computation to outsource the matrix, the work performed by the client to send $\boldsymbol{y}$ and verify the result is $O(N + L)$, which outperforms the cost of running this computation, which is $O(L \cdot N)$. Note also that the pre computation cost can be amortized when asking many queries $\boldsymbol{y}$ for the same matrix X. For privacy note that by input privacy and function privacy the server does not learn information about the matrix X and the vector $\boldsymbol{y}$ respectively.

While the protocol above is described for the Euclidean distance, it is easy to see that the same approach works also for other degree-2 functions with the same properties. Namely, for any $f(\boldsymbol{y}, X_i)$ such that by fixing $\boldsymbol{y}$, f is of degree-2 and multiplications by y_i involve only degree-1 monomials. For instance, this property holds when computing the covariance or the correlations coefficients between $\boldsymbol{y}$ and every row of a matrix X.

9.3 Discrete Fourier Transform

The discrete Fourier transform (DFT) of a t-dimensional vector $\boldsymbol{f}$ is defined as the vector $\boldsymbol{y} = (f(\alpha_1), \ldots, f(\alpha_t))$ where $\boldsymbol{f}$ is interpreted as the coefficients vector of a polynomial of degree $(t-1)$, and the α_i's are the t roots of unity. It is easy to see that by using our scheme for univariate polynomials of section 6, a client can store encrypted vectors on a server and then request the DFT transform of these vectors in a private and verifiable way. In particular, note that the delegation and verification cost is optimal: $O(t)$ (i.e., $O(1)$ for every entry of $\boldsymbol{y}$).

[3] For simplicity, we assume that the final square root can be directly computed by the client.

10 Experimental Evaluation

To give the reader a glance at the practical applicability of our procedures, we implemented the schemes in sections 5, 6, and 7, and reported the cost of each procedure.

10.1 Setup

Hardware and Software. Our timings were performed on a 2011 MacBook Pro (Intel Core i5-2415M, 2 hyperthreaded cores at 2.30GHz, 8GB RAM at 1.333GHz), running Ubuntu (linux kernel 3.11, SMP, x86_64). All our implementations are single-threaded.

For our code, we made use of the following libraries:

- HElib [38], which implements the BGV scheme [18]. We tweaked some homomorphic operations to avoid KeySwitch and ModulusSwitch.
- NTL [52], to perform operations over polynomial rings (e.g. evaluating $\mathsf{H}_{\alpha,\beta}$).
- PBC [45], to perform group and pairing operations (e.g. evaluating $e(X^{(1)}, Y^{(2)})$ in the scheme in section 5).

Input. Our schemes are targeted at the encryption of a database that can be thought of as a big table. Each *row* of the table represents an *input*, and each *column* represents an *attribute*. An example is given in table 3.

Day	Bicycles	Cars	Trucks	...
March 1	55	128	20	...
March 2	10	28	0	...
⋮	⋮	⋮	⋮	

Table 3. Number of vehicles driving through Paper Street at a given time.

The schemes we implemented all use the BGV encryption scheme [18], which, as pointed out in section 4.1 has the nice feature batching. Therefore, for the computation of the same function for each attribute (e.g. in table 3, the average amount of bicycles, cars, trucks, etc, going through Paper Street in March), it is natural to encrypt each entire *row* of the database into a single ciphertext, encoding each data set element into a different slot.

Parameters selection. In our implementations we covered 80bit and 128bit security, and we required the length of each data set item to be at most 32bit. These choices lead to the following parameters (s denotes the number of slots):

Polynomials (Section 6 and 5): For 80bit (respectively 128bit) security, we chose $\log q = 173$ ($\log q = 272$), $n = 5418$ ($n = 8820$), $s = 165$ ($s = 275$).

Linear Combinations (Section 7): For 80bit (respectively 128bit) security, we chose $\log q = 123$ ($\log q = 173$), $n = 4050$ ($n = 6370$), $s = 125$ ($s = 196$).

10.2 Timings

We are now ready to present our timings. We introduce the concept of PCost (a shorthand for "privacy cost") of a process, defined as the ratio of the total execution time of the process over the

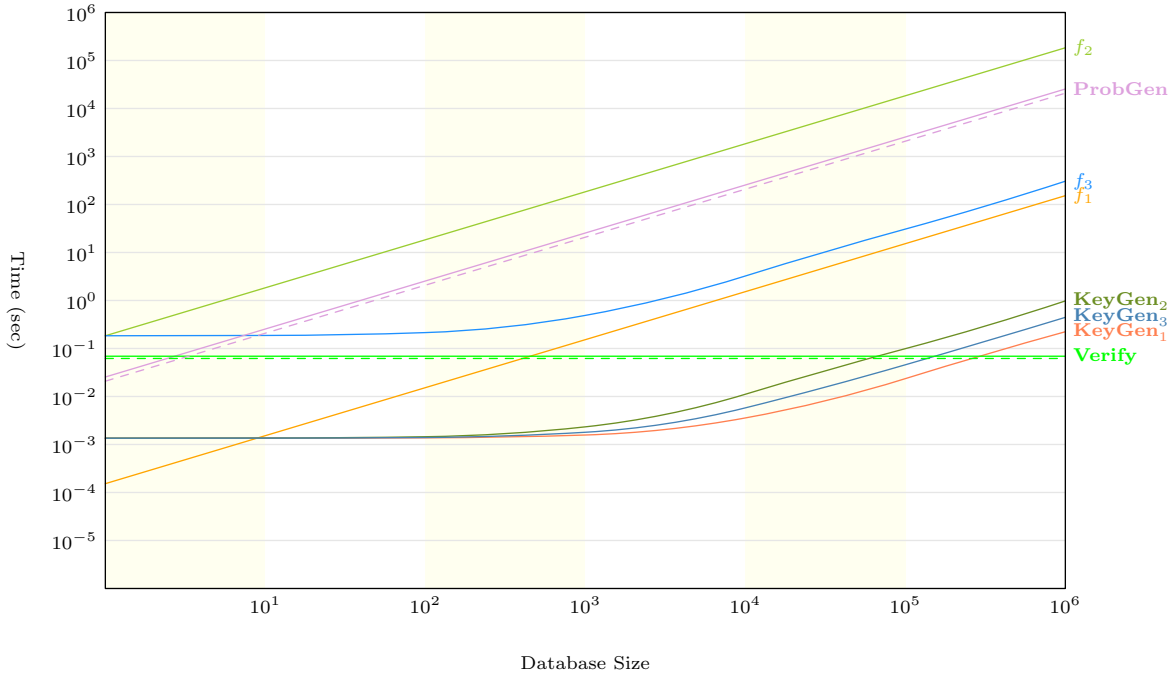
time taken by the process to compute authentication operations (i.e. excluding privacy operations such as encryption, FHE evaluation, hashing, etc). PCost can therefore be seen as a measure for the cost for bringing privacy to each operation of a VC scheme.

In the following, by "Amortized" cost, we mean the per-attribute cost, which equals the "Raw" (or total) cost of an operation divided by the number of attributes, according to the above parameters choice. Moreover, costs associated with procedures followed by "$*$" are obtained using exponentiations with precomputation.

Multi-Variate Quadratic Polynomials.

Timing (ms)	Raw Cost		Amortized Cost		PCost	
Security	80bit	128bit	80bit	128bit	80bit	128bit
ProbGen	$25d$	$75d$	$0.15d$	$0.27d$	2.92	3.09
ProbGen*	$21d$	$65d$	$0.13d$	$0.24d$	5.08	8.00
Add	0.15	0.28	0.0009	0.001	2.21	3.45
Const. Mult.	16	28	0.09	0.01	1.10	1.08
Mult.	184	369	1.11	1.32	1.14	1.33
Verify	70	180	0.42	0.65	1.92	2.77
Verify*	62	166	0.37	0.60	2.14	3.23

Table 4. Summary of the costs associated with each procedure (ms) for the scheme described in section 5. We omitted the cost of **KeyGen**, as that is dependent on the function to compute; see figure 1 for more information.



- f_1, f_2, and f_3 denote the cost of the corresponding function (see section 9.1 for more details). Notice that the cost of f_3 does not scale linearly with the database size (there is a constant cost of one multiplication, which dominates for small databases).
- KeyGen$_1$, KeyGen$_2$, and KeyGen$_3$ denote the cost of **KeyGen** for the corresponding function. Here there is a constant term (BGV.ParamGen) and a linear term that depends on the cost of the function.
- ProbGen, and Verify denote the cost of the corresponding function. Dashed lines depict costs obtained using precomputation. Notice that the cost of **Verify** is constant.

Fig. 1. Summary of the raw costs for the scheme in section 5, with 80bit security.

Univariate Polynomials of Large Degree.

Timing (ms)	**Raw Cost**		**Amortized Cost**		**PCost**	
Security	80bit	128bit	80bit	128bit	80bit	128bit
KeyGen	$1.35 + 20.7d$	$4.33 + 65d$	$0.008 + 0.125d$	$0.015 + 0.234d$	5.29	8.26
KeyGen*	$1.35 + 19.7d$	$4.33 + 63d$	$0.008 + 0.124d$	$0.015 + 0.233d$	5.95	9.62
ProbGen	N/A	N/A	N/A	N/A	N/A	N/A
Compute	$2d$	$4d$	$0.01d$	$0.01d$	3.34	2.65
Verify	17	53	0.11	0.19	3.89	5.78
Verify*	16	51	0.10	0.18	4.76	7.46

Table 5. Summary of the costs associated with each procedure (ms) for the scheme described in section 6.

Linear Combinations.

Timing (ms)	**Raw Cost**		**Amortized Cost**		**PCost**	
Security	80bit	128bit	80bit	128bit	80bit	128bit
KeyGen	$0.5 + 9.6d$	$1.4 + 21.7d$	$0.004 + 0.077d$	$0.007 + 0.111d$	5.35	5.18
ProbGen	$10d$	$22d$	$0.08d$	$0.11d$	5.35	5.18
Compute	$9d$	$24d$	$0.07d$	$0.12d$	10.53	10.87
Verify	18	44	0.14	0.23	4.41	4.48

Table 6. Summary of the costs associated with each procedure (ms) for the scheme described in section 7.

On the Impact of Homomorphic Hashing. Our experiments showed the improvement obtained by applying our technique of homomorphic hashing. We compared our ad-hoc protocols with some of the best possible instantiations of our generic scheme and observed a remarkable speedup. For instance, for the case of multivariate quadratic polynomials, one may use BGV to encrypt and then use a homomorphic MAC [6] to authenticate each of the $2n$ ciphertext entries. However, this would further require to authenticate bigger circuits: for example, one has to validate $2n$ additions over $\mathbb{Z}/q\mathbb{Z}$ for the addition of two ciphertexts, and at least $4(n \log n)$ multiplications and $5n$ additions over $\mathbb{Z}/q\mathbb{Z}$ for a multiplication of ciphertexts (estimate obtained assuming FFT multiplication; using a trivial method would lead to $4n^2$ multiplications and $5n$ additions). In contrast, by applying our homomorphic hash we can use the same original circuit and the additional cost of computing the hash becomes negligible.

In tables 7 and 8 we give a list of ratios between the cost of the alternate approach (tagging without hashing) versus our approach (hashing and then tagging) in the schemes described in sections 6 and 5 respectively.

	KeyGen	**Compute**	**Verify**
80bit Security	304	3223	762
128bit Security	368	6641	951

Table 7. Ratios between the costs of tagging each single ciphertext entry versus "hashing and tagging" for the scheme in section 6.

	ProbGen	Add1	Add2	Const. Mult.	Mult. (FFT)	Mult. (Trivial)	**Verify**
80bit Security	774	2453	3680	4943	13632	$73 \cdot 10^6$	7929
128bit Security	935	2555	3832	8144	18974	$167 \cdot 10^6$	8519

Table 8. Ratios between the costs of tagging each single ciphertext entry versus "hashing and tagging" for the scheme in section 5. Add1 represents the overhead of adding before multiplying, while Add2 represents the overhead of adding after multiplying.

Acknowledgments

The research of Rosario Gennaro was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defense 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.

The research of Valerio Pastro was supported by NSF Grant No.1017660

References

1. B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, pages 152–163, 2010.
2. S. Arora and S. Safra. Probabilistic checking of proofs; a new characterization of np. In *FOCS*, pages 2–13. IEEE Computer Society, 1992.
3. L. Babai. Trading group theory for randomness. In R. Sedgewick, editor, *STOC*, pages 421–429. ACM, 1985.
4. L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In C. Koutsougeras and J. S. Vitter, editors, *STOC*, pages 21–31. ACM, 1991.
5. L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. In *FOCS*, pages 16–25. IEEE Computer Society, 1990.
6. M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In *2013 ACM Conference on Computer and Communication Security*. ACM Press, November 2013.
7. M. Bellare, V. T. Hoang, and S. Keelveedhi. Instantiating random oracles via uces. In Canetti and Garay [20], pages 398–415.
8. M. Bellare, V. T. Hoang, and P. Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In X. Wang and K. Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 134–153. Springer, 2012.
9. M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *ACM Conference on Computer and Communications Security*, pages 784–796, 2012.
10. E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *CRYPTO*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.
11. S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In Rogaway [51], pages 111–131.

12. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Goldwasser [32], pages 326–349.
13. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. *IACR Cryptology ePrint Archive*, 2012. http://eprint.iacr.org/2012/095.
14. D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2005.
15. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004.
16. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, 2011.
17. Z. Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.
18. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In Goldwasser [32], pages 309–325.
19. Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In Rogaway [51], pages 505–524.
20. R. Canetti and J. A. Garay, editors. *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*. Springer, 2013.
21. D. Catalano, A. Marcedone, and O. Puglisi. Authenticating computation on groups: New homomorphic primitives and applications. Cryptology ePrint Archive, Report 2013/801, 2013. http://eprint.iacr.org/.
22. K.-M. Chung, Y. T. Kalai, and S. P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 483–501. Springer, 2010.
23. G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with streaming interactive proofs. In *Innovations in Theoretical Computer Science – ITCS*, pages 90–112. ACM, 2012.
24. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013.
25. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Rabin [49], pages 465–482.
26. R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.
27. R. Gennaro and D. Wichs. Fully homomorphic message authenticators. In *ASIACRYPT (2)*, volume 8270 of *Lecture Notes in Computer Science*, pages 301–320. Springer, 2013.
28. C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.
29. C. Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In Rabin [49], pages 116–137.
30. C. Gentry and S. Halevi. Implementing gentry's fully-homomorphic encryption scheme. In K. G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011.
31. C. Gentry, S. Halevi, and N. P. Smart. Fully homomorphic encryption with polylog overhead. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012.
32. S. Goldwasser, editor. *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*. ACM, 2012.
33. S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, pages 555–564, 2013.
34. S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In C. Dwork, editor, *STOC*, pages 113–122. ACM, 2008.
35. S. Goldwasser, H. Lin, and A. Rubinstein. Delegation of computation without rejection problem from designated verifier CS-proofs. *IACR Cryptology ePrint Archive*, 2011:456, 2011.
36. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.
37. J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321–340, 2010.
38. S. Halevi and V. Shoup. Helib. https://github.com/shaih/HElib. Accessed: 2014-05-14.
39. C. Joo and A. Yun. Homomorphic authenticated encryption secure against chosen-ciphertext attack. *IACR Cryptology ePrint Archive*, 2013:726, 2013.
40. M. Joye and B. Libert. Efficient cryptosystems from 2k-th power residue symbols. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 72–92. Springer, 2013.

41. J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In S. R. Kosaraju, M. Fellows, A. Wigderson, and J. A. Ellis, editors, *STOC*, pages 723–732. ACM, 1992.
42. J. Kilian. Improved efficient arguments (preliminary version). In D. Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 311–324. Springer, 1995.
43. B. Libert, T. Peters, M. Joye, and M. Yung. Linearly homomorphic structure-preserving signatures and their applications. In Canetti and Garay [20], pages 289–307.
44. H. Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 169–189. Springer, 2012.
45. B. Lynn. Pbc library. http://crypto.stanford.edu/pbc/. Accessed: 2014-05-14.
46. S. Micali. Cs proofs (extended abstracts). In *FOCS*, pages 436–453. IEEE Computer Society, 1994.
47. B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society, 2013.
48. B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer, 2012.
49. T. Rabin, editor. *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*. Springer, 2010.
50. R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. pages 169–177. Academic Press, 1978.
51. P. Rogaway, editor. *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011.
52. V. Shoup. Ntl library. http://www.shoup.net/ntl/. Accessed: 2014-05-14.
53. N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In P. Q. Nguyen and D. Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.
54. N. P. Smart and F. Vercauteren. Fully homomorphic simd operations. *IACR Cryptology ePrint Archive*, 2011:133, 2011.
55. J. Thaler. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2013.