Verifiable Delegation of Computation on Outsourced Data*

Michael Backes[1,2], Dario Fiore[1], and Raphael M. Reischuk[2]

[1] Max Planck Institute for Software Systems (MPI-SWS)
Saarbrücken, Germany
{backes,fiore}@mpi-sws.org
[2] Saarland University
Saarbrücken, Germany
reischuk@cs.uni-saarland.de

Abstract. We address the problem in which a client stores a large amount of data with an untrusted server in such a way that, at any moment, the client can ask the server to compute a function on some portion of its outsourced data. In this scenario, the client must be able to efficiently verify the correctness of the result despite no longer knowing the inputs of the delegated computation, it must be able to keep adding elements to its remote storage, and it does not have to fix in advance (i.e., at data outsourcing time) the functions that it will delegate. Even more ambitiously, clients should be able to verify in time independent of the input-size – a very appealing property for computations over huge amounts of data.

In this work we propose novel cryptographic techniques that solve the above problem for the class of computations of quadratic polynomials over a large number of variables. This class covers a wide range of significant arithmetic computations – notably, many important statistics. To confirm the efficiency of our solution, we show encouraging performance results, e.g., correctness proofs have size below 1 kB and are verifiable by clients in less than 10 milliseconds.

Keywords: Verifiable Delegation of Computation; CloudComputing; Secure Data Outsourcing; Homomorphic MACs; Amortized Closed-Form Efficient PRF

* An extended abstract of this work appears in the proceedings of ACM CCS 2013 [9]. This if the full version.

Table of Contents

Verifiable Delegation of Computation on Outsourced Data 1
Michael Backes, Dario Fiore, and Raphael M. Reischuk
1 Introduction 3
1.1 Related Work 5
1.2 A High-Level Overview of Our Techniques 7
1.3 Organization of the Paper 9
2 Preliminaries 9
3 Homomorphic Message Authenticators with Efficient Verification 10
3.1 Multi-Labeled Programs 11
3.2 Homomorphic MACs for Multi-Labeled Programs 12
3.3 Homomorphic MACs with Efficient Verification for Multi-Labeled Programs 14
4 Utilities 16
4.1 Homomorphic Evaluation of Arithmetic Circuits 16
4.2 Pseudorandom Functions with Amortized Closed-Form Efficiency 18
4.3 A PRF with Amortized Closed-Form Efficiency for GroupEval 19
5 Homomorphic Message Authenticators with Efficient Verification 23
5.1 Construction 23
5.2 Proof of Correctness 25
5.3 Proof of Security 27
5.4 Efficiency Analysis 31

1 Introduction

Given the emergence of cloud computing (an infrastructure where clients or businesses lease computing and storage resources from powerful service providers), it is of critical importance to provide integrity guarantees for outsourced data management. Consider for example the following scenario. A client has a collection of a large (potentially unbounded) amount of data $D = D_1, D_2, D_3, \ldots$, for instance, environmental data such as air pollution levels at fixed time intervals (e.g., every hour), and it may wish to compute statistics on such data. If the client's memory is not large enough to store the entire data, the client might consider relying on a cloud service and storing the data on a remote server $\mathcal{S}$. Other significant examples of this scenario include arbitrary files at remote storage systems, as well as endless data streams such as financial data (e.g., price fixing data from the stock markets, financial figures and revenues of companies), experimental data (e.g., genetic data, laboratory measurements), and further environmental data (e.g., surface weather observations). In this scenario, we hence have a client who incrementally sends D to a server $\mathcal{S}$, the server stores D, and at certain points in time the client asks $\mathcal{S}$ to compute a function on (a portion of) the currently outsourced data. We stress that the data D and its size cannot be fixed in advance as the client may need to add additional data to the outsourced storage. Analogously, the client might not know in advance what functions it will apply on the outsourced data (e.g., it may wish to compute several statistics).

However, if the server is *untrusted* (i.e., it is malicious or becomes prey to an external attack), how can the client verify that the results provided by the server are correct? This question naturally leads to two important requirements: (1) *security*, meaning that the server should be able to "prove" the correctness of the delegated computation for some function f; and (2) *efficiency*, meaning that the client should be able to check the proof by requiring significantly fewer resources than those that are needed to compute f (including both computation and communication). Furthermore, if we consider computations over very large sets of inputs (e.g., statistics on huge data sets), we want to be more ambitious and envision the achievement of (3) *input-independent efficiency*, meaning that verifying the correctness of a computation $f(D_1, \ldots, D_n)$ requires time *independent* of n. Moreover, two further requirements are crucial in this setting: (4) *unbounded storage*, meaning that the size of the outsourced data should not be fixed a-priori, i.e., clients should be able to outsource any (possibly growing) amount of data; and (5) *function-independence*, meaning that a client should be able to outsource its data without having to know in advance the functions that it will delegate later.

Relation with Verifiable Computation. The problem of securely and efficiently outsourcing the computation of a function f to a remote server has been the subject of many works in the so-called field of *verifiable computation*. Most of these works achieve the goals of security (1) and efficiency (2), but they inevitably fail in achieving requirements (3)–(5). Roughly speaking, the issue is that most existing work requires the client to know (i.e., to store a local copy of) the input D for the verification of the delegated function (e.g., in SNARG-based approaches [12,28] and in signatures of correct computation [43]), or, otherwise, to send D to the server *all at once* (rather than sending it over time) and to keep a small local state which would *not* allow to append additional data at a later time (e.g., in [47,25]). Perhaps more critically, many of the existing solutions in this area require the delegator to run in time proportional to the input size n of the delegated function, e.g., in time $\mathsf{poly}(n)$. In the various existing protocols, these limitations arise for different reasons (see Section 1.1 for a more detailed discussion). However, even if verification in these works is more

efficient than running f, we think that, for computations over huge data sets, a $\mathsf{poly}(n)$ overhead is still unacceptably high.

The only approach that comes close to achieving requirements (1)–(5) is the work by Chung et al. on memory delegation [22]. The authors propose a scheme based on techniques from [32] which exploit the power of the PCP theorem [8,24,3,4]. With this scheme, a client can delegate a broad class of computations over its outsourced memory fulfilling the requirements from above (except for verification efficiency, which requires time $\log n$, instead of constant time). While providing a satisfying solution in theory, this approach suffers from the usual impracticality issues of general-purpose PCP techniques and hence does not lead to truly practical solutions to the problem.

Our Contribution. In this work, we address the problem of verifiable delegation of computations on (growing) outsourced data. Our main contribution is the first *practical* protocol that achieves all five of the requirements stated before. Namely, a client can (continuously) store a large amount of data $D = D_1, D_2, D_3 \ldots$ with the server, and then, at certain points in time, it can request the computation of a function f on (a portion of) the outsourced data, e.g., $v = f(D_{i_1}, \ldots, D_{i_n})$. Using our protocol, the server sends to the client a short piece of information vouching for the correctness of v. The protocol achieves input-independent efficiency in the amortized model: after a single precomputation with cost $|f|$, the client can verify *every* subsequent evaluation of f in *constant* time, i.e., regardless of the input size n. Moreover, fulfilling properties (4)–(5), we have that data outsourcing and function delegation are completely *decoupled*, i.e., the client can *continuously* add elements to the remote storage, and the delegated functions do *not* have to be fixed a priori. This means that the cost of outsourcing the data can be, in fact, excluded from the delegation; think for instance of incrementally outsourcing a large data stream during an entire year, and then computing statistics on the data at the end of the year.

Our solution works for computations over integers in the ring $\mathbb{Z}_p$ (where p is a large prime of roughly 2λ bits, for a security parameter λ), and supports the evaluation of arithmetic circuits of degree up to 2. This restricted class of computations is enough to capture a wide range of significant arithmetic computations, such as meaningful statistics, including counting, summation, (weighted) average, arithmetic mean, standard deviation, variance, covariance, weighted variance with constant weights, quadratic mean (aka root-mean square – RMS), mean squared error (MSE), the Pearson product-moment correlation coefficient, the coefficient of determination (R^2), and the least squares fit of a data set $\{(x_i, v_i)\}_{i=1}^{n}$ (in the case when the x_i are universal constants, e.g., days of the year)[3].

Our key technical contribution is the introduction of *homomorphic MACs with efficient verification*. This cryptographic primitive extends homomorphic message authenticators [30] by adding a crucial efficiency property for the verification algorithm. We propose a first realization of homomorphic MACs with efficient verification (see Section 1.2 for an overview of our techniques), and we prove its security under the Decision Linear assumption [13]. Using the above construction we build an efficient protocol that can be implemented using bilinear pairings.

To demonstrate the practicality of our solution, we evaluate the concrete operations that have to be performed by the client and the server, as well as the bandwidth overhead introduced by the protocol for transferring the proofs. If we consider 80 bits of security and an implementation of symmetric pairings [38] on a standard desktop machine, we observe the following costs (see Table 2 for the 128-bit case): For outsourcing a data item D_i, the client needs to perform a single modular exponentiation in 0.24ms. This operation yields a very short authentication tag of size 0.08kB,

[3] The least squares fit for this case can indeed be computed using a linear function [15].

which is sent to the server along with D_i. For the verification of a computation result v, the client receives a proof σ_v of size 0.21kB from the server, and can check this proof by computing one pairing and one multi-exponentiation in 1.06ms.

As we mentioned before, we achieve input-independent efficiency in an amortized sense. So, the above verification costs are obtained after the precomputation of some concise information ω_f related to the delegated function f. Precomputing ω_f takes the same time as computing f (with almost no additional overhead!), it does not require knowledge of the input data, and ω_f can be re-used an unbounded number of times to verify several evaluations of f on many different outsourced data sets. To generate the proof σ_v related to the evaluation of a function f, the server has to run f with an additional, yet constant, overhead – derived from replacing additions in f with a group operation, and replacing multiplications with a pairing. Although our solution can still not capture general-purpose computations, the above performance evaluation shows that for our case of interest we achieve results that are encouraging for a practical deployment of this protocol.

In summary, in this work we focus on the important problem of delegating computations over data which continuously grows and is outsourced to remote servers. This specific problem has not received much attention so far: the only existing solution [22], though very general, does not seem to lead to efficient implementations. In contrast, we propose a protocol that achieves all the desired requirements for a restricted, yet practical and useful, class of computations, and has the advantage of achieving performances that are promising for a practically efficient solution.

1.1 Related Work

Memory Delegation. The work of Chung et al. [22] on memory delegation and streaming delegation is the closest one to the model considered in our work. Indeed, Chung et al. consider the problem in which a client outsources a large amount of data to a server and it later delegates computations over the data, while maintaining the ability to verify the results in time independent of the input size. In memory delegation the client uploads his memory to the server (in an offline phase), and it can later ask the server to update the outsourced memory and to compute a function f on its entire memory (in an online phase). In streaming delegation the memory can be updated only by appending elements. The main advantages of the work of Chung et al. over our results are that: (i) the client can change values in the outsourced memory, (ii) they provide solutions for more expressive computations (i.e., a 4-round protocol for arbitrary poly-time programs). However, their solutions also suffer some disadvantages. First, they require the client to be stateful (in our solution the client keeps only a *fixed* secret key). Second, in streaming delegation, the size N of the stream has to be a-priori bounded. Such a bound also affects the client's memory since it requires a local storage size of approximately $\log N$ at the client, meaning that N cannot be chosen arbitrarily long, and thus the stream cannot be endless. Also, in their solutions, the client still runs in time $\mathsf{polylog}(n)$ in the online phase, where n is the size of the entire memory. In contrast, our solution supports unbounded data streams, and allows for clients that (after a preprocessing phase which is input-independent) can verify computations in constant time.

Authenticated Data Structures. A line of research which addresses a problem closely related to the one considered in this paper is the existing work on *authenticated data structures* [42,53]. This area considers a setting in which clients want to securely delegate certain operations on data structures that are stored at untrusted remote servers. Existing work addresses both static settings and dynamic settings (where data structures can be updated), and it mostly focuses on specific data

structure operations, such as range search queries over databases [34,39], authenticated dictionaries [23,44,33], and set operations (e.g., intersection, union, set difference) over a dynamic collection of sets [45]. However, none of the works in this area considers the secure outsourcing of arbitrary or arithmetic computations (e.g., statistics) over remotely stored data.

Multi-Function Verifiable Computation. The notion of *multi-function verifiable computation* proposed by Parno, Raykova, and Vaikuntanathan [47] is close to our model, in that a client can delegate the computation of many functions $f_1, f_2, \ldots$ on the same input D, while being able to efficiently verify the results. Even though multi-function verifiable computation does not require the client to fix the function f before outsourcing the data, this model still falls short of our requirements. The main problem is that in multi-function verifiable computation, the client has to store some information τ_D for every input D on which it will ask to compute a function $f_i(D)$. Furthermore, there is no possibility of updating τ_D without locally storing the previous data. This essentially means that the data D has to be sent all at once, thus ruling out all applications in the growing data scenario.

Homomorphic Signatures and MACs. The problem of realizing homomorphic message authentication schemes in both the symmetric setting (MACs) and in the asymmetric setting (signatures) has been considered by many prior works. Homomorphic signatures were first proposed by Johnson et al. [35]. However, since then, most works focus solely on *linear functions*, mainly because of the important application to network coding [14]. Several efficient schemes for linear functions have been proposed both in the random oracle model [14,29,16,18] and in the standard model [1,5,19,20,26,6,7]. Three more recent works consider the case of larger classes of functions [15,30,17]. Boneh and Freeman [15] proposed a realization of homomorphic signatures for bounded constant degree polynomials. Gennaro and Wichs [30] introduced homomorphic MACs and gave a construction for arbitrary computations which is based on fully homomorphic encryption and is proven secure in a weaker model where the adversary cannot ask verification queries. Catalano and Fiore [17] proposed realizations of homomorphic MACs that, despite capturing a restricted class of computations (i.e., arithmetic circuits with polynomially-bounded degree), support verification queries and are more efficient than previous works.

However, virtually all of the above works suffer the problem of having a verification algorithm which runs in time proportional to the function. Gennaro and Wichs [30] discuss the possibility of verifying a MAC in time better than executing the function, and propose some general solutions for their scheme which are based on fully homomorphic encryption and SNARGs [40]. However, neither the proposed solutions nor the suggested techniques yield schemes that achieve input-independent efficiency, and they do not seem to lead to practically efficient solutions, at least not as practical as required in this work.

Succinct Non-Interactive Arguments of Knowledge (SNARKs). A solution for realizing fully homomorphic signatures would be to use succinct non-interactive arguments of knowledge (SNARKs) [12]. For a given NP statement, this primitive allows for producing a succinct argument for proving knowledge of the corresponding witness. The main advantage of SNARKs is the succinctness of the argument (i.e., its size is independent of the size of both the NP statement and its witness), which can thus be verified efficiently. However, SNARKs are not as practically efficient as we might wish, and require either the random oracle model [40] or non standard, non-falsifiable, assumptions [31].

VERIFIABLE COMPUTATION. As we mentioned earlier, the problem considered by our work and addressed via homomorphic authenticators is related to the notion of verifiable computation for which there exits a vast literature, ranging from works for arbitrary computations [36,40,32,27,21,2,47,28,46] to works for specific classes of computations [11,25,43,18]. In verifiable computation, a client wants to delegate a computationally heavy task to a remote server while being able to verify the result in a very efficient way. As we mentioned before, most of these works suffer several limitations that do not make them appropriate for the model considered in this paper. For example, many existing solutions require the delegator to run in time proportional to the input size of the delegated function. This limitation arises for different reasons. For instance, in the definition proposed by Gennaro, Gentry, and Parno [27] (and later adopted in several works, e.g., [21,11,47,25]), to delegate the computation of $f(D)$, the client has to compute an encoding $\tau_{D,f}$ of D, which *depends* on the function f. However, if we want to choose f after outsourcing D, the computation of $\tau_{D,f}$ is no longer possible. Alternatively, one could keep the entire input D locally and then compute $\tau_{D,f}$ from D and f, which would yield a running time proportional to the input size. In other work (e.g., [36,40,32]) the efficiency requirement for a client is to run in time $\mathsf{poly}(n, \log T)$, when delegating a function f that runs in time T and takes inputs of size n.

Furthermore, as observed by Gennaro and Wichs [30], even if it is possible to reinterpret some of the results on verifiable computation in the setting of homomorphic message authenticators, the resulting solutions are still not appropriate. In particular, they might require a client to send the data all at once and would not allow for composition of several authenticated computations. We refer the reader to [30] for a thorough discussion about this.

Another interesting line of work in this area recently proposed efficient systems for verifiable computation [50,49,48,54]. The proposed solutions also work in a model where the client needs to know the input of the computation, and it also has to engage in an interactive protocol with the server in order to verify the results. In contrast, our work considers a completely non-interactive setting in which the proof is transferred from the server to the client in a single round of communication. In the past there have been proposals of practical solutions, but of limited provable security: e.g., solutions based on audit (e.g., [41,10]) or secure co-processors (e.g., [52,55]) which prove the computation as correct, under the assumption that the adversary cannot tamper with the processor. Compared to these results, our work relies only on standard cryptographic assumptions, and does not require any trusted hardware.

1.2 A High-Level Overview of Our Techniques

In what follows we give a high level overview of our construction and the techniques used therein. To obtain our solution we build on the notion of *homomorphic message authenticators* proposed by Gennaro and Wichs [30], a primitive which can be considered the secret-key equivalent of homomorphic signatures [15]. The basic idea of homomorphic MACs is that a user can use a secret key to generate a set of tags $\sigma_1, \ldots, \sigma_n$ authenticating values $D_1, \ldots, D_n$ respectively. Then, *anyone* can homomorphically execute a function f over $(\sigma_1, \ldots, \sigma_n)$ to generate a *short* tag σ that authenticates D as the output of $f(D_1, \ldots, D_n)$. At first glance, homomorphic MACs seem to perfectly fit the problem of verifiable computations on (growing) outsourced data. However, a closer look at this primitive reveals that this idea lacks the very important property of efficient verification. As discussed in Section 1.1, the issue is that in all existing constructions the verification algorithm of homomorphic MACs runs in time proportional to the description of the function. Our

key contribution is therefore to solve this efficiency issue by proposing a definition and a *first* practical realization of *homomorphic MACs with efficient verification*.

The starting point for the design of our construction is the homomorphic MAC scheme of Catalano and Fiore [17]: to authenticate a value $m \in \mathbb{Z}_p$, one "encodes" m into a degree-1 polynomial $y \in \mathbb{Z}_p[x]$ such that $y(0) = m$ and $y(\alpha) = \mathsf{F}_K(\mathsf{L})$. Here $\alpha \in \mathbb{Z}_p$ is a secret value randomly chosen by the client, and $\mathsf{F}_K(\cdot)$ is a pseudorandom function that is used to "randomize" a label L. One can think of a label as arbitrary information (e.g., a string) chosen by the client to describe the meaning of the authenticated value m (e.g., "air pollution on 2013/08/14 at 9:06:12"). Given a set of n authentication polynomials $y_1, \ldots, y_n$, the server creates a new MAC y which authenticates (i.e., it proves) that m is the result of $f(m_1, \ldots, m_n)$, e.g., f could be the variance of pollution levels at all time instants within a specific day/year etc. More specifically, the basic idea in [17] is to compute y by homomorphically executing the function f on the corresponding authentication polynomials, i.e., $y = f(y_1, \ldots, y_n)$. By the design of the y_i, this computation satisfies $y(0) = f(m_1, \ldots, m_n)$ and also $y(\alpha) = f(\mathsf{F}_K(\mathsf{L}_1), \ldots, \mathsf{F}_K(\mathsf{L}_n))$. Hence, the client can test whether a value m' (proposed by the server) is indeed the result of a computation $f(m_1, \ldots, m_n)$ by checking whether the MAC y provided by the server verifies the two conditions: (i) $y(0) = m'$ and (ii) $y(\alpha) = f(\mathsf{F}_K(\mathsf{L}_1), \ldots, \mathsf{F}_K(\mathsf{L}_n))$.

However, the Catalano-Fiore homomorphic MAC cannot be adopted in our setting: verifying a MAC for a function f requires the client to compute $W = f(\mathsf{F}_K(\mathsf{L}_1), \ldots, \mathsf{F}_K(\mathsf{L}_n))$ to perform check (ii), but this clearly takes the same time T as that for computing f — exactly what we want to avoid! One may then hope that once this value W is computed, it could be re-used, e.g., to verify other computations involving f. Unfortunately, this would require the re-use of labels, which is not possible at all: it is forbidden by the security definition used in [17]. More critically, the security of the Catalano-Fiore MAC completely breaks down in the presence of label re-use!

In our work, we solve this critical issue with two main ideas. Very informally, we first elaborate a model that allows us to partially, but *safely*, re-use labels. Then, we introduce the construction of a pseudorandom function which allows us to precompute a piece of label-independent information ω_f, such that ω_f can be re-used to compute W very efficiently (when the labels L_i are known).

To allow for a meaningful re-use of labels, we split labels in two dimensions, thus elaborating a model of *multi-labels*. A multi-label L consists of two components (Δ, τ) where Δ is the *data set identifier* and τ is the *input identifier*. A data set identifier could for instance be "air pollution on 2013/08/14"; and an input identifier could be used to identify a time, e.g., 9:06:12 am. For the example of the stock market data, the values could be the stock market prices for a company C at different times T. Then, the data set identifier could be the name of C while the input identifier could be the date and time T of the stock market price. The data set identifier is essentially a way of grouping together homogeneous data (e.g., data of the same population over which one wants to compute significant statistics) in such a way that one can compute within a data set Δ.

While a multi-label $\mathsf{L} = (\Delta, \tau)$ can still *not* be re-used to authenticate different messages, this model does allow us to assign the *same* input identifiers τ to as many messages as we need, as long as such messages lie in different data sets. In any case, a re-use of a complete multi-label for authentication purposes would not make much sense, as multi-labels are used by clients to "remember" and categorize the outsourced data. This transition from labels to multi-labels is natural: think again of the air pollution levels for a specific day. The input identifiers capture the hours of a day. Hence, the input identifiers might be re-used for other days, but the combination of date and time would never be re-used.

The use of multi-labels, however, does not in itself solve the issue of the inefficient verification algorithm: in this case one still has to compute $W = f(\mathsf{F}_K(\Delta, \tau_1), \ldots, \mathsf{F}_K(\Delta, \tau_n))$. Our key technical tool for achieving efficient verification is the introduction of a pseudorandom function F with a new property that we call *amortized closed-form efficiency*: if one precomputes some information ω_f related to a program f with input identifiers $\tau_1, \ldots, \tau_n$, but *independent* of the data set Δ, then it is possible to use ω_f to compute W, for *any* data set Δ, very efficiently, e.g., in constant time. Amortized closed-form efficiency essentially extends the closed-form efficiency of Benabbas et al. [11] to the setting in which the *same* function f is evaluated on *many pseudorandom* inputs.[4]

If we consider the example mentioned before, then one can precompute the verification information ω_f for the function "variance of the air pollution levels at all time instants within a day" (without knowing the actual data), and then use such ω_f for verifying the computation of this statistic on *any* specific day (i.e., the data set) in constant time.

We propose an efficient instantiation of amortized closed-form efficient PRFs whose security is based on standard PRFs and on the Decision Linear assumption [13], thereby achieving amortized closed-form efficiency in constant time, i.e., independent of the input size n. Our PRF maps pairs of binary strings (Δ, τ) to pseudorandom values in a group $\mathbb{G}$ of prime order p. For this technical reason, we changed the Catalano-Fiore MAC (which works with a PRF mapping to $\mathbb{Z}_p$) so as to encode the MACs y into elements of the group $\mathbb{G}$, and we used pairings to "simulate" the ring behavior over $\mathbb{Z}_p$ for all those computations that require at most one multiplication, i.e., arithmetic circuits of degree bounded by 2.

1.3 Organization of the Paper

The paper is organized as follows. In Section 2 we review notation and basic definitions. In Section 3 we introduce the notions of multi-labeled programs and the definition of homomorphic message authenticators with efficient verification for multi-labeled programs. Next, Section 4 contains the description of two technical tools that will be important for the design of our new construction of homomorphic MACs: algorithms for the homomorphic evaluation of arithmetic circuits, and pseudorandom functions with amortized closed-form efficiency. Finally, in Section 5, we give our construction of homomorphic MACs with efficient verification, we discuss its efficiency, and we prove its security.

2 Preliminaries

In this section, we review the notation and some basic definitions that we will use in our work.

Notation. We will denote with $\lambda \in \mathbb{N}$ a security parameter. We say that a function $\epsilon : \mathbb{N} \to \mathbb{R}^+$ is negligible if and only if for every positive polynomial $p(\lambda)$ there exists $\lambda_0 \in \mathbb{N}$ such that for all $\lambda > \lambda_0$: $\epsilon(\lambda) < 1/p(\lambda)$. If S is a set, $x \leftarrow_{\mathcal{R}} S$ denotes the process of selecting x uniformly at random in S. If $\mathcal{A}$ is a probabilistic algorithm, $x \leftarrow_{\mathcal{R}} \mathcal{A}(\cdot)$ denotes the process of running $\mathcal{A}$ on some appropriate input and assigning its output to x.

Algebraic Tools. Let $\mathcal{G}(1^\lambda)$ be an algorithm that on input the security parameter 1^λ, outputs the description of bilinear groups $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ where $\mathbb{G}$ and $\mathbb{G}_T$ are groups of the same

[4] We notice that the amortized extension was necessary in our case: while previous works [11,25] used the PRF to obtain a shorter description of the function f (e.g., by defining the coefficients of a polynomial in a pseudorandom way), this is not possible in our case where the description of f remains arbitrary.

prime order $p > 2^\lambda$, $g \in \mathbb{G}$ is a generator and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an efficiently computable bilinear map. We call such an algorithm $\mathcal{G}$ a *bilinear group generator*.

Arithmetic Circuits. We review some useful definitions and facts of arithmetic circuits. We refer the interested reader to [51] for a useful survey on this subject.

An arithmetic circuit over a field $\mathbb{F}$ and a set of variables $X = \{\tau_1 \dots \tau_n\}$, is a directed acyclic graph with the following properties. Each node in the graph is called *gate*. Gates with in-degree 0 are called *input* gates (or input nodes) while gates with out-degree 0 are called *output* gates. Each input gate is labeled by either a *variable* or a *constant*. Variable input nodes are labeled with binary strings $\tau_1, \dots, \tau_n$, and can take arbitrary values in $\mathbb{F}$. A constant input node instead is labeled with some constant c and it can take only some fixed value $c \in \mathbb{F}$.

Gates with in-degree and out-degree greater than 0 are called *internal* gates. Each internal gate is labeled with an arithmetic operation symbol. Gates labeled with $+$ are called sum gates, while gates labeled with $\times$ are called product gates. In this paper, we consider circuits with a single output node and where the in-degree of each internal gate is 2. The *size* of the circuit is the number of its gates. The *depth* of the circuit is the length of the longest path from input to output.

Arithmetic circuits evaluate polynomials in the following way. Input gates compute the polynomial defined by their labels. Sum gates compute the polynomial obtained by the sum of the (two) polynomials on their incoming wires. Product gates compute the product of the two polynomials on their incoming wires. The output of the circuit is the value contained on the outgoing wire of the output gate. The *degree of a gate* is defined as the total degree of the polynomial computed by that gate. The *degree of a circuit* is defined as the maximal degree of all gates in the circuit.

We stress that arithmetic circuits should be seen as computing *specific* polynomials in $\mathbb{F}[X]$ rather than functions from $\mathbb{F}^{|X|}$ to $\mathbb{F}$. In this paper, we restrict our interest to families of polynomials $\{f_\lambda\}$ over $\mathbb{F}$ which have degree bounded by 2.

3 Homomorphic Message Authenticators with Efficient Verification

Homomorphic message authenticators were first defined by Gennaro and Wichs [30]. Their definition was tailored to the model of labeled programs defined therein. Roughly speaking, a labeled program is a function f (e.g., a circuit) which takes in n variable inputs such that each of these variables is assigned a label τ (e.g., a binary string). One may think of such labeling of variables as a way to give useful names to the variables of a program. Using this model, homomorphic message authenticators were defined in [30] in such a way that a message m is authenticated with respect to a label τ. Binding m with τ essentially means that the value m can be assigned to those input variables of a labeled program f whose label is τ. This, however, imposes a limitation: a label *cannot* be re-used for multiple messages, i.e., one cannot authenticate two different messages m, m' with respect to the same label τ. This limitation makes perfect sense if one considers labeling of the data as a way to uniquely "categorize" the data, which is useful, for instance, in cases where a user outsources her data to a remote server and does not keep a local copy of the data. However, for the purpose of labeling programs, the re-use limitation also requires changing the labeling of the variable inputs of f whenever f is executed on a different set of inputs.

In other words, labels are useful to identify both concrete data items and variable inputs of programs. The current definition of homomorphic MACs, however, focuses more on a labeling mechanism for data items, instead of capturing the notion of identifying the program inputs. In the next section, we bridge this gap by introducing so-called *multi-labels* that aim to capture both useful

properties of labels: program variable labeling and data labeling. Thereafter, we give a definition of homomorphic MACs for multi-labeled programs.

3.1 Multi-Labeled Programs

We elaborate a variation of labeled programs that we call *multi-labeled programs*. The basic idea behind our model is to introduce the notion of a *multi-label* L, which consists of two parts: a *data set identifier* Δ and an *input identifier* τ. Input identifiers, in isolation, are used to label the variable inputs of a function f, whereas the combination of both, i.e., the full multi-label $\mathsf{L} = (\Delta, \tau)$, is used to uniquely identify a specific data item. Precisely, binding a value m with multi-label (Δ, τ) means that m can be assigned to those input variables with input identifier τ. The pair (Δ, τ) is necessary to uniquely identify m. While one can still not re-use a pair (Δ, τ) for authentication purposes, one can re-use the input identifier τ, instead.

For the sake of illustration, consider the multi-labeled approach as a separation of data items into two independent dimensions. One might think of a database table, e.g., storing air pollution levels, where some function $f : \mathcal{M}^n \to \mathcal{M}$ is evaluated over n columns (labeled $\tau_1, \ldots, \tau_n$). Each such column could represent a point in time, e.g., 7:05, 07:10, etc. This computation is performed for each row (labeled Δ_i) of the table. Each such row could represent a different day, e.g., 2013/08/14, 2013/08/15, etc. We hence evaluate $f_{\Delta_i}(\tau_1, \ldots, \tau_n)$ for each row i, hence for each day.

Labeled Programs. First, we review the notion of labeled programs introduced by Gennaro and Wichs [30]. While this notion was given for the case of Boolean circuits $f : \{0,1\}^n \to \{0,1\}$, here we generalize it to the case of any function f defined over an appropriate set $\mathcal{M}$. A *labeled program* $\mathcal{P}$ is defined by a tuple $(f, \tau_1, \ldots, \tau_n)$ where $f : \mathcal{M}^n \to \mathcal{M}$ is a function on n variables, and each $\tau_i \in \{0,1\}^*$ is the label of the i-th variable input of f. Labeled programs allow for composition as follows. Given labeled programs $\mathcal{P}_1, \ldots, \mathcal{P}_t$ and given a function $g : \mathcal{M}^t \to \mathcal{M}$, the *composed program* $\mathcal{P}^*$ corresponds to evaluating g on the outputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$. The composed program is compactly denoted as $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$. The labeled inputs of $\mathcal{P}^*$ are all distinct labeled inputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$, i.e., all inputs with the same label are grouped together in a single input of the new program. If $f_{id} : \mathcal{M} \to \mathcal{M}$ is the canonical identity function and $\tau \in \{0,1\}^*$ is a label, then $\mathcal{I}_\tau = (f_{id}, \tau)$ denotes the *identity program* for input label τ. Notice that any program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ can be expressed as the composition of n identity programs $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \ldots, \mathcal{I}_{\tau_n})$.

Multi-labeled Programs. Intuitively, multi-labeled programs are an extension of labeled programs in which a labeled program $\mathcal{P}$ is augmented with a data set identifier Δ. Formally, we define a *multi-labeled program* $\mathcal{P}_\Delta$ as a pair $(\mathcal{P}, \Delta)$ where $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ is a labeled program (as defined above) and $\Delta \in \{0,1\}^*$ is a binary string called the *data set identifier*. Multi-labeled programs allow for composition within the same data set in the most natural way, i.e., given multi-labeled programs $(\mathcal{P}_1, \Delta), \ldots, (\mathcal{P}_t, \Delta)$ having the same data set identifier Δ, and given a function $g : \mathcal{M}^t \to \mathcal{M}$, the *composed multi-labeled program* $\mathcal{P}^*_\Delta$ is the pair $(\mathcal{P}^*, \Delta)$ where $\mathcal{P}^*$ is the composed program $g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$, and Δ is the data set identifier shared by all the $\mathcal{P}_i$. If $f_{id} : \mathcal{M} \to \mathcal{M}$ is the canonical identity function and $\mathsf{L} = (\Delta, \tau) \in (\{0,1\}^*)^2$ is a multi-label, then $\mathcal{I}_\mathsf{L} = (f_{id}, \mathsf{L})$ denotes the *identity multi-labeled program* for data set Δ and input label τ. As for labeled programs, any multi-labeled program $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$ can also be expressed as the composition of n identity multi-labeled programs: $\mathcal{P}_\Delta = f(\mathcal{I}_{\mathsf{L}_1}, \ldots, \mathcal{I}_{\mathsf{L}_n})$ where $\mathsf{L}_i = (\Delta, \tau_i)$.

It is worth noting that, in the notation of [30], a multi-labeled program $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$ is essentially a labeled program $(f, \mathsf{L}_1, \ldots, \mathsf{L}_n)$ where each string L_i is a multi-label (Δ, τ_i). The main

difference here is the (explicit) notion of labeled data sets that we use in order to group together several inputs, similarly to the definition used for homomorphic signatures [15,26]. This explicit splitting will turn out to be crucial in order to achieve the desired property of efficient verification.

3.2 Homomorphic MACs for Multi-Labeled Programs

We review the notion of homomorphic message authenticators [30,17]. We have adapted the definition to our model of multi-labeled programs as defined in the previous section.

Definition 1. *A* homomorphic message authenticator scheme $\mathsf{HomMAC\text{-}ML}$ for multi-label programs *is a tuple of algorithms* $(\mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Ver}, \mathsf{Eval})$ *satisfying four properties:* authentication correctness*,* evaluation correctness*,* succinctness*, and* security*. More precisely:*

$\mathsf{KeyGen}(1^\lambda)$**:** *given the security parameter* λ*, the key generation algorithm outputs a secret key* sk *and a public evaluation key* ek*.*

$\mathsf{Auth}(\mathsf{sk}, \mathsf{L}, m)$**:** *given the secret key* sk*, a multi-label* $\mathsf{L} = (\Delta, \tau)$ *and a message* $m \in \mathcal{M}$*, it outputs a tag* σ*.*

$\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_\Delta, m, \sigma)$**:** *given the secret key* sk*, a multi-labeled program* $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$*, a message* $m \in \mathcal{M}$*, and a tag* σ*, the verification algorithm outputs 0 (reject) or 1 (accept).*

$\mathsf{Eval}(\mathsf{ek}, f, \boldsymbol{\sigma})$**:** *on input the evaluation key* ek*, a circuit* $f : \mathcal{M}^n \to \mathcal{M}$ *and a vector of tags* $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$*, the evaluation algorithm outputs a new tag* σ*.*

Authentication Correctness. Informally speaking, a homomorphic MAC has authentication correctness if any tag σ generated by the algorithm $\mathsf{Auth}(\mathsf{sk}, \mathsf{L}, m)$ authenticates m with respect to the identity program $\mathcal{I}_\mathsf{L}$. More formally, we say that a scheme $\mathsf{HomMAC\text{-}ML}$ satisfies authentication correctness if for any message $m \in \mathcal{M}$, all keys $(\mathsf{sk}, \mathsf{ek}) \leftarrow_\mathcal{R} \mathsf{KeyGen}(1^\lambda)$, any multi-label $\mathsf{L} = (\Delta, \tau) \in (\{0,1\}^*)^2$, and any tag $\sigma \leftarrow_\mathcal{R} \mathsf{Auth}(\mathsf{sk}, \mathsf{L}, m)$, we have that $\mathsf{Ver}(\mathsf{sk}, \mathcal{I}_\mathsf{L}, m, \sigma) = 1$ holds with probability 1.

Evaluation Correctness. This property aims at capturing that if the evaluation algorithm is run on a vector of tags $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ such that each σ_i authenticates some message m_i as the output of a multi-labeled program $(\mathcal{P}_i, \Delta)$, then the tag σ produced by Eval must authenticate $f(m_1, \ldots, m_n)$ as the output of the composed program $(f(\mathcal{P}_1, \ldots, \mathcal{P}_n), \Delta)$. More formally, let us fix a pair of keys $(\mathsf{sk}, \mathsf{ek}) \leftarrow_\mathcal{R} \mathsf{KeyGen}(1^\lambda)$, a function $g : \mathcal{M}^t \to \mathcal{M}$ and any set of message/program/tag triples $\{(m_i, \mathcal{P}_{\Delta,i}, \sigma_i)\}_{i=1}^t$ such that all multi-labeled programs $\mathcal{P}_{\Delta,i} = (\mathcal{P}_i, \Delta)$ (i.e., share the same data set identifier Δ) and $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_{\Delta,i}, m_i, \sigma_i) = 1$. If $m^* = g(m_1, \ldots, m_t)$, $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$, and $\sigma^* = \mathsf{Eval}(\mathsf{ek}, g, (\sigma_1, \ldots, \sigma_t))$, then $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}^*_\Delta, m^*, \sigma^*) = 1$ holds with probability 1.

Succinctness. The size of a tag is bounded by some fixed polynomial in the security parameter, which is independent of the number n of inputs taken by the evaluated circuit.

Security. A homomorphic MAC has to satisfy the following notion of unforgeability. Let $\mathsf{HomMAC\text{-}ML}$ be a homomorphic MAC scheme as defined above and let $\mathcal{A}$ be an adversary. $\mathsf{HomMAC\text{-}ML}$ is said to be unforgeable if for every PPT adv. $\mathcal{A}$, we have $\Pr[\mathsf{HomUF{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}(\lambda) = 1] \leq \epsilon(\lambda)$ where $\epsilon(\lambda)$ is a negligible function. The experiment $\mathsf{HomUF{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}(\lambda)$ is the one defined below.

Setup The challenger generates $(\mathsf{sk}, \mathsf{ek}) \leftarrow_\mathcal{R} \mathsf{KeyGen}(1^\lambda)$ and gives ek to $\mathcal{A}$.

Authentication queries The adversary can adaptively ask for tags on multi-labels and messages of its choice. Given a query (L, m) where $\mathsf{L} = (\Delta, \tau)$, the challenger proceeds as follows: If (L, m) is the first query with data set identifier Δ, then the challenger initializes an empty list $T_\Delta = \emptyset$ for data set identifier Δ. If T_Δ does not contain a tuple $(\tau, \cdot)$ (i.e., the multi-label (Δ, τ) was never queried), the challenger computes $\sigma \leftarrow_{\mathcal{R}} \mathsf{Auth}(\mathsf{sk}, \mathsf{L}, m)$, returns σ to $\mathcal{A}$ and updates the list $T_\Delta \leftarrow T_\Delta \cup (\tau, m)$. If $(\tau, m) \in T_\Delta$ (i.e., the query was previously made), then the challenger replies with the same tag generated before. If T_Δ contains a tuple (τ, m') for some message $m' \neq m$, then the challenger ignores the query.

Verification queries The adversary has access to a verification oracle as follows: Given a query $(\mathcal{P}_\Delta, m, \sigma)$ from $\mathcal{A}$, the challenger replies with the output of $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_\Delta, m, \sigma)$.

Forgery The adversary terminates the experiment by returning a forgery $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ for some $\mathcal{P}^*_{\Delta^*} = (\mathcal{P}^*, \Delta^*)$ and $\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*)$. Notice that, equivalently, $\mathcal{A}$ can implicitly return such a tuple as a verification query $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ during the experiment.

Before describing the outcome of this experiment, we review the notion of well-defined programs with respect to a list T_Δ [17]. A labeled program $\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*)$ is *well-defined with respect to* T_{Δ^*} if either one of the following two cases holds:

- there exist messages $m_1, \ldots, m_n$ such that the list T_{Δ^*} contains all tuples $(\tau_1^*, m_1), \ldots, (\tau_n^*, m_n)$. Intuitively, this means that the entire input space of f for data set Δ^* has been authenticated.
- there exist indices $i \in \{1, \ldots, n\}$ such that $(\tau_i^*, \cdot) \notin T_{\Delta^*}$ (i.e., $\mathcal{A}$ never asked authentication queries with multi-label (Δ^*, τ_i^*)), and the function $f^*(\{m_j\}_{(\tau_j, m_j) \in T_{\Delta^*}} \cup \{\tilde{m}_j\}_{(\tau_j, \cdot) \notin T_{\Delta^*}})$ outputs the same value for all possible choices of $\tilde{m}_j \in \mathcal{M}$. Intuitively, this case means that the unauthenticated inputs never contribute to the computation of f.

To define the output of the experiment $\mathsf{HomUF{-}CMA}$, we say it outputs 1 if and only if $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}^*_{\Delta^*}, m^*, \sigma^*) = 1$ and one of the following conditions holds:

- *Type 1 Forgery:* no list T_{Δ^*} was created during the game, i.e., no message m has been authenticated with respect to a data set identifier Δ^* during the experiment.
- *Type 2 Forgery:* $\mathcal{P}^*$ is well-defined w.r.t. T_{Δ^*} and $m^* \neq f^*(\{m_j\}_{(\tau_j, m_j) \in T_{\Delta^*}})$, i.e., m^* is not the correct output of the labeled program $\mathcal{P}^*$ when executed on previously authenticated messages $(m_1, \ldots, m_n)$.
- *Type 3 Forgery:* $\mathcal{P}^*$ is not well-defined w.r.t. T_{Δ^*}.

Our definition is obtained by extending the one by Catalano and Fiore [17] to our model of multi-labeled programs. The resulting definition is very close to the one proposed by Freeman for homomorphic signatures [26], with the exception that we allow for arbitrary labels, and we do not impose any a-priori fixed bound on the number of elements in a data set.

In the most general case where f can be any function, it might not be possible to efficiently (i.e., in polynomial time) check whether a program $\mathcal{P}$ is well-defined w.r.t. a list T. However, for more specific classes of computations, this is not an issue. For example, Freeman showed that this is not a problem for linear functions [26]. In the following proposition, we show a similar result for the classes of computations considered in our work, i.e., arithmetic circuits defined over the finite field $\mathbb{Z}_p$ where p is a prime of roughly λ bits, and whose degree d is bounded by a polynomial. In particular, we show that any adversary who wins by producing a Type 3 forgery can be converted into one who outputs a Type 2 forgery.

Proposition 1. *Let $\lambda \in \mathbb{N}$ be the security parameter, let $p > 2^\lambda$ be a prime number, and let $\{f_\lambda\}$ be a family of arithmetic circuits over $\mathbb{Z}_p$ whose degree is bounded by some polynomial $d = poly(\lambda)$. If for any adversary $\mathcal{B}$ producing a Type 2 forgery we have that $\Pr[\mathsf{HomUF{-}CMA}_{\mathcal{B},\mathsf{HomMAC\text{-}ML}}(\lambda) = 1] \leq \epsilon$, then for any adversary $\mathcal{A}$ producing a Type 3 forgery it holds $\Pr[\mathsf{HomUF{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}(\lambda) = 1] \leq \epsilon + d/p$.*

Proof. The proof is by contradiction. Assume there exists an adversary $\mathcal{A}$ such that

$$\Pr[\mathsf{HomUF{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}(\lambda) = 1] > \epsilon + d/p$$

and $\mathcal{A}$ produces a Type 3 forgery, then we show an adversary $\mathcal{B}$ such that

$$\Pr[\mathsf{HomUF{-}CMA}_{\mathcal{B},\mathsf{HomMAC\text{-}ML}}(\lambda) = 1] > \epsilon$$

by producing a Type 2 forgery. We construct $\mathcal{B}$ out of $\mathcal{A}$ as follows.

$\mathcal{B}$ first runs the adversary $\mathcal{A}$ to obtain a Type 3 forgery $(m^*, \mathcal{P}^*_{\Delta^*}, \sigma^*)$, i.e., $\mathcal{B}$ simulates the $\mathsf{HomUF{-}CMA}$ game to $\mathcal{A}$ by forwarding all messages back and forth from its challenger. Let $\mathcal{P}^*_{\Delta^*} = (\mathcal{P}^*, \Delta^*)$ where $\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*)$, and assume that $\mathcal{B}$ maintains the lists of queries made by $\mathcal{A}$ as done by the challenger. Let T_{Δ^*} be the list of queries for the data set Δ^*. Since $\mathcal{P}^*$ is not well-defined w.r.t. T_{Δ^*} there exists an index $j \in \{1, \ldots, n\}$ such that $(\tau_j^*, \cdot) \notin T_{\Delta^*}$. $\mathcal{B}$ proceeds as follows. For all $j \in \{1, \ldots, n\}$ such that $(\tau_j^*, \cdot) \notin T_{\Delta^*}$, $\mathcal{B}$ chooses random messages $r_j \leftarrow_{\mathcal{R}} \mathbb{Z}_p$, queries its challenger for tags on $((\Delta^*, \tau_j^*), r_j)$, and finally outputs $(m^*, \mathcal{P}^*_{\Delta^*}, \sigma^*)$ as a forgery.

To complete the proof, we show that in the experiment $\mathsf{HomUF{-}CMA}_{\mathcal{B},\mathsf{HomMAC\text{-}ML}}(\lambda)$ played by $\mathcal{B}$ the tuple $(m^*, \mathcal{P}^*_{\Delta^*}, \sigma^*)$ is a Type 2 forgery with probability $1 - d/p$. First, notice that by definition, $(m^*, \mathcal{P}^*_{\Delta^*}, \sigma^*)$ verifies correctly, and that in $\mathcal{B}$'s experiment, the program $\mathcal{P}^*$ is well-defined. Second, we argue that $\Pr[m^* = f^*(\{m_j\}_{(\tau_j, m_j) \in T_{\Delta^*}} \cup \{r_j\}_{(\tau_j, \cdot) \notin T_{\Delta^*}})] \leq d/p$. This bound follows from the fact that the program $\mathcal{P}^*$ in the experiment simulated to $\mathcal{A}$ is not well-defined, i.e., the polynomial represented by the circuit f^* for fixed values $\{m_j\}_{(\tau_j, m_j) \in T_{\Delta^*}}$ is not a constant function. Therefore, if d is an upper bound on the degree of such polynomial it is not hard to see that over the random choices of r_j in $\mathbb{Z}_p$, the above equality will be satisfied with probability at most d/p. So, the tuple is a forgery of Type 2 with probability at least $1 - d/p$. Hence, we can bound $\mathcal{B}$'s probability of success by

$$\begin{aligned}
\Pr[\mathsf{HomUF{-}CMA}_{\mathcal{B},\mathsf{HomMAC\text{-}ML}}(\lambda) = 1] &\geq \Pr[\mathsf{HomUF{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}(\lambda) = 1](1 - d/p) \\
&\geq \Pr[\mathsf{HomUF{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}(\lambda) = 1] - d/p \\
&> \epsilon
\end{aligned}$$

which concludes the proof. □

3.3 Homomorphic MACs with Efficient Verification for Multi-Labeled Programs

In this section we introduce a new property for homomorphic MACs that we call *efficient verification*. Informally, a homomorphic MAC satisfies efficient verification if it is possible to verify a tag σ against a multi-labeled program $\mathcal{P}_\Delta = (\mathcal{P}, \Delta)$ in less time than that required to compute $\mathcal{P}$. We define this efficiency property in an amortized sense, so that the verification is more efficient when the same program $\mathcal{P}$ is executed on different data sets. The formal definition follows.

Definition 2. *Let* $\mathsf{HomMAC\text{-}ML} = (\mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Ver}, \mathsf{Eval})$ *be a homomorphic MAC scheme for multi-labeled programs as defined in the previous section.* HomMAC-ML satisfies efficient verification *if there exist two additional algorithms* $(\mathsf{VerPrep}, \mathsf{EffVer})$ *as follows:*

$\mathsf{VerPrep}(\mathsf{sk}, \mathcal{P})$**:** *on input the secret key* sk *and a labeled program* $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$*, this algorithm generates a concise verification key* $\mathsf{VK}_{\mathcal{P}}$*. We stress that this verification key does* not *depend on any data set identifier* Δ*.*

$\mathsf{EffVer}(\mathsf{sk}, \mathsf{VK}_{\mathcal{P}}, \Delta, m, \sigma)$**:** *given the secret key* sk*, a verification key* $\mathsf{VK}_{\mathcal{P}}$*, a data set identifier* Δ*, a message* $m \in \mathcal{M}$ *and a tag* σ*, the efficient verification algorithm outputs 0 (reject) or 1 (accept).*

The above algorithms are required to satisfy the following two properties:

Correctness. *Let* $(\mathsf{sk}, \mathsf{ek}) \leftarrow_{\mathcal{R}} \mathsf{KeyGen}(1^\lambda)$ *be honestly generated keys, and* $(\mathcal{P}_\Delta, m, \sigma)$ *be any program/message/tag tuple with* $\mathcal{P}_\Delta = (\mathcal{P}, \Delta)$ *such that* $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_\Delta, m, \sigma) = 1$*. Then, for every* $\mathsf{VK}_{\mathcal{P}} \leftarrow_{\mathcal{R}} \mathsf{VerPrep}(\mathsf{sk}, \mathcal{P})$*, we have* $\Pr[\mathsf{EffVer}(\mathsf{sk}, \mathsf{VK}_{\mathcal{P}}, \Delta, m, \sigma) = 1] = 1$*.*

Amortized Efficiency. *Let* $\mathcal{P}_\Delta = (\mathcal{P}, \Delta)$ *be a program, let* $(m_1, \ldots, m_n) \in \mathcal{M}^n$ *be any vector of inputs, and let* $t(n)$ *be the time required to compute* $\mathcal{P}(m_1, \ldots, m_n)$*. If* $\mathsf{VK}_{\mathcal{P}} \leftarrow \mathsf{VerPrep}(\mathsf{sk}, \mathcal{P})$*, then the time required for* $\mathsf{EffVer}(\mathsf{sk}, \mathsf{VK}_{\mathcal{P}}, \Delta, m, \sigma)$ *is* $O(1)$*, i.e., independent of* n*.*

Notice that in our efficiency requirement, we do not include the time needed to compute $\mathsf{VK}_{\mathcal{P}}$. The reason is, since $\mathsf{VK}_{\mathcal{P}}$ is independent of Δ, the same $\mathsf{VK}_{\mathcal{P}}$ can be re-used in many verifications involving the same labeled program $\mathcal{P}$ but many different Δ. In this sense, the cost of computing $\mathsf{VK}_{\mathcal{P}}$ is *amortized* over many verifications of the same function on different data sets.

Application to Verifiable Computation on Outsourced Data. A homomorphic MAC scheme with efficient verification can be easily used to obtain a protocol for verifiable delegation of computations on outsourced data, satisfying the requirements (1)–(5) mentioned in Section 1. Below, we sketch such a protocol between a client $\mathcal{C}$ and a server $\mathcal{S}$:

Setup: $\mathcal{C}$ generates the keys $(\mathsf{sk}, \mathsf{ek}) \leftarrow_{\mathcal{R}} \mathsf{KeyGen}(1^\lambda)$ for a homomorphic MAC, sends ek to $\mathcal{S}$ and stores sk.

Data Outsourcing: to outsource a value m, $\mathcal{C}$ first authenticates m wrt. some multi-label L, i.e., $\sigma \leftarrow_{\mathcal{R}} \mathsf{Auth}(\mathsf{sk}, \mathsf{L}, m)$, and then sends (m, L, σ) to the server. It is easy to see that this phase satisfies the requirements of unbounded storage (4) and function independence (5).

Client's Preparation: assume that $\mathcal{C}$ needs to evaluate a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ on some of its outsourced data sets. In this preparation phase (offline), the client computes and stores $\mathsf{VK}_{\mathcal{P}} \leftarrow_{\mathcal{R}} \mathsf{VerPrep}(\mathsf{sk}, \mathcal{P})$ (independently of any Δ).

Delegation: when the client wants to compute $\mathcal{P}$ on a data set Δ (online), it simply sends $(\mathcal{P}, \Delta)$ to the server.[5]

Computation: to compute $(\mathcal{P}, \Delta)$, where $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$, the server first looks for the corresponding data $(m_1, \ldots, m_n)$ and tags $(\sigma_1, \ldots, \sigma_n)$ according to the labeling previously sent by $\mathcal{C}$. Next, $\mathcal{S}$ computes $m = f(m_1, \ldots, m_n)$ and $\sigma \leftarrow \mathsf{Eval}(\mathsf{ek}, f, \sigma_1, \ldots, \sigma_n)$, and sends (m, σ) to $\mathcal{C}$.

Verification: given the result (m, σ) sent by $\mathcal{S}$, the client checks that m is the correct output of the multi-labeled program $(\mathcal{P}, \Delta)$ by running $\mathsf{EffVer}(\mathsf{sk}, \mathsf{VK}_{\mathcal{P}}, \Delta, m, \sigma)$. By the amortized efficiency property of the homomorphic MAC, we obtain that $\mathcal{C}$ achieves amortized input-independent efficiency (3) – and thus also efficiency (2) – in verifying the delegated computations.

[5] While in general the description of $\mathcal{P}$ may be large, here we assume the case in which $\mathcal{P}$ has a succinct description, e.g., "daily variance of the air pollution levels at every 5 minutes". Hence, the cost of communicating $\mathcal{P}$ can, in fact, be ignored.

Finally, from the unforgeability of the homomorphic MAC, it is straightforward to see that the server cannot induce the client to accept incorrect results (1).

4 Utilities

This section provides some technical tools that will be useful to obtain our construction of homomorphic MACs with efficient verification.

4.1 Homomorphic Evaluation of Arithmetic Circuits

In the next two sections, we describe algorithms that allow for the homomorphic evaluation of an arithmetic circuit $f : \mathcal{M}^n \to \mathcal{M}$ over values defined in some appropriate set $\mathcal{J} \neq \mathcal{M}$.

Homomorphic Evaluation over Polynomials. As a first example, we consider the case in which $\mathcal{J}$ is a ring of polynomials. More formally, let $\mathcal{J}_{\mathsf{poly}} = \mathbb{Z}_p[x_1, \ldots, x_m]$ be the ring of polynomials in variables $x_1, \ldots, x_m$ over $\mathbb{Z}_p$. For every fixed tuple $\boldsymbol{a} = (a_1, \ldots, a_m) \in \mathbb{Z}_p^m$, let $\phi_{\boldsymbol{a}} : \mathcal{J}_{\mathsf{poly}} \to \mathbb{Z}_p$ be the function defined by $\phi_{\boldsymbol{a}}(y) = y(a_1, \ldots, a_m)$ for any $y \in \mathcal{J}_{\mathsf{poly}}$. By the substitution property of polynomials, $\phi_{\boldsymbol{a}}$ is a homomorphism from $\mathcal{J}_{\mathsf{poly}} = \mathbb{Z}_p[x_1, \ldots, x_m]$ to $\mathbb{Z}_p$, i.e., $\forall y_1, y_2 \in \mathcal{J}_{\mathsf{poly}}$ it holds: $\phi_{\boldsymbol{a}}(y_1 + y_2) = \phi_{\boldsymbol{a}}(y_1) + \phi_{\boldsymbol{a}}(y_2)$ and $\phi_{\boldsymbol{a}}(y_1 \cdot y_2) = \phi_{\boldsymbol{a}}(y_1) \cdot \phi_{\boldsymbol{a}}(y_2)$. By simple induction, we then observe that for a given arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$, there exists another circuit $\hat{f} : \mathcal{J}_{\mathsf{poly}}^n \to \mathcal{J}_{\mathsf{poly}}$ such that $\forall y_1, \ldots, y_n \in \mathcal{J}_{\mathsf{poly}}$: $\phi_{\boldsymbol{a}}(\hat{f}(y_1, \ldots, y_n)) = f(\phi_{\boldsymbol{a}}(y_1), \ldots, \phi_{\boldsymbol{a}}(y_n))$. The circuit $\hat{f}$ is structurally the same as f. The only difference is that in every gate the operation in $\mathbb{Z}_p$ is replaced by the corresponding operation over polynomials in $\mathbb{Z}_p[x_1, \ldots, x_m]$.

For every positive integer $m \in \mathbb{N}$ and a given arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$, we formally define the computation of $\hat{f}$ on $(y_1, \ldots, y_n) \in \mathcal{J}_{\mathsf{poly}}^n$ as an algorithm $\mathsf{PolyEval}(m, f, y_1, \ldots, y_n)$. Concretely, $\mathsf{PolyEval}$ is a simple algorithm that at every gate f_g, on input two polynomials $y_1, y_2 \in \mathcal{J}_{\mathsf{poly}}$, proceeds as follows: if f_g is an addition gate, it outputs $y = y_1 + y_2$ (i.e., it adds all coefficients component-wise); if f_g is a multiplication gate, it outputs $y = y_1 \cdot y_2$ (i.e., it uses the convolution operator on the coefficients). We notice that every multiplication gate increases the degree of y, and thus it also increases the number of its coefficients. In particular, if y_1, y_2 have degrees d_1, d_2 respectively, then the degree of $y = y_1 \cdot y_2$ is $d_1 + d_2$.

For any homomorphism $\phi_{\boldsymbol{a}}$ defined by a tuple $\boldsymbol{a} = (a_1, \ldots, a_m) \in \mathbb{Z}_p^m$, and for any circuit f and any values $y_1, \ldots, y_n \in \mathcal{J}_{\mathsf{poly}}$ the following property clearly holds for $\mathsf{PolyEval}$:

$$\phi_{\boldsymbol{a}}(\mathsf{PolyEval}(m, f, y_1, \ldots, y_n)) = f(\phi_{\boldsymbol{a}}(y_1), \ldots, \phi_{\boldsymbol{a}}(y_n)). \tag{1}$$

Above, we gave for completeness a generic definition of $\mathsf{PolyEval}$ for any possible $m \in \mathbb{N}$. However, we observe that in our work we will use $\mathsf{PolyEval}$ only with $m = 1$ and $m = 2$.

Homomorphic Evaluation over Bilinear Groups. As a second example, we show how to homomorphically evaluate arithmetic circuits, of degree at most 2, over prime order groups with bilinear maps. Let $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ be the description of bilinear groups where $\mathbb{G}$ has prime order p. If we fix a generator $g \in \mathbb{G}$, then $\mathbb{G}$ and the additive group $(\mathbb{Z}_p, +)$ are isomorphic by considering the isomorphism $\phi_g(x) = g^x$ for every $x \in \mathbb{Z}_p$. Similarly, by the property of the pairing function e, we also have that $\mathbb{G}_T$ and the additive group $(\mathbb{Z}_p, +)$ are isomorphic by considering $\phi_{g_T}(x) = e(g, g)^x$. Since ϕ_g and ϕ_{g_T} are isomorphisms there also exist the corresponding inverses $\phi_g^{-1} : \mathbb{G} \to \mathbb{Z}_p$ and $\phi_{g_T}^{-1} : \mathbb{G}_T \to \mathbb{Z}_p$, even though these are not known to be efficiently computable.

For every arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$ of degree at most 2, we define $\mathsf{GroupEval}(f, X_1, \ldots, X_n)$ to be the algorithm which homomorphically evaluates f with inputs in $\mathbb{G}$ and output in $\mathbb{G}_T$ in such a way that, for every tuple $(X_1, \ldots, X_n) \in \mathbb{G}^n$, and every such circuit f, it holds

$$\phi_{g_T}^{-1}(\mathsf{GroupEval}(f, X_1, \ldots, X_n)) = f(\phi_g^{-1}(X_1), \ldots, \phi_g^{-1}(X_n)) \tag{2}$$

or, equivalently, for every $(X_1, \ldots, X_n) \in \mathbb{G}^n$, we have that

$$\mathsf{GroupEval}(f, X_1, \ldots, X_n) = e(g, g)^{f(x_1, \ldots, x_n)} : \forall i = 1, \ldots, n : x_i = \phi_g^{-1}(X_i). \tag{3}$$

Notice that the equivalence of equations (2) and (3) holds by using the fact that

$$\mathsf{GroupEval}(f, X_1, \ldots, X_n) = \phi_{g_T}(f(\phi_g^{-1}(X_1), \ldots, \phi_g^{-1}(X_n))).$$

Given a circuit f of degree at most 2, and given an n-tuple of values $(X_1, \ldots, X_n) \in \mathbb{G}^n$, $\mathsf{GroupEval}$ intuitively proceeds as follows. It computes additions by using the group operation in $\mathbb{G}$ or in $\mathbb{G}_T$. To compute multiplications, it uses the pairing function, e.g., $R = e(R_1, R_2)$, thus "lifting" the result to the group $\mathbb{G}_T$. By our assumption on the degree of f, one can see that multiplication is well defined.

More precisely, given a circuit f and an n-tuple of values $(X_1, \ldots, X_n) \in \mathbb{G}^n$, $\mathsf{GroupEval}$ proceeds gate-by-gate as follows. For an addition gate f_+ there are four cases depending on the type of its inputs. Namely, for inputs

- $X_1 \in \mathbb{G}$ and $X_2 \in \mathbb{G}$, output $X \in \mathbb{G}$ with $X = X_1 \cdot X_2$.
- $\hat{X}_1 \in \mathbb{G}_T$ and $\hat{X}_2 \in \mathbb{G}_T$, output $\hat{X} \in \mathbb{G}_T$ with $\hat{X} = \hat{X}_1 \cdot \hat{X}_2$.
- $\hat{X}_1 \in \mathbb{G}_T$ and $X_2 \in \mathbb{G}$, output $\hat{X} \in \mathbb{G}_T$ with $\hat{X} = \hat{X}_1 \cdot e(X_2, g)$.
- $X_1 \in \mathbb{G}$ and $\hat{X}_2 \in \mathbb{G}_T$, output $\hat{X} \in \mathbb{G}_T$ with $\hat{X} = e(X_1, g) \cdot \hat{X}_2$.

For a multiplication gate $f_\times$ there is only a single case with two variable inputs where both $X_1, X_2 \in \mathbb{G}$. The reason is that multiplication "lifts" the evaluation from $\mathbb{G}$ to $\mathbb{G}_T$. Assuming that $\deg(f) \leq 2$, we know that every multiplication must take as input two terms of degree 1, hence two elements in $\mathbb{G}$. Multiplication gates thus output $\hat{X} \in \mathbb{G}_T$ with $\hat{X} = e(X_1, X_2)$. For the multiplication of $X_1 \in \mathbb{G} \cup \mathbb{G}_T$ with a constant $c \in \mathbb{Z}_p$, output $X = (X_1)^c$.

The final output X^* of $\mathsf{GroupEval}$ is the output of the last gate of the circuit. In case no multiplication has occurred while evaluating the circuit, i.e., $X^* \in \mathbb{G}$, output $e(X^*, g) \in \mathbb{G}_T$ as final result.

The following theorem proves that $\mathsf{GroupEval}$ achieves the desired homomorphic property:

Theorem 1. *Let* $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ *be the description of bilinear groups. Then, the algorithm* $\mathsf{GroupEval}$ *satisfies Equation* (3)*, i.e.,* $\forall (X_1, \ldots, X_n) \in \mathbb{G}^n$*:* $\mathsf{GroupEval}(f, X_1, \ldots, X_n) = e(g, g)^{f(x_1, \ldots, x_n)}$ *for the unique values* $\{x_i\}_{i=1}^n \in \mathbb{Z}_p$ *such that* $X_i = g^{x_i}$*.*

Proof. The proof is by induction on the structure of f and proceeds gate-by-gate. We show the case for the identity circuit f_{id} first, and then we show the case for addition and multiplication gates by an inductive argument. For the identity circuit: $\mathsf{GroupEval}(f_{id}, X) = e(X, g) = e(g^x, g) = e(g, g)^x = e(g, g)^{f_{id}(x)}$. For the inductive case, we distinguish three cases depending on the number of previous multiplications in the two input branches of the gates:

(1) *No multiplication before.*

- The evaluation of an addition gate f_+ for $X_1, X_2 \in \mathbb{G}$ yields $X = X_1 \cdot X_2 \stackrel{\text{ind}}{=} g^{x_1} g^{x_2} = g^{x_1+x_2}$.
  Eventually, X is 'lifted' to $\mathbb{G}_T$ in the case of a subsequent multiplication or by the final step of $\mathsf{GroupEval}$, hence we eventually obtain $\hat{X} = e(X, g) = e(g, g)^{x_1+x_2}$.
- The evaluation of a multiplication gate $f_\times$ for variable inputs $X_1, X_2 \in \mathbb{G}$ yields $\hat{X} = e(X_1, X_2) \stackrel{\text{ind}}{=} e(g^{x_1}, g^{x_2}) = e(g, g)^{x_1 x_2}$.
- The evaluation of a multiplication gate $f_\times$ for input $X_1 \in \mathbb{G}$ with a constant $c \in \mathbb{Z}_p$ yields $X = (X_1)^c \stackrel{\text{ind}}{=} (g^{x_1})^c = g^{x_1 c}$.

(2) *Multiplication in one input branch.*
W.l.o.g., we assume the multiplication to have occurred in the left branch, hence $\hat{X}_1 \in \mathbb{G}_T$.
- The evaluation of an addition gate f_+ with $X_2 \in \mathbb{G}$ yields $\hat{X} = \hat{X}_1 \cdot e(X_2, g) \stackrel{\text{ind}}{=} e(g, g)^{x_1} \cdot e(g, g)^{x_2} = e(g, g)^{x_1+x_2}$.
- The evaluation of a multiplication gate $f_\times$ with a constant $c \in \mathbb{Z}_p$ yields $\hat{X} = (X_1)^c \stackrel{\text{ind}}{=} (e(g, g)^{x_1})^c = e(g, g)^{x_1 c}$.
- The evaluation of a multiplication gate $f_\times$ for two variable inputs is undefined for this case because of $\deg(f) \leq 2$.

(3) *Multiplications in both input branches.*
- The evaluation of an addition gate f_+ with inputs $\hat{X}_1, \hat{X}_2 \in \mathbb{G}_T$ yields $\hat{X} = \hat{X}_1 \cdot \hat{X}_2 \stackrel{\text{ind}}{=} e(g, g)^{x_1} \cdot e(g, g)^{x_2} = e(g, g)^{x_1+x_2}$.
- The evaluation of a multiplication gate $f_\times$ for two variable inputs is undefined for this case because of $\deg(f) \leq 2$. □

4.2 Pseudorandom Functions with Amortized Closed-Form Efficiency

Here we introduce one of most important technical tools for our construction, that is the notion of pseudorandom functions with *amortized closed-form efficiency*. This primitive is an extension of closed-form efficient PRFs proposed by Benabbas et al. [11], and later refined by Fiore and Gennaro [25]. As we will show in Section 5, this new notion of PRFs will be crucial for achieving the property of efficient verification in our homomorphic MAC realization.

In a nutshell, closed-form efficient PRFs [11] are defined like standard PRFs with the additional requirement of satisfying the following efficiency property. Assume there exists a computation $\mathsf{Comp}(R_1, \ldots, R_n, \boldsymbol{z})$ which takes random inputs $R_1, \ldots, R_n$ and arbitrary inputs $\boldsymbol{z}$, and runs in time $t(n, |\boldsymbol{z}|)$. Also, think of the case in which each R_i is generated as $\mathsf{F}_K(\mathsf{L}_i)$. Then the PRF F is said to satisfy closed-form efficiency for $(\mathsf{Comp}, \mathbf{L})$ if, by knowing the seed K, one can compute $\mathsf{Comp}(\mathsf{F}_K(\mathsf{L}_1), \ldots, \mathsf{F}_K(\mathsf{L}_n), \boldsymbol{z})$ in time strictly less than t. Here, the key observation is that in the pseudorandom case all the R_i values have a shorter "closed-form" representation (as function of K), and this might also allow for a shorter closed-form representation of the computation.

Starting from the above considerations, we introduce a new property for PRFs that we call *amortized closed-form efficiency*. Our basic idea is to address computations $\mathsf{Comp}(R_1, \ldots, R_n, \boldsymbol{z})$ of the above form, but then consider the case in which all values R_i are generated as $\mathsf{F}_K(\Delta, \tau_i)$. Basically, we interpret the PRF inputs L_i as pairs of values (Δ, τ_i), all sharing the same Δ component. Then, we informally say that F satisfies amortized closed-form efficiency if it is possible to compute ℓ computations $\{\mathsf{Comp}(\mathsf{F}_K(\Delta_j, \tau_1), \ldots, \mathsf{F}_K(\Delta_j, \tau_n), \boldsymbol{z})\}_{j=1}^{\ell}$ in time strictly less than $\ell \cdot t$. More detailed definitions follow.

A PRF consists of two algorithms $(\mathsf{KG}, \mathsf{F})$ such that (1) the key generation 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 (2) 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 $(\mathsf{KG}, \mathsf{F})$ 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, $(K, \mathsf{pp}) \leftarrow_{\mathcal{R}} \mathsf{KG}(1^\lambda)$, and $\Phi : \mathcal{X} \to \mathcal{R}$ is a random function.

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

Definition 3 (Amortized Closed-Form Efficiency). *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* $(\mathsf{KG}, \mathsf{F})$ *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* $o(t)$*.*

We remark two important facts on our definition. First, the computation of $\omega \leftarrow \mathsf{CFEval}^{\mathsf{off}}_{\mathsf{Comp},\boldsymbol{\tau}}(K, \boldsymbol{z})$ does *not* depend on Δ, which means that the same value ω can be re-used in $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{Comp},\Delta}(K, \omega)$ to compute $\mathsf{Comp}(\mathsf{F}_K(\Delta, \tau_1), \ldots, \mathsf{F}_K(\Delta, \tau_n), \boldsymbol{z})$ for many different Δ. Second, the efficiency property puts a restriction only on the running time of $\mathsf{CFEval}^{\mathsf{on}}$. This is related to the previous remark, and it captures the idea of achieving efficiency in an *amortized* sense when considering many evaluations of $\mathsf{Comp}(\mathsf{F}_K(\Delta, \tau_1), \ldots, \mathsf{F}_K(\Delta, \tau_n), \boldsymbol{z})$, each with a different data set identifier Δ. More concretely, this means that one can precompute ω once, and then use it to run $\mathsf{CFEval}^{\mathsf{on}}$ as many times as he needs, almost for free.

It is worth noting that the structure of Comp may enforce some constraints on the range $\mathcal{R}$ of the PRF, and that due to the pseudorandomness property, the output distribution of $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{Comp},\Delta}(K, \mathsf{CFEval}^{\mathsf{off}}_{\mathsf{Comp},\boldsymbol{\tau}}(K, \boldsymbol{z}))$ (over the random choice of K) is computationally indistinguishable from the output distribution of $\mathsf{Comp}(R_1, \ldots, R_n, \boldsymbol{z})$ (over the random choices of the $R_i \in \mathcal{R}$).

4.3 A PRF with Amortized Closed-Form Efficiency for GroupEval

We propose an efficient construction of a pseudorandom function which satisfies amortized closed-form efficiency for the algorithm $\mathsf{GroupEval}$, given in Section 4.1.

Our PRF construction uses two generic pseudorandom functions which map binary strings to integers in $\mathbb{Z}_p$ (where p is a sufficiently large prime number), together with a weak PRF whose security relies on the Decision Linear assumption, first introduced by Boneh, Boyen, and Shacham [13] and recalled below:

Definition 4 (Decision Linear [13]). *Let $\mathcal{G}$ be a bilinear group generator, and let* $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow_{\mathcal{R}} \mathcal{G}(1^\lambda)$. *Let $g_0, g_1, g_2 \leftarrow_{\mathcal{R}} \mathbb{G}$, and $r_0, r_1, r_2 \leftarrow_{\mathcal{R}} \mathbb{Z}_p$ be chosen uniformly at random. We define the advantage of an adversary $\mathcal{A}$ in solving the* Decision Linear *problem as*

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

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

Also, in our proof we will use the following useful Lemma (Lemma 7 in [37]) which basically shows that the Decision Linear problem is random self-reducible[6]:

Lemma 1 ([37]). *Given $g_0, g_1, g_2, g_1^{r_1}, g_2^{r_2}, g_0^{r_0} \in \mathbb{G}$, one can generate $g_1^{r'_1}, g_2^{r'_2}, g_0^{r'_0}$ such that: (1) r'_1, r'_2 are uniformly random in $\mathbb{Z}_p$, and (2) $r'_0 = r'_1 + r'_2$ if $r_0 = r_1 + r_2$, or r'_0 is uniformly random otherwise.*

Our pseudorandom function. Here we describe our PRF with amortized closed-form efficiency:

$\mathsf{KG}(1^\lambda)$**.** Let $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ be the description of bilinear groups $\mathbb{G}$ and $\mathbb{G}_T$ having the same prime order $p > 2^\lambda$ and such that $g \in \mathbb{G}$ is a generator and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an efficiently computable bilinear map. The key generation chooses two seeds K_1, K_2 for a family of PRFs $\mathsf{F}'_{K_{1,2}} : \{0,1\}^* \to \mathbb{Z}_p^2$. Finally, it outputs $K = (\mathsf{bgpp}, K_1, K_2)$ and $\mathsf{pp} = \mathsf{bgpp}$. The parameters define a function F with domain $\mathcal{X} = \{0,1\}^* \times \{0,1\}^*$ and range $\mathbb{G}$, as described below.

$\mathsf{F}_K(x)$**.** Let $x = (\Delta, \tau) \in \mathcal{X}$ be the input value. To compute the corresponding output $R \in \mathbb{G}$, the algorithm generates values $(u, v) \leftarrow \mathsf{F}'_{K_1}(\tau)$ and $(a, b) \leftarrow \mathsf{F}'_{K_2}(\Delta)$, and then outputs $R = g^{ua+vb}$.

We first show that the above function is pseudorandom, and then we will show that it admits amortized closed-form efficiency for $\mathsf{GroupEval}$.

Theorem 2. *If F' is a pseudorandom function and the Decision Linear assumption holds for $\mathcal{G}$, then the function $(\mathsf{KG}, \mathsf{F})$ described above is a pseudorandom function.*

Proof. The proof follows by a standard hybrid argument based on the following games.

Game 0: this is the pseudorandomness game for the function F.

Game 1: this is Game 0 where the function F'_{K_1} is replaced by a random function $\Phi_1 : \{0,1\}^* \to \mathbb{Z}_p^2$. It is easy to see that Game 1 is computationally indistinguishable from Game 0 by the security of the pseudorandom function F'.

Game 2: this is Game 1 where the function F'_{K_2} is replaced by a random function $\Phi_2 : \{0,1\}^* \to \mathbb{Z}_p^2$. Similarly to the previous case, one can easily argue that Game 2 is computationally indistinguishable from Game 1 by the security of the pseudorandom function F'.

Game (3, j): informally, for $j = 0, \ldots, Q_\Delta$, Game $(3, j)$ is a modification of Game 2 in which the queries (Δ, τ), where Δ is among the first j distinct values $\Delta_1, \ldots, \Delta_j$ queried by $\mathcal{A}$, are answered with randomly chosen outputs. More formally, let Q_Δ be the number of distinct Δ's queried by the adversary $\mathcal{A}$ during the experiment. If $S = \{\Delta_1, \ldots, \Delta_{Q_\Delta}\}$ is the ordered set of all such values queried by $\mathcal{A}$, then, for $0 \le j \le Q_\Delta$, we define the following partitioning sets of

[6] Lewko and Waters [37] state this Lemma for the k-Linear problem. We only recall the version for $k = 2$.

S: $S_{\leq j} = \{\Delta_i \in S : i \leq j\}$ and $S_{>j} = \{\Delta_i \in S : i > j\}$. So, we define Game (3, j) as the game which is the same as Game 2, except that queries (Δ, τ) such that $\Delta \in S_{\leq j}$ are answered with a value $R \leftarrow_{\mathcal{R}} \mathbb{G}$ chosen uniformly at random, whereas queries (Δ, τ) such that $\Delta \in S_{>j}$ are answered with $R = g^{ua+vb}$ where $(u, v) \leftarrow \Phi_1(\tau)$ and $(a, b) \leftarrow \Phi_2(\Delta)$.

As one can notice, Game (3,0) is identical to Game 2, while Game $3,Q_\Delta$ is the game in which all queries are answered with freshly random values in $\mathbb{G}$, i.e., it is like if $\mathcal{A}$ is given oracle access to a truly random function from $\mathcal{X}$ to $\mathbb{G}$.

Therefore, in order to complete the proof we claim that, for every $1 \leq j \leq Q_\Delta$, Game $(3, j-1)$ is computationally indistinguishable from Game $(3, j)$ under the assumption that Decision Linear holds for $\mathcal{G}$. This is obtained by proving the following Lemma:

Lemma 2. *For $1 \leq j \leq Q_\Delta$, let $G_{3,j}$ be the event that Game $(3, j)$, run with adversary $\mathcal{A}$, outputs 1. If the Decision Linear assumption holds for $\mathcal{G}$, then $|\Pr[G_{3,j-1}] - \Pr[G_{3,j}]|$ is negligible.*

The key tool of our proof is the following Lemma which essentially shows that the function $f_{a,b}(U, V) = (U^a V^b)$ is a weak pseudorandom function under the Decision Linear assumption.

Lemma 3. *If the Decision Linear assumption holds for $\mathcal{G}$ then the function $f_{a,b}(U, V) = (U^a V^b)$, where $a, b \leftarrow_{\mathcal{R}} \mathbb{Z}_p$ are randomly chosen, is a weak pseudorandom function.*

Proof. First, notice that given a tuple $(g_0, g_1, g_2, g_1^{r_1}, g_2^{r_2}, g_0^{r_0})$ we can rename values as $U = g_1^{r_1}, V = g_2^{r_2}, Z = g_0^{r_0}$. Next, we observe that for a fixed g_0, given two random values $g_1, g_2 \in \mathbb{G}$ there exist two values a, b (uniformly distributed in $\mathbb{Z}_p$) such that $g_0 = g_1^a$ and $g_0 = g_2^b$.

So, given such renaming of variables, we can reduce the security of $f_{a,b}(\cdot, \cdot)$ to Decision Linear by observing that by Lemma 1 we can create polynomially-many triples (U_i, V_i, Z_i) such that Z_i has the desired form that it is either $f_{a,b}(U_i, V_i)$ or uniformly random. □

Proof (Lemma 2). Given the result of Lemma 3, we are now ready to prove Lemma 2. To this end, we show that any PPT adversary $\mathcal{A}$ who has non-negligible probability in distinguishing between Game $(3, j-1)$ and Game $(3, j)$ can be used to build a PPT algorithm $\mathcal{B}$ who breaks the security of the weak PRF $f_{a,b}(U, V) = U^a V^b$, thus contradicting Lemma 3.

$\mathcal{B}$ receives as input a bilinear groups description $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ and gets access to an oracle $\mathcal{O}$ that upon each query (with no input) it outputs a triple (U, V, Z). Recall that if $\mathcal{O} = \mathcal{O}_f$, then $Z = U^a V^b$ where (a, b) is the secret seed of the weak PRF f. Otherwise, if $\mathcal{O} = \mathcal{O}_R$, then Z is randomly chosen in $\mathbb{G}$. In both cases, U and V are randomly chosen at every new query.

$\mathcal{B}$ runs the simulation for $\mathcal{A}$ as follows.

First, assume that Q_τ is an upper bound on the number of distinct τ's queried by $\mathcal{A}$. $\mathcal{B}$ queries the $\mathcal{O}$ oracle Q_τ times in order to get Q_τ triples $\{(U_i, V_i, Z_i)\}_{i=1}^{Q_\tau}$. Moreover, for $k = j+1, \ldots, Q_\Delta$, $\mathcal{B}$ chooses $(a_k, b_k) \leftarrow_{\mathcal{R}} \mathbb{Z}_p$ at random.

Notice that all this preparation is made at the beginning of the simulation only for ease of presentation. Both the queries to $\mathcal{O}$ and the generation of (a_k, b_k) could be done during the experiment without explicitly knowing the bounds Q_Δ, Q_τ.

Let (Δ, τ) be a query from $\mathcal{A}$, and assume that $(\Delta, \tau) = (\Delta_k, \tau_i)$, for $1 \leq k \leq Q_\Delta$ and $1 \leq i \leq Q_\tau$. $\mathcal{B}$ answers (Δ_k, τ_i) as follows.

- If $k \leq j-1$, then $\mathcal{B}$ chooses $R \leftarrow_{\mathcal{R}} \mathbb{G}$ uniformly at random and returns R.
- If $k > j$, then $\mathcal{B}$ returns $R = U_i^{a_k} \cdot V_i^{b_k}$.

– If $k = j$, then $\mathcal{B}$ returns $R = Z_i$.

Basically, the simulator is implicitly setting $(a_j, b_j) = (a, b)$ where (a, b) is the secret seed of the weak PRF f. Finally, if $\mathcal{A}$ outputs b, then $\mathcal{B}$ outputs the same value b.

It is not hard to see that the simulation is perfect. Precisely, in the case when $\mathcal{B}$ is given access to the weak PRF, i.e., $Z_i = f_{a,b}(U_i, V_i)$, then $\mathcal{B}$ is simulating Game $(3, j-1)$. On the other hand, when $\mathcal{B}$ gets access to a random function, i.e., Z_i is random and independent of U_i, V_i, then $\mathcal{B}$ simulates the view of Game $(3, j)$. Therefore, we have that $\Pr[\mathcal{B}^{\mathcal{O}_f} = 1] = \Pr[G_{3,j-1}]$ and $\Pr[\mathcal{B}^{\mathcal{O}_R} = 1] = \Pr[G_{3,j}]$. Thus, it holds:

$$|\Pr[\mathcal{B}^{\mathcal{O}_f} = 1] - \Pr[\mathcal{B}^{\mathcal{O}_R} = 1]| = |\Pr[G_{3,j-1}] - \Pr[G_{3,j}]|$$

which concludes the proof of Lemma 2. □

Amortized Closed-Form Efficiency. Here we show that the pseudorandom function described before satisfies amortized closed-form efficiency for $(\mathsf{GroupEval}, \mathbf{L})$. Recall that $\mathsf{GroupEval}$ is the algorithm which takes as input n random values $R_1, \ldots, R_n \in \mathbb{G}$ and the description of an arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$, and it returns a value $W \in \mathbb{G}_T$, where $\mathbf{L}$ is a vector $(\mathsf{L}_1, \ldots, \mathsf{L}_n)$ such that $\mathsf{L}_i = (\Delta, \tau_i) \in \mathcal{X}$. Below we describe the algorithms $\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\boldsymbol{\tau}}$ and $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}$:

$\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\boldsymbol{\tau}}(K, f)$**.** Let $K = (\mathsf{bgpp}, K_1, K_2)$ be a secret key as generated by $\mathsf{KG}(1^\lambda)$. For $i = 1$ to n, compute $(u_i, v_i) \leftarrow \mathsf{F}'_{K_1}(\tau_i)$, and set $\boldsymbol{\rho}_i = (0, u_i, v_i)$: $\boldsymbol{\rho}_i$ are essentially the coefficients of a degree-1 polynomial $\rho_i(z_1, z_2)$ in two (unknown) variables z_1, z_2.
Next, run $\boldsymbol{\rho} \leftarrow \mathsf{PolyEval}(2, f, \rho_1, \ldots, \rho_n)$ to compute the coefficients $\boldsymbol{\rho}$ of a polynomial $\rho(z_1, z_2)$ such that $\forall z_1, z_2 \in \mathbb{Z}_p$ it holds $\rho(z_1, z_2) = f(\rho_1(z_1, z_2), \ldots, \rho_n(z_1, z_2))$.
Finally, output $\omega_f = \boldsymbol{\rho}$.

$\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}(K, \omega_f)$**.** Let $K = (\mathsf{bgpp}, K_1, K_2)$ be a secret key and let $\omega_f = \boldsymbol{\rho}$ be as computed by the previous algorithm. The online evaluation algorithm first generates $(a, b) \leftarrow \mathsf{F}'_{K_2}(\Delta)$, and then it uses the coefficients $\boldsymbol{\rho}$ to compute $w = \rho(a, b)$, and it finally outputs $W = e(g, g)^w$.

Theorem 3. *Let* $\mathbf{L} = (\mathsf{L}_1, \ldots, \mathsf{L}_n)$ *be such that* $\mathsf{L}_i = (\Delta, \tau_i) \in \mathcal{X}$*, let* $\mathsf{GroupEval}$ *be the algorithm described in Section 4.1, and let* t *be the running time of* $\mathsf{GroupEval}$*. Then the pseudorandom function* $(\mathsf{KG}, \mathsf{F})$*, extended with the algorithms* $\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\boldsymbol{\tau}}$ *and* $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}$ *described above, satisfies amortized closed-form efficiency for* $(\mathsf{GroupEval}, \mathbf{L})$ *according to Definition 3, having* $\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\boldsymbol{\tau}}$ *run in time* $O(t)$ *and* $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}$ *run in time* $O(1)$*.*

Proof. To prove the theorem we show that our algorithms satisfy both the correctness and efficiency properties of Definition 3. Let K be a secret key as generated by $\mathsf{KG}(1^\lambda)$, and let $\mathbf{L}$ be any vector of n values $(\mathsf{L}_1, \ldots, \mathsf{L}_n)$ such that $\mathsf{L}_i = (\Delta, \tau_i) \in \mathcal{X}$ for arbitrary binary strings $\Delta, \tau_1, \ldots, \tau_n \in \{0,1\}^*$. Let $\omega_f = \boldsymbol{\rho}$ be the output of $\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\boldsymbol{\tau}}(K, f)$. Then, we have:

$$\begin{aligned}
\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}(K, \omega_f) &= W \\
&\overset{\mathsf{CFEval}^{\mathsf{on}}}{=} e(g,g)^{\rho(a,b)} \\
&\overset{\mathsf{PolyEval}}{=} e(g,g)^{f(\rho_1(a,b),\ldots,\rho_n(a,b))} \\
&\overset{\mathsf{CFEval}^{\mathsf{off}}}{=} e(g,g)^{f(u_1 a + v_1 b, \ldots, u_n a + v_n b)} \\
&= \mathsf{GroupEval}(f, \mathsf{F}_K(\Delta, \tau_1), \ldots, \mathsf{F}_K(\Delta, \tau_n))
\end{aligned}$$

where the last equality holds by the correctness of $\mathsf{GroupEval}$ (Theorem 1).

To see the efficiency property, we first observe that the running time of $\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\boldsymbol{\tau}}(K, f)$ is essentially dominated by the computation of $\boldsymbol{\rho}$ using $\mathsf{PolyEval}(2, f, \rho_1, \ldots, \rho_n)$. Interestingly, due to the bound $deg(f) \leq 2$ and due to having only $m = 2$ variables, the polynomial $\boldsymbol{\rho}$ can be computed at roughly the same cost of running f, which is the cost of $\mathsf{GroupEval}$, i.e., $O(t)$. Regarding the online algorithm $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}(K, \omega_f)$, its complexity depends on the size of $\boldsymbol{\rho}$, hence on the number of coefficients of a two-variate polynomial whose degree is the same as the degree of f. In general, for f of degree d, this would be $|\boldsymbol{\rho}| = \binom{d+2}{d}$. Considering our specific case of $\mathsf{GroupEval}$ which evaluates arithmetic circuits of degree at most 2, and by observing that the degree-0 coefficient is always 0, we obtain a vector $\boldsymbol{\rho}$ which can be represented with 5 elements of $\mathbb{Z}_p$, from which we have $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}(K, \omega_f)$ running in time $O(1)$. □

5 Homomorphic Message Authenticators with Efficient Verification

In this section, we describe our construction of homomorphic MACs with efficient verification for multi-labeled programs as introduced in Section 3.3. In particular, the following theorem summarizes the main result of this work which is obtained by combining the $\mathsf{EVH{-}MAC}$ construction (Section 5.1) and our concrete instantiation of the PRF with amortized closed-form efficiency based on the Decision Linear assumption (Section 4.3).

Theorem 4. *If the Decision Linear assumption holds, then* $\mathsf{EVH{-}MAC}$ *is a secure homomorphic message authenticator which supports evaluations of any arithmetic circuit f of degree at most 2, and achieves efficient verification, i.e.,* $\mathsf{EVH{-}MAC}$ *has amortized efficiency in which the offline verification* $\mathsf{VerPrep}$ *takes time $O(|f|)$, and the online verification* EffVer *takes time $O(1)$.*

We proceed by detailing our construction (Section 5.1), showing its correctness (Section 5.2), and then proving its security (Section 5.3), and finally discussing its efficiency (Section 5.4).

5.1 Construction

Our construction works for circuits whose additive gates do not get inputs labeled by constants. As mentioned in [17], this can be done without loss of generality as one can use an equivalent circuit with a special variable/label for the constant 1 and publish the MAC of 1. Our scheme $\mathsf{EVH{-}MAC}$ is defined as follows:

$\mathsf{KeyGen}(1^\lambda)$**.** Run $\mathsf{bgpp} \leftarrow_{\mathcal{R}} \mathcal{G}(1^\lambda)$ to generate the description of bilinear groups. Let $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ as defined above. Let the message space $\mathcal{M}$ be $\mathbb{Z}_p$. Choose a random value $\alpha \leftarrow_{\mathcal{R}} \mathbb{Z}_p$, and run $(K, \mathsf{pp}) \leftarrow_{\mathcal{R}} \mathsf{KG}(1^\lambda)$ to obtain the seed K of a pseudorandom function $\mathsf{F}_K : \{0,1\}^* \times \{0,1\}^* \to \mathbb{G}$. Output the secret key $\mathsf{sk} = (\mathsf{bgpp}, \mathsf{pp}, K, \alpha)$, and the evaluation key $\mathsf{ek} = (\mathsf{bgpp}, \mathsf{pp})$.

$\mathsf{Auth}(\mathsf{sk}, \mathsf{L}, m)$**.** To authenticate a message $m \in \mathbb{Z}_p$ with multi-label $\mathsf{L} = (\Delta, \tau)$ where $\Delta \in \{0,1\}^\lambda$ is the identifier of a data set and $\tau \in \{0,1\}^\lambda$ is an input identifier, proceed as follows.

First, compute $R \leftarrow \mathsf{F}_K(\Delta, \tau)$ and then compute values $(y_0, Y_1) \in \mathbb{Z}_p \times \mathbb{G}$ by setting: $y_0 = m$ and $Y_1 = (R \cdot g^{-m})^{1/\alpha}$. Finally, output the tag $\sigma = (y_0, Y_1)$.

If we let $y_1 \in \mathbb{Z}_p$ be the (unique) value such that $Y_1 = g^{y_1}$, then (y_0, y_1) are basically the coefficients of a degree-1 polynomial $y(x)$ that evaluates to m on the point 0 (i.e., $y(0) = m$) and it evaluates to $r = \phi_g^{-1}(R)$ on a hidden random point α (i.e., $y(\alpha) = r$).

$\mathsf{Eval}(\mathsf{ek}, f, \boldsymbol{\sigma})$**.** The homomorphic evaluation algorithm takes as input the evaluation key $\mathsf{ek} = (\mathsf{bgpp}, \mathsf{pp})$, an arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$, and a vector $\boldsymbol{\sigma}$ of tags $(\sigma_1, \ldots, \sigma_n)$.

Eval proceeds gate-by-gate as follows. At every gate f_g, given two tags σ_1, σ_2 (or a tag σ_1 and a constant $c \in \mathbb{Z}_p$), it runs the algorithm $\sigma \leftarrow \mathsf{GateEval}(\mathsf{ek}, f_g, \sigma_1, \sigma_2)$ described below that returns a new tag σ, which is in turn passed on as input to the next gate in the circuit. When the computation reaches the last gate of the circuit f, Eval outputs the tag vector σ obtained by running GateEval on such last gate.

To complete the description of Eval we thus describe the subroutine GateEval:

– $\mathsf{GateEval}(\mathsf{ek}, f_g, \sigma^{(1)}, \sigma^{(2)})$. Let $\sigma^{(i)} = (y_0^{(i)}, Y_1^{(i)}, \hat{Y}_2^{(i)}) \in \mathbb{Z}_p \times \mathbb{G} \times \mathbb{G}_T$ for $i = 1, 2$ (see below for the special case when one of the two inputs is a constant $c \in \mathbb{Z}_p$). For ease of description, whenever $\hat{Y}_2^{(i)}$ is not defined, we assume $\hat{Y}_2^{(i)} = 1 \in \mathbb{G}_T$.

- **Addition.** If $f_g = f_+$, then compute $(y_0, Y_1, \hat{Y}_2)$ as follows:

$$y_0 = {y_0}^{(1)} + {y_0}^{(2)}, \quad Y_1 = Y_1^{(1)} \cdot Y_1^{(2)}, \quad \hat{Y}_2 = \hat{Y}_2^{(1)} \cdot \hat{Y}_2^{(2)}.$$

- **Multiplication.** If $f_g = f_\times$, then compute $(y_0, Y_1, \hat{Y}_2)$ as follows:

$$y_0 = {y_0}^{(1)} \cdot {y_0}^{(2)}, \quad Y_1 = (Y_1^{(1)})^{y_0^{(2)}} \cdot (Y_1^{(2)})^{y_0^{(1)}}, \quad \hat{Y}_2 = e(Y_1^{(1)}, Y_1^{(2)}).$$

  Because of our assumption that $deg(f) \leq 2$, we can assume that $\sigma^{(i)} = (y_0^{(i)}, Y_1^{(i)}) \in \mathbb{Z}_p \times \mathbb{G}$ for both $i = 1, 2$.

- **Multiplication with constant.** If $f_g = f_\times$ and one of the two inputs, say σ_2, is a constant $c \in \mathbb{Z}_p$, then compute $(y_0, Y_1, \hat{Y}_2)$ as follows:

$$y_0 = c \cdot {y_0}^{(1)}, \quad Y_1 = (Y_1^{(1)})^c, \quad \hat{Y}_2 = (Y_2^{(1)})^c.$$

Return $\sigma = (y_0, Y_1, \hat{Y}_2)$.

$\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_\Delta, m, \sigma)$**.** Let $\mathsf{sk} = (\mathsf{bgpp}, \mathsf{pp}, K, \alpha)$ be a secret key. Let $\mathcal{P}_\Delta = (\mathcal{P}, \Delta)$ be a multi-labeled program for $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and data set Δ. Let $m \in \mathbb{Z}_p$ be the result to be verified, and let $\sigma = (y_0, Y_1, \hat{Y}_2)$ be a tag. The verification proceeds as follows. For $i = 1$ to n, compute $R_i \leftarrow \mathsf{F}_K(\Delta, \tau_i)$. Then run $W \leftarrow \mathsf{GroupEval}(f, R_1, \ldots, R_n) \in \mathbb{G}_T$, as described in Section 4.1. Finally, check the following equations:

$$m = y_0 \tag{4}$$

$$W = e(g, g)^{y_0} \cdot e(Y_1, g)^\alpha \cdot (\hat{Y}_2)^{\alpha^2} \tag{5}$$

If both checks are satisfied, then output 1, and 0 otherwise.

Finally, to complete the description of EVH−MAC we give the algorithms for efficient verification:

$\mathsf{VerPrep}(\mathsf{sk}, \mathcal{P})$**.** Let $\mathcal{P} = (f, \boldsymbol{\tau})$ be a labeled program where $f \in \mathbb{Z}_p^n \to \mathbb{Z}_p$ is an arithmetic circuit and $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_n)$ is a vector of input identifiers for f. The algorithm computes concise verification information $\mathsf{VK}_\mathcal{P} = \omega$ where ω is obtained by using the offline closed-form efficient algorithm of F for GroupEval, i.e., $\omega \leftarrow \mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval}, \boldsymbol{\tau}}(K, f)$.

$\mathsf{EffVer}(\mathsf{sk}, \mathsf{VK}_\mathcal{P}, \Delta, m, \sigma)$**.** Let $\mathsf{sk} = (\mathsf{bgpp}, \mathsf{pp}, K, \alpha)$ be a secret key. Let $\mathsf{VK}_\mathcal{P} = \omega$ be the concise verification information for $\mathcal{P}$. Let $m \in \mathbb{Z}_p$ be the result to be verified and let $\sigma = (y_0, Y_1, \hat{Y}_2)$ be a tag. The online verification proceeds as follows. First, it runs the online closed-form efficient algorithm of F for GroupEval, in order to compute $W \leftarrow \mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval}, \Delta}(K, \omega)$. Finally, it runs the same checks (4) and (5) as in standard verification. If both checks are satisfied, then output 1. Otherwise output 0.

5.2 Proof of Correctness

In this section, we prove that EVH−MAC satisfies authentication and evaluation correctness.

Theorem 5. EVH−MAC *satisfies authentication correctness.*

Proof. Let $m \in \mathbb{Z}_p$ and $\mathsf{L} = (\Delta, \tau)$ be given. Let a correctly generated secret key $\mathsf{sk} = (\mathsf{bgpp}, \mathsf{pp}, K, \alpha)$ and a correctly generated evaluation key $\mathsf{ek} = (\mathsf{bgpp}, \mathsf{pp})$ with $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ be given. Let further $\sigma = (y_o, Y_1)$ be an authentication tag obtained from running $\mathsf{Auth}(\mathsf{sk}, \mathsf{L}, m)$. We show that $\mathsf{Ver}(\mathsf{sk}, \mathcal{I}_\mathsf{L}, m, \sigma) = 1$ with probability 1 for some identity program $\mathcal{I}_\mathsf{L}$ computing the identity function f_{id} for L. To this end, we verify the equations (4) and (5). For the first equation, it is obvious to see that indeed $m = y_0$ because of our definition of Auth. For the second equation, we know that $Y_1 \overset{\mathsf{Auth}}{=} (R \cdot g^{-m})^{1/\alpha}$ with $R = F_K(\Delta, \tau)$, and $\hat{Y}_2 = 1$.

$$
\begin{aligned}
& e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} \\
\overset{\mathsf{Auth}}{=} \; & e(g,g)^{m} \cdot e((R \cdot g^{-m})^{1/\alpha}, g)^{\alpha} \cdot 1 \\
= \; & e(g,g)^{m} \cdot e(R \cdot g^{-m}, g) \\
= \; & e(g,g)^{m} \cdot e(R, g) \cdot e(g^{-m}, g) \\
= \; & e(g,g)^{m} \cdot e(R, g) \cdot e(g, g)^{-m} \\
= \; & e(R, g) \\
\overset{\mathsf{GroupEval}}{=} \; & \mathsf{GroupEval}(f_{id}, R) = W
\end{aligned}
$$

For the last equality, the verification is successful only if $\mathsf{GroupEval}(f_{id}, R) = e(R, g)$, which follows immediately from the correctness of $\mathsf{GroupEval}$ (Theorem 1). □

Theorem 6. EVH−MAC *satisfies evaluation correctness.*

Proof. Let a valid pair of keys $(\mathsf{sk}, \mathsf{ek}) \leftarrow_{\mathcal{R}} \mathsf{KeyGen}(1^\lambda)$ be given. Let $f' : \mathcal{M}^n \to \mathcal{M}$ and a set of message/program/tag triples $\{(m_i, \mathcal{P}_{\Delta,i}, \sigma_i)\}_{i=1}^n$ be given, such that all $\mathcal{P}_{\Delta,i} = (\mathcal{P}_i, \Delta)$ share the same data set identifier Δ and $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_{\Delta,i}, m_i, \sigma_i) = 1$. Let $m^* = f'(m_1, \ldots, m_n)$, let $\mathcal{P}^*$ be the composed program $f'(\mathcal{P}_1, \ldots, \mathcal{P}_n)$, and let $\sigma^* = \mathsf{Eval}(\mathsf{ek}, f', (\sigma_1, \ldots, \sigma_n))$. We show that $\mathsf{Ver}(sk, \mathcal{P}^*_\Delta, m^*, \sigma^*) = 1$ holds with probability 1.

For $i = 1$ to n, let W_i be the values obtained by computing $\mathsf{GroupEval}$ in the runs of $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_{\Delta,i}, m_i, \sigma_i)$, and let $\sigma_i = ({y_0}^{(i)}, Y_1^{(i)}, \hat{Y}_2^{(i)})$. By our inductive hypothesis, i.e., $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_{\Delta,i}, m_i, \sigma_i) = 1$, we know that both the following equations

$$m_i = {y_0}^{(i)} \tag{6}$$

$$W_i = e(g,g)^{{y_0}^{(i)}} \cdot e(Y_1^{(i)}, g)^{\alpha} \cdot (\hat{Y}_2^{(i)})^{\alpha^2} \tag{7}$$

are satisfied. For all $i = 1, \ldots, n$, consider the unique values $y_1^{(i)} = \phi_g^{-1}(Y_1^{(i)})$, $y_2^{(i)} = \phi_{g_T}^{-1}(\hat{Y}_2^{(i)})$, and $w_i = \phi_{g_T}^{-1}(W_i)$, and let us compactly denote by $y^{(i)}$ the degree-2 polynomial with coefficients $y_0^{(i)}, y_1^{(i)}, y_2^{(i)} \in \mathbb{Z}_p$. Equations (6) and (7) imply that $y^{(i)}(0) = m_i$ and $y^{(i)}(\alpha) = w_i$, $\forall i = 1, \ldots, n$.

Similarly, for the tag $\sigma^* = ({y_0}^*, Y_1^*, (\hat{Y}_2)^*)$ we let y^* be the degree-2 polynomial with coefficients $y_0^*, y_1^*, y_2^* \in \mathbb{Z}_p$ uniquely defined as above. Also, let W^* be the value obtained by running $\mathsf{GroupEval}$ in $\mathsf{Ver}(sk, \mathcal{P}^*_\Delta, m^*, \sigma^*) = 1$.

To prove this theorem we will show that $y^*(0) = m^*$ and $e(g,g)^{y^*(\alpha)} = W^*$ are satisfied. To this end, we first prove the following claim. Intuitively, the claim shows that our algorithm $\mathsf{GateEval}$ is computing $\mathsf{PolyEval}$ "in the exponent", over the input polynomials $\{y^{(i)}\}_{i=1}^n$ encoded in the groups $\mathbb{G}, \mathbb{G}_T$.

Claim 1 *Let f_g be a gate of an arithmetic circuit, let $\sigma^{(1)} = (y_0^{(1)}, Y_1^{(1)}, \hat{Y}_2^{(1)})$ and $\sigma^{(2)} = (y_0^{(2)}, Y_1^{(2)}, \hat{Y}_2^{(2)})$ be any two tags in $\mathbb{Z}_p \times \mathbb{G} \times \mathbb{G}_T$, and let $\sigma = (y_0, Y_1, \hat{Y}_2)$ be the output of $\mathsf{GateEval}(\mathsf{ek}, f_g, \sigma^{(1)}, \sigma^{(2)})$. If we define the three polynomials $y^{(1)}, y^{(2)}$, y from the three tags $\sigma^{(1)}, \sigma^{(2)}$, σ, respectively, by using the homomorphisms ϕ_g and ϕ_{g_T}, then it holds $y = \mathsf{PolyEval}(1, f_g, y^{(1)}, y^{(2)})$.*

Proof. To prove the claim we consider the two cases in which f_g is either an addition or a multiplication gate.

For an addition gate f_+, we have

$$\begin{aligned}(y_0, Y_1, \hat{Y}_2) &\overset{\mathsf{GateEval}}{=} (y_0{}^{(1)} + y_0{}^{(2)}, Y_1^{(1)} \cdot Y_1^{(2)}, \hat{Y}_2^{(1)} \cdot \hat{Y}_2^{(2)}) \\ &= (y_0{}^{(1)} + y_0{}^{(2)}, g^{y_1^{(1)} + y_1^{(2)}}, e(g,g)^{y_2^{(1)} + y_2^{(2)}})\end{aligned}$$

Hence, we clearly have that $y = \mathsf{PolyEval}(1, f_+, y^{(1)}, y^{(2)})$.

For a multiplication gate $f_\times$, we have

$$\begin{aligned}(y_0, Y_1, \hat{Y}_2) &\overset{\mathsf{GateEval}}{=} (y_0{}^{(1)} y_0{}^{(2)}, (Y_1^{(1)})^{y_0{}^{(2)}} \cdot (Y_1^{(2)})^{y_0{}^{(1)}}, e(Y_1^{(1)}, Y_1^{(2)})) \\ &= (y_0{}^{(1)} y_0{}^{(2)}, g^{y_1^{(1)} y_0{}^{(2)} + y_1^{(2)} y_0{}^{(1)}}, e(g,g)^{y_1^{(1)} y_1^{(2)}})\end{aligned}$$

Hence, we clearly have that $y = \mathsf{PolyEval}(1, f_\times, y^{(1)}, y^{(2)})$. □

By inductively extending the result of Claim 1 over the entire circuit f', we obtain that $y^* = \mathsf{PolyEval}(1, f', y^{(1)}, \ldots, y^{(n)})$. So, by relying on the homomorphic property of $\mathsf{PolyEval}$ and on our inductive hypothesis we have that the first equation of Ver is satisfied, i.e.:

$$\begin{aligned}& y^*(0) \\ \overset{\mathsf{PolyEval}}{=}\ & f'(y^{(1)}(0), \ldots, y^{(n)}(0)) \\ \overset{(6)}{=}\ & f'(m_1, \ldots, m_n) = m^*.\end{aligned}$$

To see that the second equation is satisfied as well, we observe that:

$$\begin{aligned}& e(g,g)^{y_0{}^*} \cdot e(Y_1^*, g)^\alpha \cdot (\hat{Y}_2^*)^{\alpha^2} \\ =\ & e(g,g)^{y^*(\alpha)} \\ \overset{\mathsf{PolyEval}}{=}\ & e(g,g)^{f'(y^{(1)}(\alpha), \ldots, y^{(n)}(\alpha))} \\ \overset{(7)}{=}\ & e(g,g)^{f'(w_1, \ldots, w_n)} \\ =\ & W^*\end{aligned}$$

where the last equality follows from the composition property of circuits applied to $\mathsf{GroupEval}$.

Theorem 7. *If* F *has amortized closed-form efficiency for* $(\mathsf{GroupEval}, \mathbf{L})$*, then* $\mathsf{EVH{-}MAC}$ *satisfies efficient verification.*

Proof. According to Definition 2, we have to show that both properties of (1) correctness and (2) efficiency hold. Correctness simply follows from the fact that the function F satisfies closed-form efficiency for $(\mathsf{GroupEval}, \mathbf{L})$ according to Theorem 3. This indeed means that by computing $W \leftarrow \mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}(K, \omega)$ for $\omega \leftarrow \mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\tau}(K, f)$, one obtains the same value W as obtained by computing $\mathsf{GroupEval}(f, \mathsf{F}_K(\Delta, \tau_1), \ldots, \mathsf{F}_K(\Delta, \tau_n))$. Hence, it is clear that the combination of the algorithms $\mathsf{VerPrep}$ and EffVer computes the same code of Ver. The amortized efficiency property is achieved by EffVer in executing $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}$ once, and performing a constant number of multiplications and exponentiations. □

We notice that by the correctness of efficient verification, it also follows that $\mathsf{EVH{-}MAC}$ satisfies authentication and evaluation correctness w.r.t. the algorithm EffVer.

5.3 Proof of Security

The security of $\mathsf{EVH{-}MAC}$ is established by the following theorem.

Theorem 8. *Let* λ *be the security parameter,* F *be a pseudorandom function with security* ϵ_{F}*, and* $\mathcal{G}$ *be a bilinear group generator. Then, any PPT adversary* $\mathcal{A}$ *making* Q *verification queries has at most probability* $\Pr[\mathsf{HomUF{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}} = 1] \leq 2 \cdot \epsilon_{\mathsf{F}} + \frac{8Q}{p-2(Q-1)}$ *of breaking the security of* $\mathsf{EVH{-}MAC}$*.*

Proof. We have to show property (4) of Definition 1. In other words, we have to show that for every PPT adversary $\mathcal{A}$ the probability of winning the experiment $\mathsf{HomUF{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}$ is negligible. Using a hybrid argument, we transform the experiment $\mathsf{HomUF{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}$ with a number of games to an experiment with indistinguishable distribution. By $G_i(\mathcal{A})$ we denote the event that an adversary $\mathcal{A}$ wins in the experiment defined in Game i, hence that the challenger outputs 1.

Game 0 is the experiment $\mathsf{HomUF{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}$.

Game 1 is like Game 0, but using Proposition 1, we omit forgeries of Type 3. Notice that after such a change, now the challenger can efficiently distinguish between forgeries of Type 1 and Type 2.

Game 2 is like Game 1, but the PRF is replaced by a truly random function $\mathcal{R} : \{0,1\}^* \times \{0,1\}^* \rightarrow \mathbb{G}$. Hence, each $R \in \mathbb{G}$ is a truly random value.

Game 3 is like Game 2, but the verification equation is split into two checks: (1) if $m \neq y_0$, reply 0; (2) if $W = e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$, reply 1 to $\mathcal{A}$. The first check is performed as the very first step after receiving a verification query from $\mathcal{A}$. The position of the second check is unchanged. Moreover, we split verification after the first check in two cases: (Type 1) for queries in which no list T_{Δ} has been created and (Type 2) in which a list T_{Δ} has been created. Both subroutines perform exactly the same operation, i.e., computing $W \leftarrow \mathsf{GroupEval}(f, R_1, \ldots, R_n)$ and checking whether $W = e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$.

Game 4 is like Game 3, but verification queries $(\mathcal{P}_{\Delta}, m, \sigma)$ in which a list T_{Δ} exists are treated differently: for each τ_i such that $(\tau_i, \cdot) \notin T_{\Delta}$, compute a dummy tag $\tilde{\sigma}_i$ (step 2). For each τ_j such that $(\tau_j, \cdot) \in T_{\Delta}$, fetch the previously stored value $\tilde{\sigma}_j \leftarrow \Sigma[\Delta, \tau_j]$ (step 3). Evaluate

f on $\tilde{\boldsymbol{\sigma}} = (\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n)$ computing $(y_0', Y_1', \hat{Y}_2') \leftarrow \mathsf{Eval}(\mathsf{ek}, f, \tilde{\boldsymbol{\sigma}})$ (step 4). Next, check if $\sigma = (y_0, Y_1, \hat{Y}_2) = (y_0', Y_1', \hat{Y}_2')$ and accept (step 5). Otherwise, check if $e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} = e(g,g)^{y_0'} \cdot e(Y_1', g)^{\alpha} \cdot (\hat{Y}_2')^{\alpha^2}$ and accept (step 6). Otherwise, reject (as before).

Game 5 is like Game 4, but instead of replying 1 to the adversary, in two cases we reply 0 and set an (initially false) flag bad to true. These two cases are: (T1) for the empty list T_{Δ} whenever $W = e(g,g)^{y(\alpha)}$ holds, and (T2) when the list T_{Δ} is not empty whenever $e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} = e(g,g)^{y_0'} \cdot e(Y_1', g)^{\alpha} \cdot (\hat{Y}_2')^{\alpha^2}$ holds.

Claim 2 *The probability for adversary $\mathcal{A}$ of winning in Game 5 is zero.*

Proof. Winning the experiment means that the answer to the adversary for a verification/forgery query is 1 (accept) and that one of the two cases in the forgery check is satisfied.

The only case to return 1 to the adversary is in line 5 in the case of $\mathcal{P}$ being well-defined with respect to Δ. The necessary condition here is that $\sigma = \sigma'$, which means that the attacker provides a MAC σ that is equal to a honestly generated MAC σ'. Since the two MACs are equal, they both authenticate a unique message m', which, by the correctness of Eval, must be the same as the attacker message m. In particular, $m' = \mathsf{Eval}(\mathsf{ek}, f, \boldsymbol{\sigma}) = f(\{m_j\}_{(\tau_j, m_j) \in T_{\Delta}})$. Therefore, the forgery check 2 with $m \neq f(\{m_j\}_{(\tau_j, m_j) \in T_{\Delta}})$ is not true. And hence there is no forgery, and the output of the experiment is never 1. □

Next, we show that for all i the difference between Game i and Game $i+1$ is negligible. This finally yields that $\Pr[G_0(\mathcal{A})]$ is negligible, which concludes the proof of security for $\mathsf{EVH{-}MAC}$.

Game 0 and Game 1 differ only in the event that an adversary $\mathcal{A}$ wins in Game 0 with a Type 3 forgery, namely $|\Pr[G_0(\mathcal{A})] - \Pr[G_1(\mathcal{A})]| = \Pr[G_0(\mathcal{A}) \wedge T_{\mathcal{A},3}]$ where $T_{\mathcal{A},3}$ is the event that $\mathcal{A}$ wins by returning a forgery of Type 3. In order to get an upper bound on the probability of an adversary winning in Game 0 with a Type 3 forgery, we can use Proposition 1 which relates this probability with the probability of winning with a Type 2 forgery.

Claim 3 *If ϵ is an upper bound on the probability $\Pr[G_1(\mathcal{B})]$ for any adversary $\mathcal{B}$, then for all adversaries $\mathcal{A}$, we have $|\Pr[G_0(\mathcal{A})] - \Pr[G_1(\mathcal{A})]| \leq \epsilon + 2/p$.*

Proof. We first observe that the difference between the two games only depends on how forgeries of Type 3 are handled. More precisely, for all adversaries $\mathcal{A}$, we have $|\Pr[G_0(\mathcal{A})] - \Pr[G_1(\mathcal{A})]| = \Pr[G_0(\mathcal{A}) \wedge T_{\mathcal{A},3}]$, where $T_{\mathcal{A},3}$ is the event in which $\mathcal{A}$ uses a forgery of Type 3 to win. Proposition 1 yields that if for all $\mathcal{B}$, we have that $\Pr[G_1(\mathcal{B})]$ is negligible, then also for all $\mathcal{A}$, we have $\Pr[G_0(\mathcal{A}) \wedge T_{\mathcal{B},3}]$ is negligible. More precisely, $\Pr[G_0(\mathcal{A}) \wedge T_{\mathcal{B},3}] \leq \epsilon + d/p$, where ϵ is an upper bound on the probability $\Pr[G_1(\mathcal{B})]$. □

It remains hence to show that $\Pr[G_1(\mathcal{A})]$ is indeed negligible for any adversary $\mathcal{A}$. To this end, we give negligible bounds for the distance of any two consecutive games.

Claim 4 *If F is a pseudorandom function, then $\Pr[G_1(\mathcal{A})] - \Pr[G_2(\mathcal{A})] \leq \epsilon_{\mathsf{F}}$, where ϵ_{F} is the negligible advantage of an adversary in breaking the security of F.*

Proof. This proof can be easily obtained by reducing any adversary with non-negligible probability of distinguishing Game 1 and Game 2 into one that breaks the security of the pseudorandom function F. □

Claim 5 $\Pr[G_2(\mathcal{A})] \equiv \Pr[G_3(\mathcal{A})]$.

Proof. It is easy to see that all changes from Game 2 to Game 3 are only syntactical. Hence, all views and all probability distributions are the same in both games. □

Claim 6 $\Pr[G_3(\mathcal{A})] \equiv \Pr[G_4(\mathcal{A})]$.

Proof. The changes from Game 3 to Game 4 only affect queries from the adversary with well-defined programs. We hence assume that $\mathcal{P}$ is well-defined. In particular, all dummy tags $\tilde{\sigma}_i$ (if any) in Game 4 do not contribute in the evaluation of f using Eval.

We show that the output to the adversary is 1 in Game 4 if and only if the output is 1 in Game 3. To this end, assume the answer to $\mathcal{A}$ is 1 in Game 3. We hence know that $m = y_0$ and that $W_3 \leftarrow \mathsf{GroupEval}(f, R_1, \ldots, R_n)$ with $W_3 = e(g,g)^{y(\alpha)}$.

In Game 4, the evaluation of $\sigma' \leftarrow \mathsf{Eval}(\mathsf{ek}, f, \tilde{\boldsymbol{\sigma}})$ is based on $\tilde{\boldsymbol{\sigma}}$, which are all taken from the list of previously generated tags. This is hence the same as the generation of the values R_i in Game 3. By correctness of Eval, we know that $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}, y_0{}', \sigma') = 1$. In particular, we have that $W_4 \leftarrow \mathsf{GroupEval}(f, R_1, \ldots, R_n)$ with $W_4 = e(g,g)^{y'(\alpha)}$. Since the values R_i for the corresponding evaluations of $\mathsf{GroupEval}$ is the same in both games, we have that $W_3 = W_4$ for the corresponding games. This yields $e(g,g)^{y(\alpha)} = W_3 = W_4 = e(g,g)^{y'(\alpha)}$, which indeed gives $e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} = e(g,g)^{y_0{}'} \cdot e(Y_1', g)^{\alpha} \cdot (\hat{Y}_2')^{\alpha^2}$.

If $(y_0, Y_1, \hat{Y}_2) = (y_0{}', Y_1', \hat{Y}_2')$, then verification returns the correct result 1 and clearly, $e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} = e(g,g)^{y_0{}'} \cdot e(Y_1', g)^{\alpha} \cdot (\hat{Y}_2')^{\alpha^2}$ is also satisfied. □

Claim 7 $|\Pr[G_4(\mathcal{A})] - \Pr[G_5(\mathcal{A})]| \leq \Pr[\mathsf{Bad}]$, *where* Bad *is the event that* bad *is set to* true *in Game 5.*

Proof. Game 4 and Game 5 are identical unless bad is set to true in Game 5. More precisely, $\Pr[G_4(\mathcal{A})] = \Pr[G_5(\mathcal{A}) \wedge \neg \mathsf{Bad}]$, hence $|\Pr[G_4(\mathcal{A})] - \Pr[G_5(\mathcal{A})]| \leq \Pr[\mathsf{Bad}]$. □

To finalize the security proof, we are left with bounding the probability of the event Bad, i.e., the event in which bad is set to true in Game 5.

Claim 8 $\Pr[\mathsf{Bad}] \leq \frac{4Q}{p-2(Q-1)}$, *where p is the prime used in the construction, and Q is an upper bound on the number of verification queries made by an adversary.*

Proof. Let B_j be the event that bad was set from false to true in the j-th verification query. Let Q be the number of verification queries performed by an attacker. Then

$$\Pr[\mathsf{Bad}] = \Pr\left[\bigvee_{j=1}^{Q} B_j\right] \leq \sum_{j=1}^{Q} \Pr[B_j]$$

In the following, we estimate the probability $\Pr[B_j]$ taken over the random choices of α and all values R_i sampled by the challenger. We also take into account all possible values chosen by the adversary. From the definition of Game 5, there are only two cases in which B_j can occur:

Event B_j^1: $W = e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$ and C_j^1,
where C_j^1 is the event that no list T_{Δ} has been created.
Event B_j^2: $e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} = e(g,g)^{y_0{}'} \cdot e(Y_1', g)^{\alpha} \cdot (\hat{Y}_2')^{\alpha^2}$ and C_j^2,
where C_j^2 is the event that $\mathcal{P}$ is well-defined on T_{Δ}, and at least for one index i we have $y_i \neq y_i'$.

In the following we will often write $y(\alpha) = y'(\alpha)$ to stand for $e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} = e(g,g)^{y_0{}'} \cdot e(Y_1', g)^{\alpha} \cdot (\hat{Y}_2')^{\alpha^2}$. From the definition of B_j, we know that bad was not set to true in the previous $j-1$ verification queries. Hence, we have

$$\Pr[B_j] = \Pr[B_j^1 \vee B_j^2 \mid \mathtt{NotZero}_j]$$

where we denote by $\mathtt{NotZero}_j$ the event that $\overline{B_1^1} \wedge \overline{B_1^2} \wedge \ldots \wedge \overline{B_{j-1}^1} \wedge \overline{B_{j-1}^2}$. Let us further note that

$$\begin{aligned}
\Pr[B_j^1 \vee B_j^2 \mid \mathtt{NotZero}_j] &= \Pr[B_j^1 \mid \mathtt{NotZero}_j] + \Pr[B_j^2 \mid \mathtt{NotZero}_j] \\
&= \Pr[W = e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} \wedge C_j^1 \mid \mathtt{NotZero}_j] \\
&\quad + \Pr[y(\alpha) = y'(\alpha) \wedge C_j^2 \mid \mathtt{NotZero}_j] \\
&\leq \Pr[W = e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} \mid C_j^1 \wedge \mathtt{NotZero}_j] \\
&\quad + \Pr[y(\alpha) = y'(\alpha) \mid C_j^2 \wedge \mathtt{NotZero}_j]
\end{aligned}$$

Let us now fix the value of α in the beginning of Game 5. Let us then have a look at what the adversary learns with each query against the challenger. We consider the case first, in which the attacker has not issued any verification query yet. Assume that the attacker have issued n authentication queries, and let $R_1, \ldots, R_n$ be the random values generated in those queries. Then, for each of the p possible values of α, there is only a single value R_i which is valid for α. Indeed, we remind to the reader that in Game 5, a new fresh R_i is generated for each multi-label L, i.e., for every authentication query. There are hence p possible tuples $(\alpha, R_1, \ldots, R_n)$ that are consistent with the attackers view after seeing n authentication queries. Next, we look at the verification queries. It is easy to see that queries with $m \neq y_0$ do not reveal any additional information about α. Moreover, if $\sigma = \sigma'$, then the attacker does not learn anything new about α since all information in this case is computed using Eval with the tags that are already known to the attacker.

Hence, without loss of generality, we assume that all Q verification queries are of case C_j^1, where W is checked against $e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$, *or* of case C_j^2, where $y(\alpha)$ is compared to $y'(\alpha)$. Indeed, as noted before, all the remaining queries can be even answered without using α, and thus they will not reveal any information. After each query of case C_j^1 or C_j^1, if $\overline{B_j^1}$ and $\overline{B_j^2}$ occur, then the number of possible values for $(\alpha, R_1, \ldots, R_n)$ in the attacker's view is reduced by at most d since the zeroes (i.e., the roots) of a non-zero polynomial of degree d are at most d, and the information revealed by a rejection answer says that at most d of such roots (i.e., d possible values of α) can be excluded. In general, after i queries with $\overline{B_1^1} \wedge \overline{B_1^2} \wedge \ldots \wedge \overline{B_i^1} \wedge \overline{B_i^2}$, the number of possible values becomes at least $p - i \cdot d$.

We hence obtain an upper bound on the second probability from above as

$$\Pr[y(\alpha) = y'(\alpha) \mid C_j^2 \wedge \mathtt{NotZero}_j] \leq \frac{d}{p - (j-1)\,d}.$$

This follows from the fact that the polynomial $y(\alpha) - y'(\alpha)$ is non-zero (as $\sigma \neq \sigma'$), its roots are at most d, and by our previous counting argument there are $p - (j-1)\,d$ possible values for α.

To evaluate the first probability $\Pr[W = e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} \mid C_j^1 \wedge \mathtt{NotZero}_j]$, we first note that W is "almost" random since by C_j^1 we know that *all* R_i have never been used before for authentication, and by definition the function f proposed by the adversary in its query is not

a constant function. In particular, the latter property means that at least one of the R_i values, say R_k, "contributes" to the computation of W. Namely, if we fix all values $\{R_i\}_{i\neq k}$, we can write $\mathsf{GroupEval}(f, R_1, \ldots, R_n)$ as $\mathsf{GroupEval}(f', R_k)$ where f' is the univariate degree-d polynomial obtained from f after fixing the values of all variables $\{R_i\}_{i\neq k}$. Notice that after every verification query j involving R_k and in which the event $\mathtt{NotZero}_j$ occurs, the adversary can exclude at most d possible values for R_k. Therefore, at the j-th query, the adversary can not guess R_k with probability better than $1/(p-(j-1)\,d)$. We hence end up with

$$\Pr[W = e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} \mid C_j^1 \wedge \mathtt{NotZero}_j] \leq \frac{d}{p-(j-1)\,d}$$

Using the facts from above, we can give an upper bound for the probability of B_j as

$$\Pr[B_j] \leq \frac{2d}{p-(j-1)\,d}$$

and hence

$$\Pr[\mathsf{Bad}] \leq \frac{2dQ}{p-(Q-1)\,d}$$

which proves the claim for the restricted degree $d = 2$. □

To finalize the proof of Theorem 8, we have to put together the results of all the above Claims. This yields that for any adversary $\mathcal{A}$, it holds

$$\Pr[\mathsf{HomUF{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}} = 1] \leq 2 \cdot \epsilon_{\mathsf{F}} + \frac{8Q}{p-2(Q-1)}.$$

The proof is completed by observing that both quantities ϵ_{F} and $\frac{8Q}{p-2(Q-1)}$ are negligible. For ϵ_{F} this fact follows from the assumption that F is secure, whereas for the second quantity this follows from observing that Q is $\mathsf{poly}(\lambda)$ and that $p \approx 2^{\lambda}$. In other words, a PPT adversary $\mathcal{A}$ has at most a negligible advantage of breaking the unforgeability of $\mathsf{EVH{-}MAC}$. □

5.4 Efficiency Analysis

The efficient verification of $\mathsf{EVH{-}MAC}$ immediately follows from the amortized closed-form efficiency of the pseudorandom function F. Indeed, the verification preparation $\mathsf{VerPrep}$ runs in the same time as $\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\boldsymbol{\tau}}$, and the online verification EffVer runs in the same time as $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\boldsymbol{\Delta}}$. By applying the result of Theorem 3, we thus obtain that $\mathsf{VerPrep}$ and EffVer run in time $O(|f|)$ and $O(1)$, respectively.

In the remainder of this section, we discuss the concrete efficiency of our scheme when implemented with specific security parameters of 80 and 128 bits. In particular, we consider the bandwidth costs for sending the MACs over the network, and the computational timings of the various algorithms at both the client and the server. The timings are obtained by evaluating the most significant operations performed by our algorithms, namely modular exponentiations and pairing computations. For our evaluation, we consider an implementation of Type-A (symmetric) pairings using the PBC library [38], on an 2.5 GHz Intel Core i5 workstation running Mac OS X 10.8.3. The timings of all basic operations needed by our scheme are summarized in Table 1. In addition, we note that by using 80 (resp. 128) bits of security, an element of $\mathbb{Z}_p$ can be represented

Operation	Time (ms)	
	80-bits	128-bits
Pairing	1.23	12
* Pairing	0.62	6.34
Exp. in $\mathbb{G}$	1.83	9.55
* Exp. in $\mathbb{G}$	0.24	1.34
Exp. in $\mathbb{G}_T$	0.22	1.15
* Exp. in $\mathbb{G}_T$	0.05	0.26
Multiexp(2). in $\mathbb{G}$	2.53	13.34
Multiexp(3). in $\mathbb{G}_T$	0.44	2.45

* Costs obtained using precomputation.

Table 1. Summary of costs per operation (in ms).

Operations at the client side	Time (ms)		Size of tags (kB)	
	80 bits	128 bits	80 bits	128 bits
Data Outsourcing	0.24	1.34	0.08	0.22
Verif. w/o prep.	1.06	8.79	0.21	0.59

Table 2. Clients' costs to outsource and to verify.

with 160 (resp. 256) bits, an element of $\mathbb{G}$ with 512 (resp. 1536) bits, and an element of $\mathbb{G}_T$ with 1024 (resp. 3072) bits. Most clients' costs are summarized in Table 2. Below we illustrate how they are obtained, and we give more details on the remaining costs.

To obtain the bandwidth costs, we observe that the MAC σ created by the client, i.e., as generated by Auth, consists of two elements $(y_0, Y_1) \in \mathbb{Z}_p \times \mathbb{G}$, whereas the MAC returned by Eval may include the additional element $\hat{Y}_2 \in \mathbb{G}_T$.

Next, let us consider the computational performances of the algorithms of $\mathsf{EVH}{-}\mathsf{MAC}$. To authenticate a data item, the client runs Auth, whose cost basically boils down to that of computing Y_1. The latter requires one PRF evaluation to generate R (which amounts to one exponentiation in $\mathbb{G}$), plus two other exponentiations, one for m, and one for α^{-1}. However, with a more careful look at our PRF construction, we observe that this operation can be optimized by computing directly $Y_1 = g^{(ua+vb-m)/\alpha}$, a *single* exponentiation in $\mathbb{G}$ (with precomputation on the fixed basis g). For verification, the client has to first prepare the re-usable verification information $\mathsf{VK}_{\mathcal{P}}$ using $\mathsf{VerPrep}$. The cost of this algorithm depends on the computation of $\omega \leftarrow \mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\boldsymbol{\tau}}(K, f)$, which is essentially the same as computing the function f (no exponentiations, pairings or group operations are needed). Such value $\mathsf{VK}_{\mathcal{P}}$ is stored by the client (its size amounts to at most 5 elements of $\mathbb{Z}_p$), and it can be re-used over and over when running $\mathcal{P}$ on different data sets, thus amortizing the cost of its computation. To verify a MAC using EffVer in the online phase, the client needs to compute only one pairing (with precomputation on the fixed g), i.e., $e(Y_1, g)$, and one multi-exponentiation with three bases[7], for $e(g,g)^{y_0-w} e(Y_1, g)^{\alpha} (\hat{Y}_2)^{\alpha^2}$. To conclude our analysis, we consider the cost required to the server for generating the correctness proofs, i.e., to run Eval. As one can notice, Eval evaluates the circuit f with an additional, constant, overhead which derives from replacing

[7] Here we observed that the explicit computation of $W = e(g,g)^w$ in $\mathsf{CFEval}^{\mathsf{on}}$ can be avoided by directly considering $e(g,g)^{y_0-w}$.

every addition of f with the group operation (in either $\mathbb{G}$ or $\mathbb{G}_T$), and every multiplication with one multi-exponentiation in $\mathbb{G}$ plus one pairing.

References

1. S. Agrawal and D. Boneh. Homomorphic MACs: MAC-based integrity for network coding. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 292–305, Paris-Rocquencourt, France, June 2–5, 2009. Springer, Berlin, Germany.
2. B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. G. Spirakis, editors, *ICALP 2010, Part I*, volume 6198 of *LNCS*, pages 152–163, Bordeaux, France, July 6–10, 2010. Springer, Berlin, Germany.
3. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, May 1998.
4. S. Arora and S. Safra. Probabilistic checking of proofs: a new characterization of np. *J. ACM*, 45(1):70–122, Jan. 1998.
5. N. Attrapadung and B. Libert. Homomorphic network coding signatures in the standard model. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 17–34, Taormina, Italy, Mar. 6–9, 2011. Springer, Berlin, Germany.
6. N. Attrapadung, B. Libert, and T. Peters. Computing on authenticated data: New privacy definitions and constructions. In *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 367–385. Springer, Berlin, Germany, Dec. 2012.
7. N. Attrapadung, B. Libert, and T. Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In *PKC 2013*, volume 7778 of *LNCS*, pages 386–404. Springer, Berlin, Germany, 2013.
8. L. Babai. Trading group theory for randomness. In *17th ACM STOC*, pages 421–429, Providence, Rhode Island, USA, May 6–8, 1985. ACM Press.
9. 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.
10. M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya. Incentivizing outsourced computation. In *Workshop on Economics of Networked Systems – NetEcon*, pages 85–90, 2008.
11. S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 111–131, Santa Barbara, CA, USA, Aug. 14–18, 2011. Springer, Berlin, Germany.
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 *ITCS '12: Proceedings of the 3rd Symposium on Innovations in Theoretical Computer Science*, 2012.
13. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55, Santa Barbara, CA, USA, Aug. 15–19, 2004. Springer, Berlin, Germany.
14. D. Boneh, D. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In S. Jarecki and G. Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 68–87, Irvine, CA, USA, Mar. 18–20, 2009. Springer, Berlin, Germany.
15. D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 149–168, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Germany.
16. D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 1–16, Taormina, Italy, Mar. 6–9, 2011. Springer, Berlin, Germany.
17. D. Catalano and D. Fiore. Practical homomorphic MACs for arithmetic circuits. In *Eurocrypt '13: Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2013.
18. D. Catalano, D. Fiore, R. Gennaro, and K. Vamvourellis. Algebraic (trapdoor) one way functions and their applications. In *TCC 2013*, volume 7785 of *LNCS*, pages 680–699. Springer, Berlin, Germany, 2013.
19. D. Catalano, D. Fiore, and B. Warinschi. Adaptive pseudo-free groups and applications. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 207–223, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Germany.
20. D. Catalano, D. Fiore, and B. Warinschi. Efficient network coding signatures in the standard model. In M. Fischlin, J. Buchmann, and M. Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 680–696, Darmstadt, Germany, May 21–23, 2012. Springer, Berlin, Germany.

21. K.-M. Chung, Y. Kalai, and S. P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 483–501, Santa Barbara, CA, USA, Aug. 15–19, 2010. Springer, Berlin, Germany.
22. K.-M. Chung, Y. T. Kalai, F.-H. Liu, and R. Raz. Memory delegation. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 151–168, Santa Barbara, CA, USA, Aug. 14–18, 2011. Springer, Berlin, Germany.
23. G. Di Battista and B. Palazzi. Authenticated relational tables and authenticated skip lists. In *Proceedings of the 21st annual IFIP WG 11.3 working conference on Data and applications security*, pages 31–46, Berlin, Heidelberg, 2007. Springer-Verlag.
24. U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, Mar. 1996.
25. D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *2012 ACM Conference on Computer and Communication Security*. Full version available at `http://eprint.iacr.org/2012/281`. ACM Press, October 2012.
26. D. M. Freeman. Improved security for linearly homomorphic signatures: A generic framework. In M. Fischlin, J. Buchmann, and M. Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 697–714, Darmstadt, Germany, May 21–23, 2012. Springer, Berlin, Germany.
27. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482, Santa Barbara, CA, USA, Aug. 15–19, 2010. Springer, Berlin, Germany.
28. R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Eurocrypt '13: Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2013. Also in Cryptology ePrint Archive, Report 2012/215, `http://eprint.iacr.org/2012/215`.
29. R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin. Secure network coding over the integers. In P. Q. Nguyen and D. Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 142–160, Paris, France, May 26–28, 2010. Springer, Berlin, Germany.
30. R. Gennaro and D. Wichs. Fully homomorphic message authenticators. Cryptology ePrint Archive, Report 2012/290, 2012. `http://eprint.iacr.org/`.
31. C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In L. Fortnow and S. P. Vadhan, editors, *43rd ACM STOC*, pages 99–108, San Jose, California, USA, June 6–8, 2011. ACM Press.
32. S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In R. E. Ladner and C. Dwork, editors, *40th ACM STOC*, pages 113–122, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press.
33. M. T. Goodrich, R. Tamassia, and J. Hasic. An efficient dynamic and distributed cryptographic accumulator. In A. H. Chan and V. D. Gligor, editors, *ISC 2002*, volume 2433 of *LNCS*, pages 372–388, Sao Paulo, Brazil, Sept. 30 – Oct. 2, 2002. Springer, Berlin, Germany.
34. M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen. Authenticated data structures for graph and geometric searching. In M. Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 295–313, San Francisco, CA, USA, Apr. 13–17, 2003. Springer, Berlin, Germany.
35. R. Johnson, D. Molnar, D. X. Song, and D. Wagner. Homomorphic signature schemes. In B. Preneel, editor, *CT-RSA 2002*, volume 2271 of *LNCS*, pages 244–262, San Jose, CA, USA, Feb. 18–22, 2002. Springer, Berlin, Germany.
36. J. Kilian. A note on efficient zero-knowledge proofs and arguments. In *24th ACM STOC*, pages 723–732, Victoria, British Columbia, Canada, May 4–6, 1992. ACM Press.
37. A. B. Lewko and B. Waters. Efficient pseudorandom functions from the decisional linear assumption and weaker variants. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *ACM CCS 09*, pages 112–120, Chicago, Illinois, USA, Nov. 9–13, 2009. ACM Press.
38. B. Lynn. PBC: The pairing-based crypto library.
39. C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, 2004.
40. S. Micali. CS proofs. In *35th FOCS*, Santa Fe, New Mexico, Nov. 20–22, 1994.
41. F. Monrose, P. Wyckoff, and A. D. Rubin. Distributed execution with remote audit. In *NDSS'99*, San Diego, California, USA, Feb. 3–5, 1999. The Internet Society.
42. M. Naor and K. Nissim. Certificate revocation and certificate update. In *in Proceedings of the 7th USENIX Security Symposium*, pages 217–228, 1998.

43. C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation, 2013.
44. C. Papamanthou and R. Tamassia. Time and space efficient algorithms for two-party authenticated data structures. In S. Qing, H. Imai, and G. Wang, editors, *ICICS 07*, volume 4861 of *LNCS*, pages 1–15, Zhengzhou, China, Dec. 12–15, 2007. Springer, Berlin, Germany.
45. C. Papamanthou, R. Tamassia, and N. Triandopoulos. Optimal verification of operations on dynamic sets. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 91–110, Santa Barbara, CA, USA, Aug. 14–18, 2011. Springer, Berlin, Germany.
46. B. Parno, C. Gentry, J. Howell, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy, Oakland*, 2013.
47. B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In R. Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 422–439, Taormina, Sicily, Italy, Mar. 19–21, 2012. Springer, Berlin, Germany.
48. S. Setty, B. Braun, V. Vu, A. Blumberg, B. Parno, and M. Walfish. Resolving the conflict between generality and plausibility in verified computation. ACM European Conference on Computer Systems, EuroSys 2013.
49. S. Setty, R. McPherson, A. Blumberg, and M. Walfish. Making argument systems for outsourced computation practical (sometimes). In *Network & Distributed System Security Symposium, NDSS 2012*, 2012.
50. S. Setty, V. Vu, N. Panpalia, B. Braun, A. Blumberg, and M. Walfish. Taking proof-based verified computation a few steps closer to practicality. In *USENIX Security Symposium, Security 2012*, August 2012.
51. A. Shpilka and A. Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010.
52. S. W. Smith and S. Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks*, 31:831–860, 1999.
53. R. Tamassia. Authenticated data structures. In G. Battista and U. Zwick, editors, *Algorithms - ESA 2003*, volume 2832 of *Lecture Notes in Computer Science*, pages 2–5. Springer Berlin Heidelberg, 2003.
54. V. Vu, S. Setty, A. Blumberg, and M. Walfish. A hybrid architecture for interactive verifiable computation. IEEE Symposium on Security and Privacy, Oakland 2013, to appear.
55. B. Yee. *Using Secure Coprocessors*. PhD thesis, Carnegie Mellon University, 1994.