Homomorphic Signatures with Efficient Verification for Polynomial Functions

Dario Catalano[1], Dario Fiore[2], and Bogdan Warinschi[3]

[1] Università di Catania, Italy. catalano@dmi.unict.it
[2] IMDEA Software Institute, Spain. dario.fiore@imdea.org
[3] University of Bristol, UK. bogdan@compsci.bristol.ac.uk

Abstract. A homomorphic signature scheme for a class of functions $\mathcal{C}$ allows a client to sign and upload elements of some data set D on a server. At any later point, the server can derive a (publicly verifiable) signature that certifies that some y is the result computing some $f \in \mathcal{C}$ on the basic data set D. This primitive has been formalized by Boneh and Freeman (Eurocrypt 2011) who also proposed the only known construction for the class of multivariate polynomials of fixed degree $d \geq 1$. In this paper we construct new homomorphic signature schemes for such functions. Our schemes provide the first alternatives to the one of Boneh-Freeman, and improve over their solution in three main aspects. First, our schemes do not rely on random oracles. Second, we obtain security in a stronger fully-adaptive model: while the solution of Boneh-Freeman requires the adversary to query messages in a given data set all at once, our schemes can tolerate adversaries that query one message at a time, in a fully-adaptive way. Third, signature verification is more efficient (in an amortized sense) than computing the function from scratch. The latter property opens the way to using homomorphic signatures for publicly-verifiable computation on outsourced data. Our schemes rely on a new assumption on leveled graded encodings which we show to hold in a generic model.

1 Introduction

Cryptographic mechanisms for building trust are essential for the shift towards a world where weak clients leverage access to all-powerful servers to remotely store and compute on data. Trust issues include availability of storage, privacy of data, authenticity of delegated computation, etc. which in turn take a multitude of forms. For example, privacy concerns range from simply ensuring the secrecy of stored data, to additionally allowing for search over outsourced data and/or optimizing storage space. This paper contributes to the area of *verifiable computation*, and specifically to the setting in which a client delegates the computation of one or more functions $F_1, F_2, \ldots, F_n$ on one or more of its data sets $D_1, D_2, \ldots, D_m$. The crucial requirement here is that the answer y returned by the server, purportedly the result of $F_i(D_j)$, can be efficiently verified. Efficiency has multiple dimensions, but two are needed to avoid trivial solutions: the client should not have to store all data D_j on which the server computes and/or verification should be faster than simply computing F_i on D_j.

In addition to the different forms of efficiency one may require, the problem of verifiable computation also comes in several different scenarios. For example, the function computed by the server may be fixed or changing, the data stored may be fixed or incrementally updated, the client may have access to multiple (non-communicating) servers, the verification of the result may be interactive, etc. In this work we focus on the scenario where the client has access to a single server, he can incrementally add data on the server, the functions to be computed are not known in advance, and the verification of the result is non-interactive and can be done publicly.

To place our contribution in the landscape of solutions for verifiable computation and to facilitate the comparison with existent solutions, we note that previously proposed protocols for verifiable computation use one of two techniques. The first type of solutions (which for brevity we call proof-based) build on foundations going back to Micali's computationally sound proofs [26]. The idea is for the server to provide (or to prove knowledge of) a certificate for the NP statement: $y = F(D)$. The earlier work used probabilistically checkable proofs (PCPs) [26], whereas recent results rely on succinct arguments (SNARGs) or succinct arguments of knowledge (SNARKs) [7,22] where the dependency between the length of the statement and the proof is greatly reduced. Other protocols where proofs are not explicitly mentioned can be thought of as instantiations where the proofs are encrypted information-theoretic secure MAC [21,27].

The second type of solutions use homomorphic authenticators; we refer to these constructions as authenticator-based. In these constructions, one attaches to every input data an unforgeable *authenticator*. The main property is that any operation (gate) used in the computation which takes as input correctly authenticated data, produces a result together with a valid authenticator. Solutions exist in both the symmetric and the public-key setting. Depending on how the authenticator is verified, we distinguish between homomorphic message authentication codes [24,10,4] and homomorphic signatures [9]. Clearly, the difficulty of the problem increases with the class of functions one considers. For example, there are numerous signature schemes homomorphic with respect to linear functions over vector spaces [1,8,23,2,13,14,19,3,12]. In contrast, there has been little progress on signature schemes homomorphic with respect to non-linear polynomials. The only known construction is provided by Boneh and Freeman [9] who construct a homomorphic signature scheme for multivariate polynomials of constant degree.

Summary of our contribution and relation to previous work. In this paper we provide the first alternative to the homomorphic signature scheme of Boneh and Freeman (henceforth BF), which is the work closest to ours. Our result improves over the BF solution in three main aspects. First, we solve a problem left open in [9], as unlike the BF scheme, our construction does *not* rely on the random oracle assumption. Second, our scheme is proven secure in a stronger adaptive model: in the BF scheme the adversary is restricted to query signatures on messages belonging to a given data set all at once; in contrast, our construction is proven secure against adversaries that can query *one message at a time* in a

fully adaptive way. Finally, our construction enjoys *efficient verification* in that verifying a signature against a function f can be done faster than computing f (and in particular does not require storing the input data). More accurately, this property holds in an amortized sense: after a single (local) pre-computation of f, one can verify the evaluation of f on any dataset more efficiently. This property has been recently identified, defined and realized for homomorphic MACs in [4]. Our construction is the *first* to achieve efficient verification for homomorphic signatures, and therefore it opens the way to using homomorphic signatures for verifiable computation.

We remark that other constructions of homomorphic authenticators are either in the symmetric key setting [24,10,4], or are for the restricted class of linear functions [1,8,23,2,13,14,19,3,12]. Below we discuss the benefits that our solution brings to the broader field of verifiable computation. We start with general remarks on the benefits that authenticator-based solutions hold over proof-based ones.

Incremental, compositional verifiable computation. Homomorphic authenticators naturally give rise to incremental/composable verifiable computation: the output of some computation on authenticated data is already authenticated so it can be fed as input for follow-up computation. This property is of particular interest to parallelize computations (e.g., MapReduce). Emulating this composition within the proof-based frameworks is possible [7] but it leads to complex statements and less natural realizations. For an extensive discussion of this issue see [24].

Flexible scenarios. Furthermore, homomorphic authenticators are applicable to a broader range of scenarios as neither the data to be computed on, nor the function to be applied need to be known in advance. For example the data can be incrementally updated (by authenticating and uploading new pieces of data), and the function to be applied can be selected at any point by the server (without having to wait for some parameters generated by the client). In contrast, in (most) proof-based solutions the function needs to be known at the moment when data is uploaded, or a copy of the data needs to be kept locally by the client [21,15,6,18,27,7,22]. Perhaps the biggest advantage of verifiable computation based on authenticators is that verification does not need the input data; indeed we only need to check that the result comes with a valid authenticator. Just like for incremental computation, an analogous result can be obtained with proof-based constructions through theoretically beautiful but practically cumbersome solutions. For example, one can fix the computation performed by the server to be some universal circuit and then see the actual function to be computed as part of the data that is uploaded. While the dependency between data and functions is broken, verification would still need the whole data (and function description) as input.

Improving flexibility is also addressed by the notions of memory delegation and streaming delegation [16] in which a client can outsource a large memory to a server, keeps a small local state, and can later delegate and verify computations on the outsourced memory. This setting is very general and is close to

the one achieved by using homomorphic authenticators. As mentioned in [24], a difference between memory delegation and homomorphic authenticators is that the former considers a single user who outsources the data all at once and keeps a state associated with the data. In contrast, by using homomorphic authenticators various users may independently upload several data items without sharing any state (beyond the fixed signing key).

Complexity assumptions. In terms of the usual trade-off between efficiency and the underlying assumptions our scheme fares well. Most proof-based constructions rely on proofs (SNARGs, SNARKs) for which instantiations either rely on the random oracle model [26] or employ non-falsifiable assumptions. Our scheme is in the standard model and is based on problems in the groups underlying a multi-linear map. Our scheme can be instantiated with any of the existing graded encoding schemes [20,17] and hence it will increase in efficiency with any progress on the implementation of the latter primitive [25].

High level idea of our construction. Our scheme signs messages in $\mathbb{Z}_p$ and is homomorphic with respect to polynomial functions on $\mathbb{Z}_p^n$ (where n is the size of the data set); the degree of the polynomial is d (which is bounded).

To realize our construction we proceed in three main stages. First we construct an homomorphic scheme (with the same domain, and homomorphic with respect to the same class of functions) secure in a weaker sense: in an attack, the adversary asks all of the messages to be signed non-adaptively before the scheme is initialized. This is the technically most difficult part of the paper. Then we provide a generic transformation that strengthens any weakly-secure homomorphic signature for degree-d polynomials to an adaptive-secure one, i.e. one that withstands adaptive chosen-message attacks. The third step is to optimize the resulting construction when instantiated with the weakly-secure scheme that we develop. Below we provide an overview of these steps, starting with the generic transformation. Then we describe the main ideas that go into the construction of our weakly-secure scheme. To conclude, we discuss the efficiency of the scheme that we obtain from our weakly-secure scheme via both the generic and the optimized transformation.

In both schemes we encode a message in $\mathbb{Z}_p$ as the free term of a polynomial of degree at most d. Messages in the data set are encoded in polynomials of degree one, whereas the results of computations will be encoded by higher degree polynomials. Start with a weakly secure homomorphic signature scheme Π. The signing key for the scheme we construct consists of $d+1$ different signing keys for Π, say $sk_1, sk_2, \ldots, sk_{d+1}$. If message m is encoded by some polynomial t, then a signature on m is of the form $(\sigma_1, \sigma_2, \ldots, \sigma_{d+1})$, where σ_i is a signature using the weakly secure scheme on $t(i)$ using sk_i. Since we only work with polynomials of degree at most d, the $d+1$ points that are signed uniquely determine the polynomial t, hence the message m. Homomorphicity of the scheme that we construct follows from that of the underlying scheme. Given signatures $(\sigma_1^1, \sigma_2^1, \ldots, \sigma_{d+1}^1)$ and $(\sigma_1^2, \sigma_2^2, \ldots, \sigma_{d+1}^2)$ for messages m_1 and m_2, a signature on $m_1 \circ m_2$ (where $\circ$ is one of the operations in $\mathbb{Z}_p$) is $(\sigma_1^1 \circ \sigma_1^2, \sigma_2^1 \circ \sigma_2^2, \ldots, \sigma_{d+1}^1 \circ \sigma_{d+1}^2)$. Without going into the details, a key idea of using the encoding of messages into polynomials

is that a simulator can adaptively sign arbitrary messages, while having access only to signatures (of Π) on a set of random messages.

Our construction of the weakly-secure signature scheme is based on graded encodings [20]. The overview here uses the (more idealized) leveled multilinear maps setting. The basic idea is that a signature on a data set message m_i is a level-1 element of the form $\Lambda = g^{(r_i - m_i x)b}$, where g^{r_i} is some public information, b is the secret key and g, g^x, g^b are in the public key[4]. Given signatures on messages m_1 and m_2 one obtains a signature on the sum by simply computing $\Lambda_1 \cdot \Lambda_2$. To obtain a signature on the multiplication $m_1 \cdot m_2$, we apply the graded map to $g^{(r_1 - m_1 x)b}$ and $g^{(r_2 - m_2 x)b}$ and obtain something of the form $g_2^{[r_1 r_2 - (r_1 m_2 + r_2 m_1)x + m_1 m_2 x^2]b^2}$ where g_2 is a generator of $\mathbb{G}_2$. The main issue with the resulting signature is verification: here, one should either know the original messages m_1, m_2 (which is what we want to avoid) or keep track in the signature of the middle term in the exponent (which we also want to avoid since this term grows with successive multiplications). We solve this problem with two main ideas: (1) we publish a randomized version g^{abx} of the secret value g^{bx}, and (2) we create a twin version of every signature which has the form $\Gamma = g^{(r - mx)ab}$. This way, a signature on the multiplication of $m_1 \cdot m_2$ is obtained by applying the graded map to $\Lambda_1 = g^{(r_1 - m_1 x)b}$ and $\Gamma_2 = g^{(r_2 - m_2 x)ab}$, which produces something of the form $g_2^{[r_1 r_2 - (r_1 m_2 + r_2 m_1)x + m_1 m_2 x^2]ab^2}$. Then, by using g^{abx}, the latter value can now be "cleaned up" (by multiplying appropriately computed values) to obtain $g_2^{[r_1 r_2 - m_1 m_2 x^2]ab^2}$. More generally, we show how to clean the multiplication of arbitrary signatures to always produce a signature of the form $g_i^{[f(\mathbf{r}) - f(\mathbf{m})x^i]a^{i-1}b^i}$, where i is the degree of polynomial f, g_i is the generator in $\mathbb{G}_i$, $\mathbf{m}$ is the vector of original messages, and $\mathbf{r}$ is the vector of r_i' in the publicly known $g^{r_1}, g^{r_2}, \ldots$. Related issues that we solve include enabling verification of these signatures, and ensuring that the cleaning information does not enable the creation of forgeries. Also, while the simplified description above works for signatures in a single dataset, our full realization provides a way to deal with multiple datasets. The construction sketched above is homomorphic for polynomials of degree-d, if instantiated with $2d$-linear maps, and is proven weakly-secure under a new, constant-size, assumption that we prove hard in the generic multilinear group model. As a final note, we observe that this construction enjoys efficient verification, which (intuitively) follows from that one can precompute $f(\boldsymbol{r})$ and reuse it to verify *all* signatures for the same f. In terms of efficiency, in this weakly-secure construction every signature consists of the message m and two group elements—Λ, Γ—and is in principle of constant size. When instantiated with currently known graded encoding schemes, each of these group elements (aka encodings) is of size $O(d^2 + d \log n)$ (ignoring the security parameter), if we want to support n-variate polynomials of degree d.

By applying our generic transformation we obtain an adaptive-secure homomorphic signature in which signatures have size $O(d)$, which turns into $O(d^3 +$

[4] We emphasize that our signatures are quite different, and we only use these to explain the intuition.

$d^2 \log n$) when instantiated with known graded encoding schemes [20,17,25]. We also show a more optimized transformation tailored to our weakly-secure scheme, which yields a more efficient adaptive-secure homomorphic signature where, for instance, the size of the public and the secret key does not grow by a factor of d. Furthermore, we show that our weakly-secure scheme can be also proven adaptive-secure, though by assuming a stronger, interactive, assumption.

2 Preliminaries

2.1 Leveled Multilinear Maps and Graded Encodings

In this section we recall the definition of leveled multilinear maps and the computational assumptions used in our scheme. Candidate implementations of this abstraction have been recently proposed [20,17,25] in the form of *graded encodings*, a concept similar to generic, leveled multilinear maps.

In generic, symmetric, leveled multilinear maps we assume the existence of an algorithm $\mathcal{G}(1^\lambda, k)$ that, on input the security parameter and an integer k indicating the number of levels (i.e., the number of allowed pairing operations), generates the description pp of leveled multilinear groups $(\mathbb{G}_1, \ldots, \mathbb{G}_k)$, each of large prime order $p > 2^\lambda$. We let g_i be a canonical generator of $\mathbb{G}_i$; we assume that pp includes $g_1 \in \mathbb{G}_1$. The groups are such that there exists a set of bilinear maps $\{e_{i,j} : \mathbb{G}_i \times \mathbb{G}_j \to \mathbb{G}_{i+j}\}_{i,j \geq 1, i+j \leq k}$ such that $\forall a, b \in \mathbb{Z}_p$: $e_{i,j}(g_i^a, g_j^b) = g_{i+j}^{ab}$. When obvious from the context we drop the indices i, j from $e_{i,j}$. We work with symmetric bilinear maps and we let the canonical generators $g_i \in \mathbb{G}_i$ be obtained by repeatedly applying the map to g_1, i.e. we let $g_i = e(g_1, g_{i-1})$.

Hardness Assumption. Below we define the computational assumption that underlies the security of our scheme. In the full version we justify the assumption by proving it holds in a generic model for level multilinear maps. The assumption can also be tested using recently proposed automated techniques [5]. Informally, the assumption says that given the level-1 encodings $g_1^a, g_1^b, g_1^{ab}, g_1^x, g_1^{xa}, g_1^{abx}$ with $a, b, x \in \mathbb{Z}_p$ random, it must be hard to compute a level-k encoding of $a^{k-1}(bx)^k$ (i.e., $g_k^{a^{k-1}(bx)^k}$). More formally:

Definition 1 (k-Augmented-Power Multilinear Diffie-Hellman). *Let* pp *be the description of a set of multilinear groups and* $g_1 \in \mathbb{G}_1$ *be a random generator. Let* $a, b, x \xleftarrow{\$} \mathbb{Z}_p$ *be chosen at random. We define the advantage of an adversary* $\mathcal{A}$ *in solving the k-APMDH problem as* $\mathbf{Adv}_{\mathcal{A}}^{APMDH}(\lambda) = \Pr[\mathcal{A}(g_1, g_1^a, g_1^b, g_1^{ab}, g_1^x, g_1^{ax}, g_1^{abx}) = g_k^{a^{k-1}(bx)^k}]$*, and we say that the k-APMDH assumption holds for* $\mathcal{G}$ *if for every PPT* $\mathcal{A}$*,* $\mathbf{Adv}_{\mathcal{A}}^{APMDH}(\lambda)$ *is negligible in* λ*.*

Graded Encodings. Informally speaking, a k-graded encoding system for a ring R includes a system of sets $\{S_i^{(\alpha)} \subset \{0,1\}^* : i \in [0,k], \alpha \in R\}$ such that for every fixed $i \in [0,k]$ the sets $\{S_i^{(\alpha)} : \alpha \in R\}$ are disjoint. The set $S_i^{(\alpha)}$ contains the level-i encodings of $\alpha \in R$. As a first requirement, the system needs an algorithm to obtain an encoding $a_i \in S_i^{(\alpha)}$ of some ring element α (notice that such encoding can be randomized). Additionally, the encoding system is homomorphic in a graded sense. Namely, let us abuse notation and assume that every

set $S_i^{(\alpha)}$ is a ring where $+, \cdot$ are the usual addition/multiplication operations. Then, for any $a_i \in S_i^{(\alpha)}$ and $b_i \in S_i^{(\beta)}$ we have $a_i + b_j \in S_i^{(\alpha+\beta)}$. Furthermore, for $a_i \in S_i^{(\alpha)}$ and $b_j \in S_j^{(\beta)}$ we have $a_i \cdot b_j \in S_{i+j}^{(\alpha \cdot \beta)}$, if $i + j \leq k$. Finally, the encoding system has an algorithm to test if a given a is an encoding of 0 in the last level k, i.e., if $a \in S_k^{(0)}$. We refer to [20] or the full version of our work for a more precise description of graded encodings.

2.2 Homomorphic Signatures for Multi-Labeled Programs

In this section we provide the definition of homomorphic signatures. Our definition is essentially the same as the one proposed by Freeman in [19] except that we adapt it to work in the model of *multi-labeled programs* introduced in [4] as an extension to labeled programs [24,10].

Multi-Labeled Programs. A *labeled program* $\mathcal{P}$ consists of a tuple $(f, \tau_1, \ldots, \tau_n)$ such that $f : \mathcal{M}^n \to \mathcal{M}$ is a function on n variables (e.g., a circuit), and $\tau_i \in \{0,1\}^*$ is the label of the i-th variable input of f. Labeled programs can be composed in the following way. Given $\mathcal{P}_1, \ldots, \mathcal{P}_t$ and a function $g : \mathcal{M}^t \to \mathcal{M}$, the *composed program* $\mathcal{P}^*$ is the one obtained by evaluating g on the outputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$, and 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. Let $f_{id} : \mathcal{M} \to \mathcal{M}$ be the canonical identity function and $\tau \in \{0,1\}^*$ be a label. Then $\mathcal{I}_\tau = (f_{id}, \tau)$ is the *identity program* for input label τ. Using this notation, observe that any program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ can be expressed as the composition of n identity programs $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \ldots, \mathcal{I}_{\tau_n})$.

A *multi-labeled program* $\mathcal{P}_\Delta$ is a pair $(\mathcal{P}, \Delta)$ in which $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ is a labeled program 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 $(\mathcal{P}_1, \Delta), \ldots, (\mathcal{P}_t, \Delta)$ sharing 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 $\mathcal{P}_i$. Similarly to the labeled case, we define a multi-labeled identity program as $\mathcal{I}_{(\Delta,\tau)} = ((f_{id}, \tau,), \Delta)$.

Definition 2 (Homomorphic Signatures). *A homomorphic signature scheme* HomSig *is a tuple of probabilistic, polynomial-time algorithms* ($\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{Eval}$) *satisfying four properties:* authentication correctness*,* evaluation correctness*,* succinctness*, and* security*. More precisely:*

- $\mathsf{KeyGen}(1^\lambda, \mathcal{L})$ *takes a security parameter* λ*, the description of the label space* $\mathcal{L}$ *(possibly fixing a maximum data set size* N*), and outputs a public key* vk *and a secret key* sk*. The public key* vk *defines implicitly a message space* $\mathcal{M}$ *and a set* $\mathcal{F}$ *of admissible functions.*
- $\mathsf{Sign}(\mathsf{sk}, \Delta, \tau, m)$ *takes a secret key* sk*, a data set identifier* Δ*, a label* $\tau \in \mathcal{L}$*, a message* $m \in \mathcal{M}$*, and it outputs a signature* σ*.*
- $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_\Delta, m, \sigma)$ *takes a public key* vk*, a multi-labeled program* $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$ *with* $f \in \mathcal{F}$*, a message* $m \in \mathcal{M}$*, and a signature* σ*. It outputs either 0 (reject) or 1 (accept).*

$\mathsf{Eval}(\mathsf{vk}, f, \boldsymbol{\sigma})$ *takes a public key* vk*, a function* $f \in \mathcal{F}$ *and a tuple of signatures* $\{\sigma_i\}_{i=1}^n$ *(assuming that* f *takes* n *inputs). It outputs a new signature* σ*.*

Authentication Correctness. Intuitively, a homomorphic signature satisfies authentication correctness if the signatures generated by $\mathsf{Sign}(\mathsf{sk}, \Delta, \tau, m)$ verify correctly for m as the output of the identity program $\mathcal{I}_{(\Delta,\tau)}$. Formally, HomSig has authentication correctness if for a given label space $\mathcal{L}$, all key pairs $(\mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, \mathcal{L})$, any label $\tau \in \mathcal{L}$, data set identifier $\Delta \in \{0,1\}^*$, and any signature $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, \Delta, \tau, m)$, $\mathsf{Ver}(\mathsf{vk}, \mathcal{I}_{(\Delta,\tau)}, m, \sigma)$ outputs 1 with all but negligible probability.

Evaluation Correctness. Informally, this property says that running the evaluation algorithm on signatures $(\sigma_1, \ldots, \sigma_n)$ such that σ_i verifies for m_i as the output of a multi-labeled program $(\mathcal{P}_i, \Delta)$, produces a signature σ which verifies for $f(m_1, \ldots, m_n)$ as the output of the composed program $(f(\mathcal{P}_1, \ldots, \mathcal{P}_n), \Delta)$. More formally, fix a key pair $(\mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, \mathcal{L})$, a function $g : \mathcal{M}^t \to \mathcal{M}$ and any set of program/message/signature triples $\{(\mathcal{P}_i, m_i, \sigma_i)\}_{i=1}^t$ such that $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_i, m_i, \sigma_i) = 1$. If $m^* = g(m_1, \ldots, m_t)$, $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$, and $\sigma^* = \mathsf{Eval}(\mathsf{vk}, g, (\sigma_1, \ldots, \sigma_t))$, then $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}^*, m^*, \sigma^*) = 1$ holds with all but negligible probability.

Succinctness. A homomorphic signature scheme is succinct if, for a fixed security parameter λ, the size of the signatures depends at most logarithmically on the data set size N.

Security. We say that a homomorphic signature scheme HomSig is *secure* if for every PPT adversary $\mathcal{A}$ we have $\Pr[\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HomSig}}(\lambda) = 1] \leq \epsilon(\lambda)$ where $\epsilon(\lambda)$ is a negligible function, and the experiment $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HomSig}}(\lambda)$ is defined as follows.

Key generation The challenger runs $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, \mathcal{L})$ and gives vk to the adversary.

Signing Queries The adversary can adaptively submit queries of the form (Δ, τ, m), where Δ is a dataset identifier, $\tau \in \mathcal{L}$, and $m \in \mathcal{M}$. The challenger proceeds as follows: If (Δ, τ, m) is the first query with data set identifier Δ, then the challenger initializes an empty list $T_\Delta = \emptyset$ for Δ. If T_Δ does not already contain a tuple $(\tau, \cdot)$ (i.e., the adversary never asked for a query $(\Delta, \tau, \cdot)$), the challenger computes $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, \Delta, \tau, m)$, 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 adversary had already queried the tuple (Δ, τ, m)), then the challenger replies with the same signature generated before. If T_Δ contains a tuple (τ, m') for some message $m' \neq m$, then the challenger ignores the query.

Forgery The previous stage is repeated a polynomial number of times until the adversary outputs a tuple $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$.

Finally, the experiment outputs 1 if the tuple returned by the adversary is a forgery, and 0 otherwise. However, to do this we need to provide a way for characterizing forgeries in this model. To this end, we recall the notion of well-defined program w.r.t. a list T_Δ [19]. A labeled program $\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*)$ is *well-defined with respect to* T_{Δ^*} if 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 challenger has generated signatures for the entire input space of f for data set Δ^*.
– there exist indices $i \in \{1, \ldots, n\}$ such that $(\tau_i^*, \cdot) \notin T_{\Delta^*}$ (i.e., $\mathcal{A}$ never asked signing queries of the form $(\Delta^*, \tau_i^*, \cdot)$), 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 inputs that were not signed in the experiment never contribute to the computation of f.

The experiment HomUF-CMA outputs 1 if and only if $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}^*_{\Delta^*}, m^*, \sigma^*) = 1$ and one of the following conditions holds:

– *Type 1 Forgery:* no list T_{Δ^*} was created during the game, i.e., during the experiment no message m has ever been signed with respect to a data set identifier Δ^*.
– *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 signed messages $(m_1, \ldots, m_n)$.
– *Type 3 Forgery:* $\mathcal{P}^*$ is *not* well-defined w.r.t. T_{Δ^*}.

As pointed out by Freeman [19], for a general class of functions it may not be possible for the challenger to efficiently decide whether a given program is well-defined or not. Freeman shows that for the case of linearly-homomorphic signatures this is not an issue. More precisely he shows that any adversary who outputs a Type-3 forgery can be converted into one that outputs a Type-2 forgery. Below, we show two simple propositions that allow to overcome this issue for the case of homomorphic signatures whose class of supported functions are arithmetic circuits of degree d, over a finite field of order p such that $d/p < 1/2$. The first proposition is taken from [11] and provides a way to probabilistically test whether a program is well-defined.

Proposition 1 ([11]). *Let $\lambda, n \in \mathbb{N}$ and let $\mathcal{F}$ be the class of arithmetic circuits $f : \mathbb{F}^n \to \mathbb{F}$ over a finite field $\mathbb{F}$ of order p and such that the degree of f is at most d, for $\frac{d}{p} < \frac{1}{2}$. Then, there exists a probabilistic polynomial-time algorithm that for any given $f \in \mathcal{F}$, decides if there exists $y \in \mathbb{F}$ such that $f(\boldsymbol{u}) = y, \forall \boldsymbol{u} \in \mathbb{F}^n$ (i.e., if f is constant) and is correct with probability at least $1 - 2^{-\lambda}$.*

The second proposition below is the analogue of the one proven by Freeman, which shows that any adversary who outputs a Type-3 forgery can be converted into one that outputs a Type-2 forgery. This result has been proven for homomorphic MACs in [11]. Here we extend it to homomorphic signatures. For lack of space, its proof appears in the full version.

Proposition 2. *Let $\lambda \in \mathbb{N}$ be the security parameter, and let $\mathcal{F}$ be the class of arithmetic circuits $f : \mathbb{F}^n \to \mathbb{F}$ over a finite field $\mathbb{F}$ of order p and such that the degree of f is at most d, for $\frac{d}{p} < \frac{1}{2}$. Let* HomSig *be a signature scheme with message space $\mathbb{F}$, and let $\mathcal{E}_b$ be the event that the adversary returns a Type-b forgery (for $b = 1, 2, 3$) in experiment* HomUF-CMA*. Then, if for any adversary*

$\mathcal{B}$ *we have that* $\Pr[\mathsf{HomUF\text{-}CMA}_{\mathcal{B},\mathsf{HomSig}}(\lambda) = 1 \wedge \mathcal{E}_2] \leq \epsilon$*, then for any adversary* $\mathcal{A}$ *producing a Type-3 forgery it holds* $\Pr[\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HomSig}}(\lambda) = 1 \wedge \mathcal{E}_3] \leq \epsilon + 2^{-\lambda}$.

Weakly-Secure Homomorphic Signatures. In our work we also consider a weaker notion of unforgeability for homomorphic signatures. We define experiment $\mathsf{Weak\text{-}HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HomSig}}$ which is a variant of $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HomSig}}$. The difference is that before key generation $\mathcal{A}$ declares *all* the signing queries that it will make (i.e., messages), but without necessarily specifying the data set names, i.e., $\mathcal{A}$ outputs $\{m_{\tau,j}\}_{\tau\in\mathcal{L}}$, for $j = 1$ to Q, where Q is the number of different queried datasets. Once applying the above change, in the signing query phase, $\mathcal{A}$ will only specify a data set Δ_j and will receive signatures on $\{(\Delta_j, \tau, m_{\tau,j})\}_{\tau\in\mathcal{L}}$. Also, notice that with this change, there are no Type-3 forgeries as the data sets are always full.

While this security notion may look rather weak, in Section 3 we show a generic way to convert any weakly-secure homomorphic signature for arithmetic circuits of degree d to an adaptively secure one (for the same class of functions.)

2.3 Homomorphic Signatures with Efficient Verification

We propose the notion of homomorphic signatures with efficient verification, which naturally extends to the public-key setting the analogous notion introduced for homomorphic MACs in [4]. Roughly speaking, this property says that the verification algorithm can be split in two phases. In an offline phase, given the verification key vk and a labeled program $\mathcal{P}$, one precomputes a concise key $\mathsf{vk}_{\mathcal{P}}$. The latter key can then be used to verify signatures (in the online phase) w.r.t. $\mathcal{P}$ and *any* dataset Δ. Crucially, $\mathsf{vk}_{\mathcal{P}}$ can be reused an unbounded number of times, and the verification cost of the online phase is much less than running $\mathcal{P}$. As in [4], this efficiency property is defined in an amortized sense, so that verification is more efficient when the same program $\mathcal{P}$ is executed on different data sets. This property enables the use of homomorphic signatures for publicly-verifiable delegation of computation on outsourced data.

The formal definition follows.

Definition 3. *Let* $\mathsf{HomSig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{Eval})$ *be a homomorphic signature scheme for multi-labeled programs.* HomSig satisfies efficient verification *if there exist two additional algorithms* $(\mathsf{VerPrep}, \mathsf{EffVer})$ *such that:*

$\mathsf{VerPrep}(\mathsf{vk}, \mathcal{P})$**:** *on input the verification key* vk *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*.*

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

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

Correctness. *Let* $(\mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$ *be honestly generated keys, and* $(\mathcal{P}_\Delta, m, \sigma)$ *be any program/message/signature tuple with* $\mathcal{P}_\Delta = (\mathcal{P}, \Delta)$ *such*

that $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_\Delta, m, \sigma) = 1$. *Then, for every* $\mathsf{vk}_\mathcal{P} \xleftarrow{\$} \mathsf{VerPrep}(\mathsf{vk}, \mathcal{P})$, $\mathsf{EffVer}(\mathsf{vk}_\mathcal{P}, \Delta, m, \sigma) = 1$ *holds with all but negligible probability.*

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* $\mathsf{vk}_\mathcal{P} \leftarrow \mathsf{VerPrep}(\mathsf{vk}, \mathcal{P})$, *then the time required for* $\mathsf{EffVer}(\mathsf{vk}_\mathcal{P}, \Delta, m, \tau)$ *is* $t' = o(t(n))$.

Notice that in our efficiency requirement, we do not include the time needed to compute $\mathsf{vk}_\mathcal{P}$. This is justified by the fact that, being $\mathsf{vk}_\mathcal{P}$ 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. Namely, the cost of computing $\mathsf{vk}_\mathcal{P}$ is *amortized* over many verifications of the same function on different data sets.

3 From Weakly-Secure to Adaptive-Secure Homomorphic Signatures

In this section we show how to convert a weakly-secure homomorphic signature that works for arithmetic circuits of degree k, into an adaptive-secure one supporting the same class of functionalities. The only restriction is that the message space is expected to be some finite field, e.g., $\mathbb{Z}_p$ for a prime p, that does not depend on the secret key. In the full version we show how to extend these ideas to the case where the messages and the polynomials supported by the homomorphic signature scheme are defined over the integers.

The basic idea behind the conversion is to interpret the message one wants to sign as the free term of a random degree-1 (univariate) polynomial $t(z)$ defined over a finite field. Next, rather than signing m, one signs $(k+1)$ points of this polynomial, e.g., $t(1), \ldots, t(k+1)$, by using $(k+1)$ different secret keys. To homomorphically evaluate a function over such signatures, one executes the Eval algorithm in a point-wise fashion. Interestingly, the homomorphic properties of the underlying signature scheme remains preserved because of analogous properties of polynomials. The formal description of the scheme follows.

Let $\mathsf{HomSig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Ver})$ be a weakly-secure scheme with message space $\mathbb{Z}_p$, our (adaptive-secure) homomorphic signature $\mathsf{HomSig}^* = (\mathsf{KeyGen}^*, \mathsf{Sign}^*, \mathsf{Eval}^*, \mathsf{Ver}^*)$ works as follows.

$\mathsf{KeyGen}^*(1^\lambda, k, \mathcal{L})$**.** Let λ be the security parameter, $k \in \mathbb{N}^+$ be a constant denoting the bound on the degree of the supported polynomials, and $\mathcal{L} \subset \{0,1\}^*$ be a set of admissible labels $\mathcal{L} = \{\tau_1, \ldots, \tau_N\}$, for some $N = \mathsf{poly}(\lambda)$. The algorithms runs $(k+1)$ times $\mathsf{KeyGen}(1^\lambda, k, \mathcal{L})$. Denoting by $(\mathsf{vk}_i, \mathsf{sk}_i)$ the public key/secret key pair obtained from the i-th execution of KeyGen, the algorithm outputs $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_{k+1})$, $\mathsf{vk} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_{k+1})$. The message space $\mathcal{M}$ is $\mathbb{Z}_p$

$\mathsf{Sign}^*(\mathsf{sk}, \Delta, \tau, m)$**.** The signing algorithm takes as input the secret key $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_{k+1})$, a data set identifier $\Delta \in \{0,1\}^*$, a label $\tau \in \mathcal{L}$ and a message $m \in \mathbb{Z}_p$. The signing procedure consists of two main steps. First it generates a random degree-1 (univariate) polynomial $t(z)$ such that $t(0) = m \in \mathbb{Z}_p$.

Second, for $i = 1, \ldots, k+1$, it signs $t(i)$ using $\sigma_i \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_i, \Delta, \tau, t(i))$. In other words, each $t(i)$ is signed with respect to a *different* signing key sk_i. The signing algorithm returns $\sigma = ((\sigma_1, t(1)), \ldots, (\sigma_{k+1}, t(k+1))$.

$\mathsf{Eval}^*(\mathsf{vk}, f, \boldsymbol{\sigma})$**.** The public evaluation algorithm takes as input the public key vk, an arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$ and a vector $\boldsymbol{\sigma}$ of n signatures $\sigma^{(1)}, \ldots, \sigma^{(n)}$ such that $\sigma^{(i)}$ is a $(k+1)$-tuple $((\sigma_1^{(i)}, t^{(i)}(1)), \ldots, (\sigma_{k+1}^{(i)}, t^{(i)}(k+1))$. Eval^* computes a signature $\sigma = ((\sigma_1, t(1)), \ldots, (\sigma_{k+1}, t(k+1)))$, by computing $\sigma_i \leftarrow \mathsf{Eval}(\mathsf{vk}_i, f, (\sigma_i^{(1)}, \ldots, \sigma_i^{(n)}))$ and $t(i) \leftarrow f(t^{(1)}(i), \ldots, t^{(n)}(i))$.

$\mathsf{Ver}^*(\mathsf{vk}, \mathcal{P}_\Delta, m, \sigma)$**.** Let $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$ be a multi-labeled program such that $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$ is an arithmetic circuit of degree $d \leq k$. Let $m \in \mathbb{Z}_p$ and $\sigma = ((\sigma_1, t(1)), \ldots, (\sigma_{k+1}, t(k+1))$.

First of all, Ver^* checks that the signatures on all the values $t(i)$ are correct. To do so, it runs $b_i \leftarrow \mathsf{Ver}(\mathsf{vk}_i, \mathcal{P}_\Delta, t(i), \sigma_i)$, $\forall i = 1, \ldots, k+1$. If $b_1 = \ldots = b_{k+1} = 1$ Ver^* proceeds to the next step, otherwise it stops and returns 0.

So, if the values $t(i)$ in the signature are valid, Ver^* uses these values to interpolate a polynomial $t(z)$ of degree (at most) k. More precisely, this is done as follows: if the degree of the arithmetic circuit f is k, $t(z)$ is interpolated using all the $t(i)$'s; if, on the other hand, f is of degree $d < k$, the algorithm first interpolates $t(z)$ using $t(1), \ldots, t(d+1)$ and then checks that $t(z)$ is correct with respect to $t(d+2), \ldots, t(k+1)$.[5] Finally, Ver^* checks whether $t(0) = m$ or not. Again, if any of the above tests fail the algorithm outputs 0, otherwise it outputs 1.

To complete the description of HomSig^* we give the algorithms for efficient verification:

$\mathsf{VerPrep}^*(\mathsf{vk}, \mathcal{P})$**.** Let $\mathcal{P} = (f, \boldsymbol{\tau})$ be a labeled program for an arithmetic circuit $f \in \mathbb{Z}_p^n \to \mathbb{Z}_p$ with labels $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_n)$. For $i = 1$ to $(k+1)$ the algorithm runs $\mathsf{vk}_{\mathcal{P}}^{(i)} = \mathsf{VerPrep}(\mathsf{vk}_i, \mathcal{P})$ and returns the efficient verification key $\mathsf{vk}_{\mathcal{P}} = (\mathsf{vk}_{\mathcal{P}}^{(1)}, \ldots, \mathsf{vk}_{\mathcal{P}}^{(k+1)})$.

$\mathsf{EffVer}^*(\mathsf{vk}_{\mathcal{P}}, \Delta, m, \sigma)$**.** Let $(\sigma = ((\sigma_1, t(1)), \ldots, (\sigma_{(k+1)}, t(k+1))$. For $i = 1$ to $(k+1)$, the online verification algorithm runs $b_i \leftarrow \mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}^{(i)}, \Delta, t(i), \sigma_i)$. If the $t(i)$'s correctly interpolate to m and $\wedge_{i=1}^{k+1} b_i = 1$, output 1. Otherwise output 0. Notice that if the EffVer provides efficient verification, then EffVer^* has efficient verification as well.

In the following theorem (its proof is in the full version), we show that if HomSig is a weakly-secure scheme, our transformation yields an adaptive-secure homomorphic signature.

Theorem 1. *If* HomSig *is a weakly-secure homomorphic signature scheme for arithmetic circuits of degree* $d \leq k$ *then* HomSig^* *is an adaptive-secure homomorphic signature scheme for the same class of circuits.*

[5] This is done by simply recomputing the interpolated polynomial on points $(d+2), \ldots, (k+1)$.

4 A Weakly-Secure Homomorphic Signature

In this section we describe our construction of homomorphic signatures with efficient verification from leveled multilinear maps. When working with $2k$-linear maps, our scheme can support the evaluation of arithmetic circuits of degree k. The scheme presented in this section is proven weakly-secure under the AP-MDH assumption (Definition 1). This construction can then be turned into an adaptive-secure scheme by either applying our generic transformation of Section 3, or by tailoring our generic technique to this scheme.

Here we describe the scheme using the abstraction of leveled multilinear maps. A discussion about implementing the scheme with graded encodings is given later in this section and more details appear in the full version.

Without loss of generality our scheme works with arithmetic circuits in which addition gates take inputs of the same degree. Notice that any arithmetic circuit $f : \mathbb{F}^n \to \mathbb{F}$ of degree d can be converted into another circuit $\tilde{f} : \mathbb{F}^{n+1} \to \mathbb{F}$ of the same degree d such that $\tilde{f}$ can compute the same function of f. The idea of the transformation is very simple: one first adds to $\tilde{f}$ (say at the end) one additional input wire, labeled by u; then, whenever there is an addition gate taking inputs x_1, x_2 such that $deg(x_1) < deg(x_2)$, one multiplies x_1 by u as many times as needed to obtain a wire x_1' such that $deg(x_1) = deg(x_2)$. Finally, by assigning 1 to the input labeled by u, it is easy to see that $\tilde{f}(m_1, \ldots, m_n, 1) = f(m_1, \ldots, m_n)$. From now on we assume that the circuits used in our scheme have this form.

In what follows we provide a full-detailed description of our construction, which is rather intricate. We refer the reader to the introduction for a more intuitive explanation of our ideas.

To build our scheme we use a regular signature scheme $\Sigma' = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Ver}')$, a pseudorandom function $F : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p^2$ with key space $\mathcal{K}$, and an implementation of leveled multilinear groups whose description is generated by $\mathcal{G}$. Our homomorphic signature scheme $\mathsf{HomSig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Ver})$ works as follows.

$\mathsf{KeyGen}(1^\lambda, k, \mathcal{L})$**.** Let λ be the security parameter, $k \in \mathbb{N}^+$ be a constant denoting the bound on the degree of the supported polynomials, and $\mathcal{L} \subset \{0,1\}^*$ be a set of admissible labels $\mathcal{L} = \{u\} \cup \{\tau_1, \ldots, \tau_N\}$, for some $N = \mathsf{poly}(\lambda)$. Here "u" (which stands for "unity") is a special additional label that is used for the modified arithmetic circuits in which addition gates always take in homogenous monomials. The set of labels is implicitly defining the maximum data set size N supported by the scheme. The key generation algorithm works as follows.

- Generate a key pair $(\mathsf{sk}', \mathsf{vk}') \xleftarrow{\$} \mathsf{KeyGen}'(1^\lambda)$ for the regular signature scheme.
- Choose a random seed $K \xleftarrow{\$} \mathcal{K}$ for the PRF $F_K : \{0,1\}^* \to \mathbb{Z}_p^2$.
- Run $\mathcal{G}(1^\lambda, 2k)$ to generate the description of $(2k)$-linear groups $\mathbb{G}_1, \ldots, \mathbb{G}_{2k}$ of order p, where p is a prime number of roughly λ bits. In the scheme, we use group elements with subscripts to denote the group they live in. Also, for an $h_1 \in \mathbb{G}_1$, we denote by $h_i \in \mathbb{G}_1$ the i-fold graded multiplication of h_1. Analogous notation is used for other group elements.

– Choose random elements $g_1, h_1 \xleftarrow{\$} \mathbb{G}_1$ as well as $N+1$ random values $R_\tau \xleftarrow{\$} \mathbb{G}_1$, $\forall \tau \in \mathcal{L}$.

Finally, output $\mathsf{sk} = (\mathsf{sk}', K)$, $\mathsf{vk} = (\mathsf{vk}', g_1, h_1, \{R_\tau\}_{\tau \in \mathcal{L}})$, and let the message space $\mathcal{M}$ be $\mathbb{Z}_p$.

$\mathsf{Sign}(\mathsf{sk}, \Delta, \tau, m)$**.** The signing algorithm takes as input the secret key $\mathsf{sk} = (\mathsf{sk}', K)$, a data set identifier $\Delta \in \{0,1\}^*$, a label $\tau \in \mathcal{L}$ and a message $m \in \mathbb{Z}_p$. The signing procedure consists of two main steps. First, it uses the pseudorandom function to (re-)derive some common parameters for the dataset Δ and signs the public part of these parameters using the regular signature scheme. Second, it uses the secret part of the parameters for Δ to create the homomorphic component of the signature which is the one strictly bound to (Δ, τ, m). The latter procedure is the core of our technique. We describe it below as a separate subroutine.

– $\mathsf{HomSign}(\mathsf{vk}, a, b, \tau, m)$: this algorithm simply computes $\Lambda_1 = (R_\tau h_1^{-m})^b$, $\Gamma_1 = \Lambda_1^a$, and returns $\nu = (m, \Lambda_1, \Gamma_1)$.

The full signing algorithm proceeds as follows.

1. Derive two integers $(a, b) \leftarrow F_K(\Delta)$ using the pseudorandom function, and compute $A_1 = g_1^a, B_1 = g_1^b, C_1 = g_1^{ab}, T_1 = h_1^a, U_1 = h_1^{ab}$.
2. Run the routine $\mathsf{HomSign}(\mathsf{vk}, a, b, u, 1)$ described above, to compute a triple $\nu_{\Delta,u} = (1, \Lambda_u, \Gamma_u) \in \mathbb{Z}_p \times \mathbb{G}_1^2$. The tuple $\nu_{\Delta,u}$ is essentially the homomorphic component of a signature of "1" with respect to the special label "u" and for the dataset Δ. This signature $\nu_{\Delta,u}$ is needed to perform the homomorphic evaluations on the modified circuits.
3. Let $\mathsf{pp}_\Delta = (\Delta, A_1, B_1, C_1, T_1, U_1, \nu_{\Delta,u})$ be the public parameters of dataset Δ. Then sign pp_Δ using the regular signature scheme, i.e., compute $\sigma_\Delta \leftarrow \mathsf{Sign}'(\mathsf{sk}', \mathsf{pp}_\Delta)$.
4. Run $\mathsf{HomSign}(\mathsf{vk}, a, b, \tau, m)$ to generate a tuple $\nu = (m, \Lambda_1, \Gamma_1) \in \mathbb{Z}_p \times \mathbb{G}_1^2$.

Finally, the signing algorithm returns the signature $\sigma = (\mathsf{pp}_\Delta, \sigma_\Delta, \nu)$. Observe that when generating many signatures for the same dataset Δ the steps 1–3 can be executed only once.

$\mathsf{Eval}(\mathsf{vk}, f, \boldsymbol{\sigma})$**.** The public evaluation algorithm takes as input the public key vk, an arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$ and a vector $\boldsymbol{\sigma}$ of n signatures $\sigma^{(1)}, \ldots, \sigma^{(n)}$ such that $\sigma^{(i)} = (\mathsf{pp}_\Delta^{(i)}, \sigma_\Delta^{(i)}, \nu_i)$ for $i = 1, \ldots, n$. Eval computes a signature $\sigma = (\mathsf{pp}_\Delta, \sigma_\Delta, \nu)$ as follows.

First, set $\mathsf{pp}_\Delta = \mathsf{pp}_\Delta^{(1)}$ and $\sigma_\Delta = \sigma_\Delta^{(1)}$. Namely, we take the common parameters of the first signature in the vector. Observe that our notion of evaluation correctness works for signatures in the same data set, i.e., all these signatures are supposed to share the same parameters.

In the second stage, Eval computes the homomorphic component ν by homomorphically evaluating the circuit f over the values $\{\nu_i\}_{i=1}^n$. To do so, it proceeds over f gate by gate.

At every gate f_g, given two values ν_1, ν_2 (or a value ν_1 and a constant $c \in \mathbb{Z}_p$), Eval runs the algorithm $\nu \leftarrow \mathsf{GateEval}(\mathsf{vk}, \mathsf{pp}_\Delta, f_g, \nu_1, \nu_2)$ described below that returns a new value ν, 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 value ν obtained by running $\mathsf{GateEval}$ on such last gate. On input $\nu_1 = (m_1, \Lambda_i^{(1)}, \Gamma_i^{(1)}) \in \mathbb{Z}_p \times \mathbb{G}_i^2$ and $\nu_2 = (m_2, \Lambda_j^{(2)}, \Gamma_j^{(2)}) \in \mathbb{Z}_p \times \mathbb{G}_j^2$, $\mathsf{GateEval}(\mathsf{vk}, \mathsf{pp}_\Delta, f_g, \nu_1, \nu_2)$ proceeds as follows. For an *addition* gate f_+, it computes $m = m_1 + m_2$, $\Lambda_i = \Lambda_i^{(1)} \cdot \Lambda_i^{(2)}$, and $\Gamma_i = \Gamma_i^{(1)} \cdot \Gamma_i^{(2)}$. For a *multiplication-by-constant* gate $f_\times$ and constant $c \in \mathbb{Z}_p$, it computes $m = c \cdot m_1$, $\Lambda_i = (\Lambda_i^{(1)})^c$, and $\Gamma_i = (\Gamma_i^{(1)})^c$. For a *multiplication* gate $f_\times$, it computes $m = m_1 \cdot m_2$, $\Lambda_d = e(\Lambda_i^{(1)}, \Gamma_j^{(2)}) \cdot e(\Lambda_i^{(1)}, U_j^{m_2}) \cdot e(U_i^{m_1}, \Lambda_j^{(2)})$, and $\Gamma_d = e(\Gamma_i^{(1)}, \Gamma_j^{(2)}) \cdot e(\Gamma_i^{(1)}, U_j^{m_2}) \cdot e(U_i^{m_1}, \Gamma_j^{(2)})$.

$\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_\Delta, m, \sigma)$**.** Let $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$ be a multi-labeled program such that $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$ is an arithmetic circuit of degree $d \leq k$. Let $m \in \mathbb{Z}_p$ and $\sigma = (\mathsf{pp}_\Delta, \sigma_\Delta, \nu)$ be a signature with $\nu = (m, \Lambda_d, \Gamma_d) \in \mathbb{Z}_p \times \mathbb{G}_d^2$. First, run $\mathsf{Ver}'(\mathsf{vk}', \mathsf{pp}_\Delta, \sigma_\Delta)$ to check that σ_Δ is a valid signature on pp_Δ for the same Δ taken as input by the verification algorithm. If σ_Δ is valid, then proceed as follows. Otherwise, stop and return 0 (reject).

Use the graded maps to evaluate the circuit f on the values $(R_{\tau_1}, \ldots, R_{\tau_n})$. Namely, replace additions in f (for inputs of degree i) by the group operation in $\mathbb{G}_i$, whereas a multiplication in f, with inputs of degree i and j respectively, is replaced by evaluating the graded map $e_{i,j}$. We compactly denote this operation as $R = f(R_{\tau_1}, \ldots, R_{\tau_n}) \in \mathbb{G}_d$. Next, output 1 only if the following two equations are satisfied:

$$e(R \cdot h_d^{-m}, g_d^{a^{d-1}b^d}) = e(\Lambda_d, g_d) \tag{1}$$

$$e(\Lambda_d, A_1) = e(\Gamma_d, g_1) \tag{2}$$

Finally, to complete the description of HomSig we give the algorithms for efficient verification:

$\mathsf{VerPrep}(\mathsf{vk}, \mathcal{P})$**.** Let $\mathcal{P} = (f, \boldsymbol{\tau})$ be a labeled program for an arithmetic circuit $f \in \mathbb{Z}_p^n \to \mathbb{Z}_p$ with labels $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_n)$. The algorithm computes $R = f(R_{\tau_1}, \ldots, R_{\tau_n}) \in \mathbb{G}_d$, h_d, $g_d^{a^{d-1}b^d} = e(C_{d-1}, B_1)$, and returns the concise verification key $\mathsf{vk}_\mathcal{P} = (\mathsf{vk}', g_1, h_d, g_d^{a^{d-1}b^d}, R)$.

$\mathsf{EffVer}(\mathsf{vk}_\mathcal{P}, \Delta, m, \sigma)$**.** The online verification is basically the same as Ver except that the values $R, h_d, g_d^{a^{d-1}b^d}$ have been already computed in the off-line phase and are now part of the online algorithm's input. Notice that the computational complexity of the online verification depends only on the complexity of computing the group operations and the bilinear maps in equations (1), (2). Using current graded encoding schemes, the cost essentially becomes $\mathsf{poly}(k, \log N)$ which is much less than the cost of evaluating an N-variate polynomial of degree k.

It is easy to see that running the combination of $\mathsf{VerPrep}$ and EffVer produces the same result as running Ver.

Very intuitively, the correctness of the scheme follows by that, for any operation $+, \times$, $\mathsf{GateEval}$ preserves the form of the signatures, i.e., $\Lambda_i = (R h_i^{-m})^{a^{i-1}b^i}$ and $\Gamma_i = \Lambda_i^a$.

In the following theorem we prove that HomSig is a weakly-secure homomorphic signature scheme. For lack of space, the proof of security and a formal proof of correctness appear in the full version.

Theorem 2. *If Σ' is an unforgeable signature scheme, F is a pseudorandom function, and $\mathcal{G}$ is the generator of $2k$-linear groups such that the $2k$-APMDH assumption holds for $\mathcal{G}$, then* HomSig *is a weakly-secure homomorphic signature scheme for arithmetic circuits of degree k.*

Achieving Adaptive Security. In order to achieve adaptive security for the scheme described above, we discuss three different approaches. The first one is to apply our generic transformation of Section 3. In the transformed scheme, both public/secret keys and the signatures are longer by a factor of d, that for the class of functions considered in this work is assumed to be independent of n. As a second possibility, we exploit the specific structure of our weakly-secure scheme, and show a more optimized transformation which avoids increasing the size of public and secret keys, i.e., they remain of the same size as in HomSig. Finally, as a third possibility, we show that, under a stronger, interactive variant of the APMDH assumption, the scheme HomSig is by itself adaptive-secure. The optimized transformation and the adaptive security of HomSig appear in the full version of our work.

Instantiating the Scheme with Graded Encodings. In the full version of our paper we show how to translate the scheme presented above to the setting of graded encodings [20,17]. Here we discuss the changes incurred by our scheme to accommodate the differences between multilinear maps and (known) graded encoding schemes. Recall that graded encodings can be randomized. In addition: (1) the ring R in which the encoded values live is not public, i.e., the order p of the encoding sets S_i may not be publicly known (although a lower bound on p is public); (2) one cannot (publicly) encode arbitrary elements "in the exponent"; (3) in order for the zero-test to work properly, one can support only a bounded number of operations over the encodings. To address the first difference, our scheme signs messages that are integers within a certain bound B, and as the class of admissible functions we consider N-variate polynomials of constant degree k over the integers. We can then bound the size of all reachable outputs (obtained by applying an admissible f on integers in $\mathbb{Z}_B$) – say it is B^* – and finally we instantiate the parameters of the graded encoding scheme accordingly so that the order p of the ring is such that $p > B^*$. For the second difference, we note that graded encodings allow one to encode arbitrary elements with the knowledge of a trapdoor which, in our case, can be made available to the signer. In the key generation we let the signer use this procedure to publish level-1 encodings of the $\log B^*$ powers of 2 (i.e., the equivalent of $h_1^{2^j}$). This way, upon verification, an encoding of m (i.e., h_d^m) can be obtained by adding up the encodings of the appropriate powers of 2, according to the bit-decomposition of m (i.e., $h_d^m = e(\prod_{j:m_j=1} h_1^{2^j}, h_{i-1})$). This operation can be done by "consuming the noise" of at most $\log B^*$ additions. To address the third difference, we note that the solutions to (1) and (2) already provide a bound on the maximum

number of operations (additions and multiplications) that will be performed over the encodings when running the homomorphic evaluation algorithm. Using such bounds it is then possible to take appropriately large parameters of the graded encodings that can accommodate this number of operations.

Acknowledgements. The research of Dario Fiore has been partially supported by the European Commission's Seventh Framework Programme Marie Curie Cofund Action AMAROUT II (grant no. 291803), and by the Madrid Regional Government under project PROMETIDOS-CM (ref. S2009/TIC1465). The work of Bogdan Warinschi has been supported in part by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO, by EPSRC via grant EP/H043454/1, and has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement 609611 (PRACTICE).

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. Springer, June 2009.
2. 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. Springer, Mar. 2011.
3. N. Attrapadung, B. Libert, and T. Peters. Computing on authenticated data: New privacy definitions and constructions. In X. Wang and K. Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 367–385. Springer, Dec. 2012.
4. M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13*, pages 863–874. ACM Press, Nov. 2013.
5. G. Barthe, E. Fagerholm, D. Fiore, J. Mitchell, A. Scedrov, and B. Schmidt. Automated analysis of cryptographic assumptions in generic group models. In *Advances in Cryptology – CRYPTO 2014*, LNCS. Springer, 2014.
6. 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. Springer, Aug. 2011.
7. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013.
8. 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. Springer, Mar. 2009.
9. 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. Springer, May 2011.
10. D. Catalano and D. Fiore. Practical homomorphic MACs for arithmetic circuits. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 336–352. Springer, May 2013.

11. D. Catalano, D. Fiore, R. Gennaro, and L. Nizzardo. Generalizing homomorphic macs for arithmetic circuits. In *PKC 2014: 17th International Workshop on Theory and Practice in Public Key Cryptography*, LNCS. Springer, 2014.
12. D. Catalano, D. Fiore, R. Gennaro, and K. Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 680–699. Springer, Mar. 2013.
13. 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. Springer, May 2011.
14. 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. Springer, May 2012.
15. 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. Springer, Aug. 2010.
16. 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. Springer, Aug. 2011.
17. J.-S. Coron, T. Lepoint, and M. Tibouchi. Practical multilinear maps over the integers. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 476–493. Springer, Aug. 2013.
18. D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 12*, pages 501–512. ACM Press, Oct. 2012.
19. 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. Springer, May 2012.
20. S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, May 2013.
21. 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. Springer, Aug. 2010.
22. R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, May 2013.
23. 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. Springer, May 2010.
24. R. Gennaro and D. Wichs. Fully homomorphic message authenticators. In K. Sako and P. Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 301–320. Springer, Dec. 2013.
25. A. Langlois, D. Stehle, and R. Steinfeld. GGHLite: More efficient multilinear maps from ideal lattices. In *Advances in Cryptology – Eurocrypt 2014*, LNCS. Springer, 2014.
26. S. Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.
27. 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. Springer, Mar. 2012.