Programmable Hash Functions go Private: Constructions and Applications to (Homomorphic) Signatures with Shorter Public Keys*

Dario Catalano[1], Dario Fiore[2], and Luca Nizzardo[2]

[1] Dipartimento di Matematica e Informatica, Università di Catania, Italy.
catalano@dmi.unict.it
[2] IMDEA Software Institute, Madrid, Spain.
{dario.fiore, luca.nizzardo}@imdea.org

Abstract. We introduce the notion of asymmetric programmable hash functions (APHFs, for short), which adapts Programmable Hash Functions, introduced by Hofheinz and Kiltz at Crypto 2008, with two main differences. First, an APHF works over bilinear groups, and it is asymmetric in the sense that, while only *secretly* computable, it admits an isomorphic copy which is publicly computable. Second, in addition to the usual programmability, APHFs may have an alternative property that we call *programmable pseudorandomness*. In a nutshell, this property states that it is possible to embed a pseudorandom value as part of the function's output, akin to a random oracle. In spite of the apparent limitation of being only secretly computable, APHFs turn out to be surprisingly powerful objects. We show that they can be used to generically implement both regular and linearly-homomorphic signature schemes in a simple and elegant way. More importantly, when instantiating these generic constructions with our concrete realizations of APHFs, we obtain: (1) the *first* linearly-homomorphic signature (in the standard model) whose public key is *sub-linear* in both the dataset size and the dimension of the signed vectors; (2) short signatures (in the standard model) whose public key is shorter than those by Hofheinz-Jager-Kiltz from Asiacrypt 2011, and essentially the same as those by Yamada, Hannoka, Kunihiro, (CT-RSA 2012).

1 Introduction

Programmable Hash Functions. Programmable Hash Functions (PHFs) were introduced by Hofheinz and Kiltz [26] as an information theoretic tool to "mimic" the behavior of a random oracle in finite groups. In a nutshell, a PHF H is an efficiently computable function that maps suitable inputs (e.g., binary strings) into a group $\mathbb{G}$, and can be generated in two different, indistinguishable, ways. In the standard modality, H hashes inputs X into group elements $\mathsf{H}(X) \in \mathbb{G}$. When generated in trapdoor mode, a trapdoor allows one to express every output in terms of two (user-specified) elements $g, h \in \mathbb{G}$, i.e., one can compute two integers a_X, b_X such that $\mathsf{H}(X) = g^{a_X} h^{b_X}$. Finally, H is programmable in the sense that it is possible to program the behavior of H so that its outputs contain (or not) g with a certain probability. More precisely, H is said (m, n)-*programmable* if for all disjoint sets of inputs $\{X_1, \ldots, X_m\}$ and $\{Z_1, \ldots, Z_n\}$, the joint probability that $\forall i, a_{X_i} = 0$ and $\forall j, a_{Z_j} \neq 0$ is significant (e.g., $1/\mathsf{poly}(\lambda)$). Programmability turns out to be particularly useful in several security proofs. For instance, consider a security proof where a signature on $\mathsf{H}(X)$ can be simulated as long as $a_X = 0$ (i.e., g does not appear) while a forgery on $\mathsf{H}(Z)$ can be successfully used if $a_Z \neq 0$ (i.e., g does appear). Then one could rely on an $(m, 1)$-programmability of H to "hope" that all the queried messages $X_1, \ldots, X_m$ are simulatable, i.e., $\forall i, a_{X_i} = 0$, while the forgery message Z is not, i.e., $a_Z \neq 0$. PHFs essentially provide a nice abstraction of the so-called partitioning technique used in many cryptographic proofs.

* This article is based on an earlier article which appears in the proceedings of CRYPTO 2015, © IACR 2015.

1.1 Our Contribution

Asymmetric Programmable Hash Functions. We introduce the notion of *asymmetric programmable hash functions* (APHFs) which modifies the original notion of PHFs [26] in two main ways. First, an APHF H maps inputs into a *bilinear* group $\mathbb{G}$ and is only *secretly computable*. At the same time, an isomorphic copy of H can be *publicly computed* in the target group $\mathbb{G}_T$, i.e., anyone can compute $e(\mathsf{H}(X), g)$.[3] Second, when generated in trapdoor mode, for two given group elements $g, h \in \mathbb{G}$ such that $h = g^z$, the trapdoor allows one to write every $\mathsf{H}(X)$ as $g^{c_X(z)}$ for a degree-d polynomial $c_X(z)$.

We define two main programmability properties of APHFs. The first one is an adaptation of the original programmability notion, and it says that H is (m, n, d)-programmable if it is (m, n)-programmable as before except that, instead of looking at the probability that $a_X = 0$, one now looks at whether $c_{X,0} = 0$, where $c_{X,0}$ is the coefficient of the degree-0 term of the polynomial $c_X(\cdot)$ obtained using the trapdoor.[4] The second programmability property is new and is called *programmable pseudo-randomness*. Roughly speaking, programmable pseudo-randomness says that one can program H so that the values $g^{c_{X,0}}$ look random to any polynomially-bounded adversary who observes the public hash key and the outputs of H on a set of adaptively chosen inputs. This functionality turns out to be useful in security proofs where one needs to cancel some random values for simulation purposes (we explain this in slightly more detail later in the introduction). In other words, programmable pseudo-randomness provides another random-oracle-like property for standard model hash functions, that is to "hide" a PRF inside the hash function. This is crucial in our security proofs, and we believe it can have further applications.

Applications. In principle, secretly computable PHFs seem less versatile than regular PHFs. In this work, however, we show that, for applications such as digital signatures, APHFs turn out to be *more* powerful than their publicly computable counterparts. Specifically, we show how to use APHFs to realize both *regular* and *linearly-homomorphic* signatures secure in the standard model. Next, we show efficient realizations of APHFs that, when plugged in our generic constructions, yield new and existing schemes that improve the state-of-the-art in the following way. First, we obtain the *first* linearly homomorphic signature scheme, secure in the standard model, achieving a public key which is *sub-linear* in both the dataset size and the dimension of the signed vectors. Second, we obtain regular signature schemes, matching the efficiency of the ones in [**?**], thus providing the shortest signatures in the standard model with a public key shorter than in [25].

In the following we elaborate more on these solutions.

Linearly-Homomorphic Signatures with Short Public Key in the Standard Model. Imagine a user Alice stores one or more datasets $D_1, D_2, \ldots, D_\ell$ on a cloud server. Imagine also that some other user, Bob, is allowed to perform queries over Alice's datasets, i.e., to compute one or more functions $F_1, \ldots, F_m$ over any D_i. The crucial requirement here is that Bob wants to be ensured about the correctness of the computation's results $F_j(D_i)$, even if the server is not trusted. An obvious way to do this (reliably) is to ask Alice to sign all her data $D_i = m_1^{(i)}, \ldots, m_N^{(i)}$. Later, Bob can check the validity of the computation by (1) downloading the full dataset locally, (2) checking all the signatures and (3) redoing the computation from scratch. Efficiency-wise, this solution is clearly undesirable in terms of bandwidth, storage (Bob has to download and store potentially large amount of data) and computation (Bob has to recompute everything on his own).

[3] Because of such asymmetric behavior we call these functions "asymmetric".
[4] For $d = 1$, this is basically the same programmability of [26].

A much better solution comes from the notion of homomorphic signatures [10]. These allow to overcome the first issue (bandwidth) in a very elegant way. Using such a scheme, Alice can sign $m_1, \ldots, m_N$, thus producing signatures $\sigma_1, \ldots, \sigma_N$, which can be verified exactly as ordinary signatures. In addition, the homomorphic property provides the extra feature that, given $\sigma_1, \ldots, \sigma_N$ and some function $F : \mathcal{M}^N \to \mathcal{M}$, one can compute a signature $\sigma_{F,y}$ on the value $y = F(m_1, \ldots, m_N)$ *without* knowledge of the secret signing key sk. In other words, for a set of signed messages and any function F, it is possible to provide $y = F(m_1, \ldots, m_N)$ along with a signature $\sigma_{F,y}$ vouching for the correctness of y. The security notion of homomorphic signatures guarantees that creating a signature σ_{F,y^*} for a $y^* \neq F(m_1, \ldots, m_N)$ is computationally hard, unless one knows sk.

To solve the second issue and allow Bob to *verify efficiently* such signatures (i.e., by spending less time than that required to compute F), one can use *homomorphic signatures with efficient verification*, a notion recently introduced in [16].

The notion of homomorphic signature was first introduced by Johnson *et al.* [28]. Since then several schemes have been proposed. The first schemes were homomorphic only for linear functions over vector spaces [9,20,1,11,14,15,18,2,13,3,30] and have nice applications to network coding and proofs of retrievability. More recent works proposed realizations that can support more expressive functionalities such as polynomials [10,16] or general circuits of bounded polynomial depth [**?**,12].

Despite the significant research work in the area, it is striking that *all* the existing homomorphic signature schemes that are proven secure in the standard model [1,14,15,18,2,3,30,16,**?**,12] suffer from a public key that is *at least linear* in the size N of the signed datasets. On one hand, the cost of storing such large public key can be, in principle, amortized since the key can be re-used for multiple datasets. On the other hand, this limitation still represents a challenging open question from both a theoretical and a practical point of view. From a practical perspective, a linear public key might be simply unaffordable by a user Bob who has limited storage capacity. From a theoretical point of view, considered the state-of-the-art, it seems unclear whether achieving a standard-model scheme with a key of length $o(N)$ is possible at all. Technically speaking, indeed, all these schemes in the standard model somehow rely on a public key as large as one dataset for simulation purposes. This essentially hints that any solution for this problem would require a novel proof strategy.

Our Contribution. We solve the above open problem by proposing the *first* standard-model homomorphic signature scheme that achieves a public key whose size is *sub-linear* in the maximal size N of the supported datasets. Slightly more in detail, we show how to use APHFs in a generic fashion to construct a linearly-homomorphic signature scheme based on bilinear maps that can sign datasets, each consisting of up to N vectors of dimension T. The public key of our scheme mainly consists of the public hash keys of two APHFs. By instantiating these using (one of) our concrete realizations we obtain a linearly-homomorphic signature with a public key of length $O(\sqrt{N} + \sqrt{T})$. We stress that ours is also the *first* linearly-homomorphic scheme where the public key is sub-linear in the dimension T of the signed vectors. Concretely, if one considers applications with datasets of 1 million of elements and a security parameter of 128bits, previous solutions (e.g., [15,2]) require a public key of at least 32 MB, whereas our solution simply works with a public key below 100 KB.

On the Power of Secretly-Computable PHFs. The main technical idea underlying this result is a new proof technique that builds on *asymmetric hash functions with programmable pseudo-randomness*. We illustrate the technique via a toy example inspired by our linearly-homomorphic signature scheme. The scheme works over asymmetric bilinear groups $\mathbb{G}_1, \mathbb{G}_2$, and with an APHF $\mathsf{H} : [N] \to \mathbb{G}_1$ that has programmable pseudo-randomness w.r.t. $d = 1$. To sign a *random* message

$M \in \mathbb{G}_1$ w.r.t. a label τ, one creates the signature

$$S = (\mathsf{H}(\tau) \cdot M)^{1/z}$$

where z is the secret key. The signature is linearly-homomorphic – $S_1 S_2 = (\mathsf{H}(\tau_1)\mathsf{H}(\tau_2)M)^{1/z}$, for $M = M_1 M_2$ – and it can be efficiently checked using a pairing – $e(S, g_2^z) = \prod_i e(\mathsf{H}(\tau_i), g_2) e(M, g_2)$ – and by relying on that $e(\mathsf{H}(\cdot), g_2)$ is publicly computable.

The first interesting thing to note is that having H *secretly* computable is necessary: if H is public the scheme could be easily broken, e.g., choose $M^* = \mathsf{H}(\tau)^{-1}$. Let us now show how to prove its security assuming that we want to do a reduction to the following assumption: given g_1, g_2, g_2^z, the challenge is to compute $W^{1/z} \in \mathbb{G}_1$ for $W \neq 1$ of adversarial choice. Missing g_1^z seems to make hard the simulation of signatures since $M, S \in \mathbb{G}_1$. However, we can use the trapdoor generation of H for $d = 1$ (that for asymmetric pairings takes $g_1, h_1 = g_1^{y_1}, g_2, h_2 = g_2^{y_2}$ and allows to express $\mathsf{H}(X) = g_1^{c_X(y_1, y_2)}$), by plugging $h_1 = 1, h_2 = g_2^z$. This allows to write every output as $\mathsf{H}(\tau) = g_1^{c_\tau(z)} = g_1^{c_{\tau,0} + c_{\tau,1} z}$. Every signing query with label τ is simulated by setting $M_\tau = g^{-c_{\tau,0}}$ and $S_\tau = (g_1^{c_{\tau,1}})$. The signature is correctly distributed since (1) $S_\tau = (\mathsf{H}(\tau) \cdot M_\tau)^{1/z}$, and (2) M_τ looks random thanks to the programmable pseudo-randomness of H. To conclude the proof, assume that the adversary comes up with a forgery M^*, S^* for label τ^* such that τ^* was already queried, and let $\hat{S}, \hat{M}$ be the values in the simulation of the signing query for τ^*. Now, $\hat{S} = (\mathsf{H}(\tau^*) \cdot \hat{M})^{1/z}$ holds by correctness, while $S^* = (\mathsf{H}(\tau^*) \cdot M^*)^{1/z}$ holds for $M^* \neq \hat{M}$ by definition of forgery. Then $(M^*/\hat{M}, S^*/\hat{S})$ is clearly a solution to the above assumption. This essentially shows that we can sign as many M's as the number of τ's, that is N. And by using our construction $\mathsf{H} = \mathsf{H}_{\mathsf{sqrt}}$ this is achievable with a key of length $O(\sqrt{N})$. Let us stress that the above one is an incomplete proof sketch, that we give only to illustrate the core ideas of using programmable pseudo-randomness. We defer the reader to Section 5 for a precise description of our signature scheme and its security proof.

Short Signatures from Bilinear Maps in the Standard Model. Hofheinz and Kiltz [26] proposed efficient realizations of PHFs, and showed how to use them to obtain black-box proofs of several cryptographic primitives. Among these applications, they use PHFs to build generic, standard-model, signature schemes from the Strong RSA problem and the Strong q-Diffie Hellman problem. Somewhat interestingly, these schemes (in particular the ones over bilinear groups) can enjoy very short signatures. The remarkable contribution of the generic construction in [26] is that signatures can be made short by reducing the size ρ of the randomness used (and included) in the signature so that ρ can go beyond the birthday bound. Precisely, by using an $(m, 1)$-programmable hash function, m can control the size of the randomness so that the larger is m, the smaller is the randomness. However, although this would call for $(m, 1)$-PHFs with a large m, the original work [26] described PHFs realizations that are only $(2, 1)$-programmable.[5]

Later, Hofheinz, Jager and Kiltz [25] showed constructions of $(m, 1)$-PHFs for any $m \geq 1$. By choosing a larger m, these new PHFs realizations yield the shortest known signatures in the standard model. On the negative side, however, this also induces much larger public keys. For instance, to obtain a signature of 302 bits from bilinear maps, they need a public key of more than 8MB. The reason of such inefficiency is that their realizations of (deterministic) $(m, 1)$-PHFs have keys of length $O(m^2 \ell)$, where ℓ is the bit size of the inputs. In a subsequent work, Yamada et al. [**?**] improved on this aspect by proposing a signature scheme with a public key of length $O(m\sqrt{\ell})$.

[5] [26] gives also a $(1, \mathsf{poly})$-programmable PHF which allows for different applications.

Signature Scheme		Ass.	Signature Size (bits)			Eff.	Public Key Size (KB)		
				$\lambda = 80$	$\lambda = 128$			$\lambda = 80$	$\lambda = 128$
[32] Waters		CDH	$2\|\mathbb{G}_1\|$	320	512	$2 \times \mathsf{Exp}$	$\|\mathbb{G}_1\| + (\ell+3)\|\mathbb{G}_2\|$	6.5	16.6
[7] Boneh-Boyen		q-SDH	$\|\mathbb{G}_1\| + \|\mathbb{Z}_p\|$	320	512	$1 \times \mathsf{Exp}$	$2\|\mathbb{G}_2\|$	0.08	0.13
[26] $\mathsf{Sig}_{\mathsf{q\text{-}SDH}}[\mathsf{H}_{\mathsf{Wat}}]$	$(m=2)$	q-SDH	$\|\mathbb{G}_1\| + \rho$	230	350	$1 \times \mathsf{Exp}$	$(\ell+1)\|\mathbb{G}_1\| + \|\mathbb{G}_2\|$	3.3	8.3
[25] $\mathsf{Sig}_{\mathsf{q\text{-}SDH}}[\mathsf{H}_{\mathsf{cfs}}]$	$(m=8)$	q-SDH	$\|\mathbb{G}_1\| + \rho$	200	302	$1 \times \mathsf{Exp}$	$(16m^2\ell)\|\mathbb{G}_1\| + \|\mathbb{G}_2\|$	3276.8	8388.7
$\Sigma_{\mathsf{q\text{-}SDH}}[\mathsf{H}_{\mathsf{acfs}}]$	$(m=8)$	q-SDH	$\|\mathbb{G}_1\| + \rho$	200	302	$1 \times \mathsf{Exp}$	$4m\lceil\sqrt{\ell}\rceil(\|\mathbb{G}_1\| + \|\mathbb{G}_2\|) + \|\mathbb{G}_2\|$	25	49.2
[25] $\mathsf{Sig}_{\mathsf{q\text{-}DH}}[\mathsf{H}_{\mathsf{Wat}}, \mathsf{H}_{\mathsf{Wat}}]$	$(m=2)$	q-DH	$\|\mathbb{G}_1\| + \rho$	230	350	$1 \times \mathsf{Exp}$	$(\ell+1)\|\mathbb{G}_1\| + (\rho+1)\|\mathbb{G}_2\|$	4.9	11.2
[25] $\mathsf{Sig}_{\mathsf{q\text{-}DH}}[\mathsf{H}_{\mathsf{cfs}}, \mathsf{H}_{\mathsf{Wat}}]$	$(m=8)$	q-DH	$\|\mathbb{G}_1\| + \rho$	200	302	$1 \times \mathsf{Exp}$	$(16m^2\ell)\|\mathbb{G}_1\| + (\rho+1)\|\mathbb{G}_2\|$	3278.4	8391.6
[?] $\Sigma_{\mathsf{q\text{-}DH}}[\mathsf{H}_{\mathsf{acfs}}, \mathsf{H}_{\mathsf{Wat}}]$	$(m=8)$	q-DH	$\|\mathbb{G}_1\| + \rho$	200	302	$1 \times \mathsf{Exp}$	$4m\lceil\sqrt{\ell}\rceil(\|\mathbb{G}_1\| + \|\mathbb{G}_2\|) + (\rho+1)\|\mathbb{G}_2\|$	26.6	52.2

Table 1. Comparison between different standard-model signature schemes from bilinear maps. The shown values consider: (i) security at both $\lambda = 80$ and $\lambda = 128$ against adversaries seeing up to $q = 2^{30}$ signatures; (ii) an implementation with Type-III pairings where $|\mathbb{G}_1| = p = 2\lambda$ and $|\mathbb{G}_2| = 2|\mathbb{G}_1|$; (iii) messages of 2λ bits so as to provide collision-resistance for λ bits of security; (iv) the size of the randomness $\rho = \log q + \lceil \frac{k}{m} \rceil$ according to the analysis in [26]. We considered an implementation of Waters' scheme which optimizes the signature size. Above Exp denotes the cost of an exponentiation in $\mathbb{G}_1$.The grey rows point out the results from this paper.

Their solution followed a different approach: instead of relying on $(m, 1)$-PHFs they obtained the signature by applying the Naor's transformation [**?**] to a new identity-based key encapsulation mechanism (IBKEM).

Our Results. Our results are mainly two. First, we revisit the generic signature constructions of [26,25] in order to work with $(m, 1, d)$-APHFs. Our generic construction is very similar to that in [26,25], and, as such, it inherits the same property: the larger is m, the shorter can be the randomness.

Second we show the construction of an APHF, $\mathsf{H}_{\mathsf{acfs}}$, that is $(m, 1, 2)$-programmable and has a hash key consisting of $O(m\sqrt{\ell})$ group elements. By plugging $\mathsf{H}_{\mathsf{acfs}}$ into our generic construction we immediately obtain standard-model signatures that achieve the same efficiency as the scheme of Yamada et al. [**?**]. Namely, they are the shortest standard model signature schemes with a public key of length $O(m\sqrt{\ell})$, that concretely allows for signatures of 302bits and a public key of 50KB. One of our two schemes recover the one in [**?**]. In this sense we provide a different conceptual approach to construct such signatures. While Yamada et al. obtained this result by going through an IBKEM, our solution revisits the original Hofheinz-Kiltz's idea of applying programmable functions.

We provide a detailed comparison of the schemes in Table 1.

1.2 Other Related Work

Hanaoka, Matsuda and Schuldt [24] show that there cannot be any black-box construction of a $(\mathsf{poly}, 1)$-PHF. The latter result has been overcome by the recent work of Freire et al. [19] who propose a $(\mathsf{poly}, 1)$-PHF based on multilinear maps. The latter result is obtained by slightly changing the definition of PHFs in order to work in the multilinear group setting. Their $(\mathsf{poly}, 1)$-PHF leads to several applications, notably standard-model versions (over multilinear groups) of BLS signatures, the Boneh-Franklin IBE, and identity-based non-interactive key-exchange. While the notion of PHFs in the multilinear setting of [19] is different from our APHFs (with the main difference being that ours are secretly computable), it is worth noting that the two notions have some relation. As we discuss in Section 3.1 , our APHFs indeed imply PHFs in the *bilinear* setting (though carrying the same degree of programmability).

The idea of using bilinear maps to reduce the size of public keys was used previously by Haralambiev et al. [**?**] in the context of public-key encryption, and by Yamada et al. [**?**] in the context of digital signatures. We note that our solutions use a similar approach in the construction of APHFs, which however also include the important novelty of programmable pseudorandomness, that turned out to be crucial in our proofs for the linearly-homomorphic signature.

2 Preliminaries

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

Notation. We denote with $\lambda \in \mathbb{N}$ a security parameter. We say that a function ϵ is *negligible* if it vanishes faster than the inverse of any polynomial. If S is a set, $x \xleftarrow{\$} S$ denotes the process of selecting x uniformly at random in S. If $\mathcal{A}$ is a probabilistic algorithm, $x \xleftarrow{\$} \mathcal{A}(\cdot)$ denotes the process of running $\mathcal{A}$ on some appropriate input and assigning its output to x. Moreover, for a positive integer n, we denote by $[n]$ the set $\{1, \ldots, n\}$.

2.1 Bilinear Groups and Complexity Assumptions

Let $\lambda \in \mathbb{N}$ be a security parameter and let $\mathcal{G}(1^\lambda)$ be an algorithm which takes as input the security parameter and outputs the description of (asymmetric) bilinear groups $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are groups of the same prime order $p > 2^\lambda$, $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ are two generators, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable, non-degenerate, bilinear map, and there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. We call such an algorithm $\mathcal{G}$ a *bilinear group generator*. In the case $\mathbb{G}_1 = \mathbb{G}_2$, the groups are said *symmetric*, else they are said *asymmetric*.

In our work we rely on specific computational and decisional assumptions in such bilinear groups.

Definition 1 (q-Strong Diffie-Hellman [7]). *Let $\mathcal{G}$ be a generator of asymmetric bilinear groups, let $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$ where g_1, g_2 are two random generators, and let $q = \mathsf{poly}(\lambda)$. We say that the q-Strong Diffie-Hellman Assumption (q-SDH) is ϵ-hard for $\mathcal{G}$ if, for every PPT adversary $\mathcal{A}$,*

$$\mathbf{Adv}^{q\text{-}SDH}_{\mathcal{A}}(\lambda) = \Pr[\mathcal{A}(g_1, g_1^z, \ldots, g_1^{z^q}, g_2, g_2^z) = (c, g_1^{1/(z+c)}) \,\big|\, z \xleftarrow{\$} \mathbb{Z}_p] \leq \epsilon$$

Definition 2 (q-Diffie-Hellman Inversion [6,31]). *Let $\mathcal{G}$ be a generator of asymmetric bilinear groups, let $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$ where g_1, g_2 are two random generators, and let $q = \mathsf{poly}(\lambda)$. We say that the q-Diffie-Hellman Inversion Assumption (q-DHI) is ϵ-hard for $\mathcal{G}$ if, for every PPT adversary $\mathcal{A}$,*

$$\mathbf{Adv}^{q\text{-}DHI}_{\mathcal{A}}(\lambda) = \Pr[\mathcal{A}(g_1, g_1^z, g_2^z, \ldots, g_1^{z^q}, g_2^{z^q}) = g_1^{1/z} \,\big|\, z \xleftarrow{\$} \mathbb{Z}_p] \leq \epsilon$$

It is not hard to see that the above problem is equivalent to the one in which the adversary is given the same input and is challenged to compute the "next power" $g_1^{z^{q+1}}$.

A weaker variant of the q-DHI assumption that we use in some of our proofs is the one in which the adversary receives only g_2, g_2^z in the group $\mathbb{G}_2$. For coherence with [25] we call this assumption q-*Diffie-Hellman* (q-DH).

Definition 3 (External Decisional Diffie-Hellman in $\mathbb{G}_1$). *Let $\mathcal{G}$ be a generator of asymmetric bilinear groups, and let* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$. *We say that the External Decisional Diffie-Hellman Assumption (XDDH) is ϵ-hard in $\mathbb{G}_1$ if, for every PPT adversary $\mathcal{A}$, it holds*

$$\left| \Pr[\mathcal{A}(g_1, g_1^a, g_1^b, g_1^{ab}) = 1 \,|\, a, b \xleftarrow{\$} \mathbb{Z}_p] - \Pr[\mathcal{A}(g_1, g_1^a, g_1^b, g_1^c) = 1 \,|\, a, b, c \xleftarrow{\$} \mathbb{Z}_p] \right| \leq \epsilon$$

Finally, we introduce the following static assumption over asymmetric bilinear groups, that we call "Flexible Diffie-Hellman Inversion" (FDHI) for its similarity to Flexible Diffie-Hellman [23]. As we discuss in Appendix B , FDHI is hard in the generic bilinear group model.

Definition 4 (Flexible Diffie-Hellman Inversion Assumption). *Let $\mathcal{G}$ be a generator of asymmetric bilinear groups, and let* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$. *We say that the Flexible Diffie-Hellman Inversion (FDHI) Assumption is ϵ-hard for $\mathcal{G}$ if for every PPT adversary $\mathcal{A}$:*

$$\mathbf{Adv}_{\mathcal{A}}^{FDHI}(\lambda) = \Pr[W \in \mathbb{G}_1 \backslash \{1_{\mathbb{G}_1}\} \wedge W' = W^{\frac{1}{z}} : (W, W') \leftarrow \mathcal{A}(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}}) \,|\, z, r, v \xleftarrow{\$} \mathbb{Z}_p] \leq \epsilon$$

3 Asymmetric Programmable Hash Functions

In this section we present our new notion of asymmetric programmable hash functions.

Let $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ be a family of asymmetric bilinear groups induced by a bilinear group generator $\mathcal{G}(1^\lambda)$ for a security parameter $\lambda \in \mathbb{N}$.[6] An *asymmetric group hash function* $\mathsf{H} : \mathcal{X} \to \mathbb{G}_1$ consists of three PPT algorithms $(\mathsf{H.Gen}, \mathsf{H.PriEval}, \mathsf{H.PubEval})$ working as follows:

$\mathsf{H.Gen}(1^\lambda, \mathsf{bgp}) \to (\mathsf{sek}, \mathsf{pek})$**:** on input the security parameter $\lambda \in \mathbb{N}$ and a bilinear group description bgp, the PPT key generation algorithm outputs a (secret) evaluation key sek and a (public) evaluation key pek.

$\mathsf{H.PriEval}(\mathsf{sek}, X) \to Y \in \mathbb{G}_1$**:** given the secret evaluation key sek and an input $X \in \mathcal{X}$, the deterministic evaluation algorithm returns an output $Y = \mathsf{H}(X) \in \mathbb{G}_1$.

$\mathsf{H.PubEval}(\mathsf{pek}, X) \to \hat{Y} \in \mathbb{G}_T$**:** on input the public evaluation key pek and an input $X \in \mathcal{X}$, the public evaluation algorithm outputs a value $\hat{Y} \in \mathbb{G}_T$ such that $\hat{Y} = e(\mathsf{H}(X), g_2)$.

For asymmetric hash functions satisfying the syntax described above, we define two different properties that model their possible programmability.

The first property is a generalization of the notion of programmable hash functions of [26,27] to our asymmetric setting (i.e., where the function is only secretly-computatble), and to the more specific setting of bilinear groups. The basic idea is that it is possible to generate the function in a trapdoor-mode that allows one to express every output of H in relation to some specified group elements. In particular, the most useful fact of programmability is that for two arbitrary disjoint sets of inputs $\bar{X}, \bar{Z} \subset \mathcal{X}$, the joint probability that some of these group elements appear in $\mathsf{H}(Z), \forall Z \in \bar{Z}$ and do not appear in $\mathsf{H}(X), \forall X \in \bar{X}$ is significant.

Definition 5 (Asymmetric Programmable Hash Functions). *An asymmetric group hash function* $\mathsf{H} = (\mathsf{H.Gen}, \mathsf{H.PriEval}, \mathsf{H.PubEval})$ *is* $(m, n, d, \gamma, \delta)$-programmable *if there exist an efficient trapdoor generation algorithm* $\mathsf{H.TrapGen}$ *and an efficient trapdoor evaluation algorithm* $\mathsf{H.TrapEval}$ *such that:*

[6] Our definition can be easily adapted to work in symmetric bilinear groups where $\mathbb{G}_1 = \mathbb{G}_2$.

Syntax: $\mathsf{H.TrapGen}(1^\lambda, \mathsf{bgp}, \hat{g}_1, \hat{h}_1, \hat{g}_2, \hat{h}_2) \to (\mathsf{td}, \mathsf{pek})$ *takes as input the security parameter* λ*, bilinear group description* bgp *and group elements* $\hat{g}_1, \hat{h}_1 \in \mathbb{G}_1, \hat{g}_2, \hat{h}_2 \in \mathbb{G}_2$*, and it generates a public hash key* pek *along with a trapdoor* td*.* $\mathsf{H.TrapEval}(\mathsf{td}, X) \to \boldsymbol{c}_X$ *takes as input the trapdoor information* td *and an input* $X \in \mathcal{X}$*, and outputs a vector of integer coefficients* $\boldsymbol{c}_X = (c_0, \ldots, c_{d'}) \in \mathbb{Z}^{d'}$ *of a 2-variate polynomial* $c_X(y_1, y_2)$ *of degree* $\leq d$*.*

Correctness: *For all group elements* $\hat{g}_1, \hat{h}_1 \in \mathbb{G}_1, \hat{g}_2, \hat{h}_2 \in \mathbb{G}_2$ *such that* $\hat{h}_1 = \hat{g}_1^{y_1}$ *and* $\hat{h}_2 = \hat{g}_2^{y_2}$ *for some* $y_1, y_2 \in \mathbb{Z}_p$*, for all trapdoor keys* $(\mathsf{td}, \mathsf{pek}) \xleftarrow{\$} \mathsf{H.TrapGen}(1^\lambda, \hat{g}_1, \hat{h}_1, \hat{g}_2, \hat{h}_2)$*, and for all inputs* $X \in \mathcal{X}$*, if* $\boldsymbol{c}_X \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, X)$*, then*

$$\mathsf{H}(X) = \hat{g}_1^{c_X(y_1, y_2)}$$

Statistically-close trapdoor keys: *For all generators* $\hat{g}_1, \hat{h}_1 \in \mathbb{G}_1, \hat{g}_2, \hat{h}_2 \in \mathbb{G}_2$ *and for all* $(\mathsf{sek}, \mathsf{pek}) \xleftarrow{\$} \mathsf{H.Gen}(1^\lambda)$*,* $(\mathsf{td}, \mathsf{pek}') \xleftarrow{\$} \mathsf{H.TrapGen}(1^\lambda, \hat{g}_1, \hat{h}_1, \hat{g}_2, \hat{h}_2)$*, the distribution of the public keys* pek *and* pek' *is within statistical distance* γ*.*

Well distributed logarithms: *For all* $\hat{g}_1, \hat{h}_1 \in \mathbb{G}_1, \hat{g}_2, \hat{h}_2 \in \mathbb{G}_2$*, all keys* $(\mathsf{td}, \mathsf{pek}) \xleftarrow{\$} \mathsf{H.TrapGen}(1^\lambda, \hat{g}_1, \hat{h}_1, \hat{g}_2, \hat{h}_2)$*, and all inputs* $X_1, \ldots, X_m \in \mathcal{X}$ *and* $Z_1, \ldots, Z_n \in \mathcal{X}$ *such that* $X_i \neq Z_j$ *for all* i, j*, we have*

$$\Pr[c_{X_1,0} = \cdots = c_{X_m,0} = 0 \ \wedge \ c_{Z_1,0}, \ldots, c_{Z_n,0} \neq 0] \geq \delta$$

where $\boldsymbol{c}_{X_i} \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, X_i)$ *and* $\boldsymbol{c}_{Z_j} \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, Z_j)$*, and* $c_{X_i,0}$ *(resp.* $c_{Z_j,0}$*) is the coefficient of the term of degree 0.*

If γ *is negligible and* δ *is noticeable we simply say that* H *is* (m, n, d)*-programmable. Furthermore, if* m *(resp.* n*) is an arbitrary polynomial in* λ*, then we say that* H *is* (poly, n, d)*-programmable (resp.* (m, poly, d)*-programmable). Finally, if* H *admits trapdoor algorithms that satisfy only the first three properties, then* H *is said* simply (d, γ)-programmable*. Note that any* H *that is* $(m, n, d, \gamma, \delta)$*-programmable is also* (d, γ)*-programmable.*

Programmable Pseudo-randomness. The second main programmability property that we define for asymmetric hash functions is quite different from the previous one. It is called *programmable pseudo-randomness*, and very intuitively it says that, when using the hash function in trapdoor mode, it is possible to "embed" a PRF into it. More precisely, the trapdoor algorithms satisfy programmable pseudo-randomness if they allow to generate keys such that even by observing pek and $\mathsf{H}(X)$ for a bunch of inputs X, then the elements $g_1^{c_{X,0}}$ look random. The formal definition follows:

Definition 6 (Asymmetric Hash Functions with Programmable Pseudorandomness). *An asymmetric hash function* $\mathsf{H} = (\mathsf{H.Gen}, \mathsf{H.PriEval}, \mathsf{H.PubEval})$ *has* (d, γ, ϵ)-programmable pseudorandomness *if there exist efficient trapdoor algorithms* $\mathsf{H.TrapGen}, \mathsf{H.TrapEval}$ *that satisfy the properties of syntax, correctness, and* γ*-statistically-close trapdoor keys as in Definition 5, and additionally satisfy the following property with parameter* ϵ*:*

Pseudorandomness: *Let* $b \in \{0, 1\}$ *and let* $\mathbf{Exp}^{PRH\text{-}b}_{\mathcal{A},\mathsf{H}}(\lambda)$ *be the following experiment between an adversary* $\mathcal{A}$ *and a challenger.*

1. *Generate* $\mathsf{bgp} \xleftarrow{\$} \mathcal{G}(1^\lambda)$*, and run* $\mathcal{A}(\mathsf{bgp})$*, that outputs two generators* $h_1 \in \mathbb{G}_1, h_2 \in \mathbb{G}_2$*.*
2. *Compute* $(\mathsf{td}, \mathsf{pek}) \xleftarrow{\$} \mathsf{H.TrapGen}(1^\lambda, g_1, h_1, g_2, h_2)$ *and run* $\mathcal{A}(\mathsf{pek})$ *with access to the following oracle:*

– *If $b = 0$, $\mathcal{A}$ is given $\mathcal{O}(\cdot)$ that on input $X \in \mathcal{X}$ returns $\mathsf{H}(X) = g_1^{c_X(y_1,y_2)}$ and $g_1^{c_{X,0}}$, where $\boldsymbol{c}_X \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, X)$;*
– *If $b = 1$, $\mathcal{A}$ is given $\mathcal{R}(\cdot)$ that on input $X \in \mathcal{X}$ returns $\mathsf{H}(X) = g_1^{c_X(y_1,y_2)}$ and $g_1^{r_X}$, for a randomly chosen $r_X \xleftarrow{\$} \mathbb{Z}_p$ (which is unique for every $X \in \mathcal{X}$).*

3. At the end the adversary outputs a bit b', and b' is returned as the output of the experiment.

Then we say that $\mathsf{H.TrapGen}, \mathsf{H.TrapEval}$ *satisfy pseudo-randomness for ϵ, if for all PPT $\mathcal{A}$*

$$\left|\Pr[\mathbf{Exp}^{PRH\text{-}0}_{\mathcal{A},\mathsf{H}}(\lambda) = 1] - \Pr[\mathbf{Exp}^{PRH\text{-}1}_{\mathcal{A},\mathsf{H}}(\lambda) = 1]\right| \leq \epsilon$$

where the probabilities are taken over all the random choices of $\mathsf{TrapGen}$*, the oracle $\mathcal{R}$ and the adversary $\mathcal{A}$.*

Remark 1 (On the mutual existence of programmability and programmable pseudorandomness). We stress that the two properties of programmability and programmable pseudorandomness defined above are mutually exclusive. Precisely, an APHF can have a pair of trapdoor algorithms $(\mathsf{TrapGen}, \mathsf{TrapEval})$ that admits either $(m, n, d, \gamma, \delta)$-programmability (for non-negligible δ), or (d, γ, ϵ)-programmable pseudorandomness (for negligible ϵ). Intuitively, the reason why the same trapdoor algorithms cannot satisfy both properties is that (m, n, δ, γ)-programmability implies that for any elements $X_1, \ldots, X_m \in \mathcal{X}$ it holds $c_{X_{i,0}} = 0$ with non negligible probability δ. However, if this holds then programmable pseudorandomness can be trivially broken, since $g_1^{c_{X_{i,0}}} = 1$ with non negligible probability δ.

On the other hand, it is quite interesting to observe that the *same* function can enjoy *both* properties through different, appropriate, pairs of trapdoor algorithms. In fact, an asymmetric group hash function can have a pair of trapdoor algorithms $(\mathsf{TrapGen}, \mathsf{TrapEval})$ for which (m, n, δ, γ)-programmability holds, and another pair of trapdoor algorithms $(\mathsf{TrapGen}', \mathsf{TrapEval}')$ for which (d, γ, δ)-programmable pseudorandomness holds. Then, since all trapdoor generations produce keys that are statistically indistinguishable from the real ones it follows that also the two trapdoor modes are statistically indistinguishable. In a nutshell, this means that the same function can be programmed in different modes in different steps of a security proof, a property which turns out to be very useful, for example, in our proofs of Section 5.4.

Other variants of programmability. Here we define two other variants of the programmability notion given in Definition 5.

Weak Programmability. We consider a weak version of the above programmability property in which one fixes at key generation time the n inputs Z_j on which $c_{Z_j,0} \neq 0$.

Definition 7 (Asymmetric Weakly-Programmable Hash Functions). *An asymmetric group hash function* $\mathsf{H} = (\mathsf{H.Gen}, \mathsf{H.PriEval}, \mathsf{H.PubEval})$ *is* weakly $(m, n, d, \gamma, \delta)$*-programmable if there exist efficient trapdoor generation* $\mathsf{H.TrapGen}$ *and trapdoor evaluation* $\mathsf{H.TrapEval}$ *algorithms such that:*

– **Syntax:** $\mathsf{H.TrapGen}(1^\lambda, \mathsf{bgp}, \hat{g}_1, \hat{h}_1, \hat{g}_2, \hat{h}_2, Z_1, \ldots, Z_n) \rightarrow (\mathsf{td}, \mathsf{pek})$ *takes as input the security parameter λ, bilinear group description* bgp*, group elements $\hat{g}_1, \hat{h}_1 \in \mathbb{G}_1, \hat{g}_2, \hat{h}_2 \in \mathbb{G}_2$, and a set of n inputs $Z_1, \ldots, Z_n \in \mathcal{X}$. It generates a public hash key* pek *along with a trapdoor* td*.* $\mathsf{H.TrapEval}(\mathsf{td}, X) \rightarrow \boldsymbol{c}_X$ *works exactly as in Definition 5.*
– *The properties of* correctness *and* statistically-close trapdoor keys *hold as in Definition 5. The property of* well-distributed logarithms *is also the same except that the inputs $Z_1, \ldots, Z_n$ are the ones fixed as input to* $\mathsf{H.TrapGen}$*.*

Remark 2. We remark that for those (deterministic) functions H whose domain $\mathcal{X}$ has polynomial size any weak programmability property for an arbitrary $m = \mathsf{poly}$ trivially holds with $\delta = 1$.

DEGREE-d PROGRAMMABILITY. In our work we also consider a variant of the above definition in which the property of well distributed logarithms is stated with respect to the *degree-d coefficients* of the polynomials generated by $\mathsf{H.TrapEval}$. In this case, we say that H is $(m, n, d, \gamma, \delta)$-*degree-d-programmable*.

3.1 Relation with existing notions

Before describing our realizations of APHFs, we discuss here the relation between our new notion and two existing notions of programmable hash functions: the original one by Hofheinz and Kiltz [26] and its adaptation to the multilinear setting recently proposed by Freire et al. [19].

When working over bilinear groups, the notion of programmable hash functions of [26] is essentially a special case of ours. The main differences are: (1) PHFs are publicly computable, (2) the trapdoor algorithms work with only two generators $\hat{g}, \hat{h}$ and every output of the function can be expressed as a linear function $\hat{g}^a \hat{h}^b$ of these two generators. As we formally state in the following theorem, a standard PHF is an APHF for $d = 1$:

Theorem 1. *Let* $\hat{\mathsf{H}} = (\mathsf{PHF.Gen}, \mathsf{PHF.Eval})$ *be an* (m, n, γ, δ)*-programmable hash function such that* $\hat{\mathsf{H}} : \mathcal{X} \to \mathbb{G}_1$*. Define* $\mathsf{H.Gen} = \mathsf{PHF.Gen}$, $\mathsf{H.PriEval} = \mathsf{PHF.Eval}$ *and (informally)* $\mathsf{H.PubEval} = e(\mathsf{PHF.Eval}, g_2)$*. Then* $\mathsf{H} = (\mathsf{H.Gen}, \mathsf{H.PriEval}, \mathsf{H.PubEval})$ *is an asymmetric* $(m, n, 1, \gamma, \delta)$*-programmable hash function.*

The proof is straightforward and is omitted.

Second, we analyze the relation between asymmetric hash functions and the PHFs in the multilinear setting introduced in [19]. Informally, for a setting of leveled multilinear groups $\mathbb{G}_1, \ldots, \mathbb{G}_\ell$, [19] considers a group hash function $\hat{\mathsf{H}} : \mathcal{X} \to \mathbb{G}_\ell$. Then, $\hat{\mathsf{H}}$ is said (m, n)-programmable if there exist two trapdoor algorithms $\mathsf{PHF.TrapGen}$, $\mathsf{PHF.TrapEval}$ such that: $\mathsf{PHF.TrapGen}(1^\lambda, g_1, \ldots, g_\ell, h)$ takes as input $g_i, h \in \mathbb{G}_1$ with $h \neq 1$ and outputs a trapdoor td and hash key hk; $\mathsf{PHF.TrapEval}(\mathsf{td}, X)$ on input X outputs an integer a_X and an element $B_X \in \mathbb{G}_{\ell-1}$ such that $\mathsf{H}(X) = e(g_1, \ldots, g_\ell)^{a_X} e(B_X, h) \in \mathbb{G}_\ell$. If we consider leveled *bilinear* groups where $\mathbb{G} = \mathbb{G}_1$ and $\mathbb{G}_T = \mathbb{G}_2$, then asymmetric programmable hash functions (for $d \leq 2$) imply PHFs in the (symmetric) bilinear group setting:

Theorem 2. *Let* $\mathbb{G}, \mathbb{G}_T$ *be symmetric bilinear groups, and let* $\mathsf{H} = (\mathsf{H.Gen}, \mathsf{H.PriEval}, \mathsf{H.PubEval})$ *be an asymmetric* $(m, n, 2, \gamma, \delta)$*-programmable hash function such that* $\mathsf{H} : \mathcal{X} \to \mathbb{G}$*. Define* $\mathsf{PHF.Gen} = \mathsf{H.Gen}$ *and* $\mathsf{PHF.Eval} = \mathsf{H.PubEval}$*. Then* $\hat{\mathsf{H}} = (\mathsf{PHF.Gen}, \mathsf{PHF.Eval})$ *is an* (m, n, γ, δ)*-programmable hash function in the bilinear setting.*

The proof is fairly easy. Here we provide a sketch. Basically, by assuming that H is programmable, we have to show two algorithms $\mathsf{PHF.TrapGen}, \mathsf{PHF.TrapEval}$ that satisfy the programmability of $\hat{\mathsf{H}}$ in the bilinear setting:

$\mathsf{PHF.TrapGen}(1^\lambda, g, h)$**:** run $(\mathsf{td}, \mathsf{pek}) \xleftarrow{\$} \mathsf{H.TrapGen}(1^\lambda, g, h)$ and output $(\mathsf{td}, \mathsf{pek})$.
$\mathsf{PHF.TrapEval}(\mathsf{td}, X)$**:** run $\boldsymbol{c}_X \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, X)$ to generate the coefficient of a degree-2 polynomial $c_X(y)$ where $y = DLog_g(h)$. Then output $a_X = c_{X,0}$, and $B_X = g^{c_{X,1}} h^{c_{X,2}}$.

It is easy to see that if $\boldsymbol{c}_X$ is such that $\mathsf{H}(X) = g^{c_{X,0}+c_{X,1}y+c_{X,2}y^2}$ then

$$\hat{\mathsf{H}}(X) = e(\mathsf{H}(X), g) = e(g,g)^{c_{X,0}} e(g^{c_{X,1}+c_{X,2}y}, g^y) = e(g,g)^{a_X} e(B_X, h)$$

Finally, the (m, n, γ, δ)-programmability of $\hat{\mathsf{H}}$ is immediately implied by the well distribution of the discrete logarithms in H for parameters $(m, n, 2, \gamma, \delta)$.

3.2 An Asymmetric Programmable Hash Function based on Cover-Free Sets

In this section we present the construction of an asymmetric hash function, $\mathsf{H}_{\mathsf{acfs}}$, based on cover-free sets. Our construction uses ideas similar to the ones used by Hofheinz, Jager and Kiltz [25] to design a (regular) programmable hash function. Our construction extends these ideas with a technique that allows us to obtain a much shorter public key. Concretely, for binary inputs of size ℓ, the programmable hash function $\mathsf{H}_{\mathsf{cfs}}$ in [25] is $(m, 1)$-programmable with a hash key of length $O(\ell m^2)$. In contrast, our new construction $\mathsf{H}_{\mathsf{acfs}}$ is $(m, 1)$-programmable with a hash key of length $O(m\sqrt{\ell})$. While such improvement is obtained at the price of obtaining the function in the secret-key model, our results of Section 4 show that *asymmetric* programmable hash are still useful to build short bilinear-map signatures, whose efficiency, in terms of signature's and key's length matches that of state-of-the-art schemes [**?**].

Before proceeding with describing our function, below we recall the notion of cover-free sets.

Cover-Free Families. If S, V are sets, we say that S does not cover V if $S \not\supseteq V$. Let T, m, s be positive integers, and let $F = \{F_i\}_{i\in[s]}$ be a family of subsets of $[T]$. A family F is said to be *m-cover-free* over $[T]$, if for any subset $I \subseteq [s]$ of cardinality at most m, then the union $\cup_{i\in I} F_i$ does not cover F_j for all $j \notin I$. More formally, for any $I \subseteq [s]$ such that $|I| \leq m$, and any $j \notin I$, $\cup_{i\in I} F_i \not\supseteq F_j$. Furthermore, we say that F is w-uniform if every subset F_i in the family have size w. In our construction, we use the following fact from [17,29]:

Lemma 1 ([17,29]). *There is a deterministic polynomial time algorithm that, on input integers $s = 2^\ell$ and m, returns w, T, F where $F = \{F_i\}_{i\in[s]}$ is a w-uniform, m-cover-free family over $[T]$, for $w = T/4m$ and $T \leq 16m^2\ell$.*

The Construction of $\mathsf{H}_{\mathsf{acfs}}$. Let $\mathcal{G}(1^\lambda)$ be a bilinear group generator, let $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ be an instance of bilinear group parameters generated by $\mathcal{G}$. Let $\ell = \ell(\lambda)$ and $m = m(\lambda)$ be two polynomials in the security parameter. We set $s = 2^\ell$, $T = 16m^2\ell$, and $w = T/4m$ as for Lemma 1, and define $t = \lceil\sqrt{T}\rceil$. Note that every integer $k \in [T]$ can be written as a pair of integers $(i, j) \in [t] \times [t]$ using some canonical mapping. For the sake of simplicity, sometimes we abuse notation and write $(i, j) \in [T]$ where $i, j \in [t]$.

In the following we describe the asymmetric hash function $\mathsf{H}_{\mathsf{acfs}} = (\mathsf{H.Gen}, \mathsf{H.PriEval}, \mathsf{H.PubEval})$ that maps $\mathsf{H}_{\mathsf{acfs}} : \mathcal{X} \to \mathbb{G}_1$ where $\mathcal{X} = \{0,1\}^\ell$. In particular, every input $X \in \{0,1\}^\ell$ is associated to a set F_i, $i \in [2^\ell]$, by interpreting X as an integer in $\{0, \ldots, 2^\ell - 1\}$ and by setting $i = X + 1$. We call F_X such subset associated to X.

$\mathsf{H.Gen}(1^\lambda, \mathsf{bgp})$**:** for $i = 1$ to t, sample $\alpha_i, \beta_i \xleftarrow{\$} \mathbb{Z}_p$ and compute $A_i = g_1^{\alpha_i}, B_i = g_2^{\beta_i}$. Finally, set $\mathsf{sek} = \{\alpha_i, \beta_i\}_{i=1}^t$, $\mathsf{pek} = \{A_i, B_i\}_{i=1}^t$, and return $(\mathsf{sek}, \mathsf{pek})$.

$\mathsf{H.PriEval}(\mathsf{sek}, X)$**:** first, compute the subset $F_X \subseteq [T]$ associated to $X \in \{0,1\}^\ell$, and then return

$$Y = g_1^{\sum_{(i,j)\in F_X} \alpha_i\beta_j} \in \mathbb{G}_1$$

$\mathsf{H.PubEval}(\mathsf{pek}, X)$**:** let $F_X \subseteq [T]$ be the subset associated to X, and compute

$$\hat{Y} = \prod_{(i,j)\in F_X} e(A_i, B_j) = e(\mathsf{H}(X), g_2)$$

Theorem 3. *Let $\mathcal{G}$ be a bilinear group generator. The hash function $\mathsf{H}_{\mathsf{acfs}}$ described above is an asymmetric $(m, n, d, \gamma, \delta)$-programmable hash function with $n = 1$, $d = 2$, $\gamma = 0$ and $\delta = 1/T$.*

Proof. First, we describe the trapdoor algorithms:

$\mathsf{H.TrapGen}(1^\lambda, \mathsf{bgp}, \hat{g}_1, \hat{h}_1, \hat{g}_2, \hat{h}_2)$**:** first, sample $a_i, b_i \xleftarrow{\$} \mathbb{Z}_p$ for all $i \in [t]$, and pick a random index $\tau \xleftarrow{\$} [T]$. Parse $\tau = (i^*, j^*) \in [t] \times [t]$. Next, set $A_{i^*} = \hat{g}_1 \hat{h}_1^{a_{i^*}}$, $B_{j^*} = \hat{g}_2 \hat{h}_2^{b_{j^*}}$, $A_i = \hat{h}_1^{a_i}$, $\forall i \neq i^*$, and $B_j = \hat{h}_2^{b_j}$, $\forall j \neq j^*$. Finally, set $\mathsf{td} = (\tau, \{a_i, b_i\}_{i=1}^t)$, $\mathsf{pek} = \{A_i, B_i\}_{i=1}^t$, and output $(\mathsf{td}, \mathsf{pek})$.

$\mathsf{H.TrapEval}(\mathsf{td}, X)$**:** first, compute the subset $F_X \subseteq [T]$ associated to $X \in \{0,1\}^\ell$, and then return the coefficients of the degree-2 polynomial

$$c_X(y_1, y_2) = \sum_{(i,j)\in F_X} \alpha_i(y_1) \cdot \beta_j(y_2)$$

where every $\alpha_i(y_1)$ (resp. $\beta_j(y_2)$) is the discrete logarithm of A_i (resp. B_j) in base $\hat{g}_1$ (resp. $\hat{g}_2$), viewed as a degree-1 polynomial in the unknown y_1 (resp. y_2).

Now, we show that the two trapdoor algorithms described above satisfy the four properties of Definition 5. First, syntax and correctness immediately follow by construction. Second, observe that each element A_i (resp. B_j) in pek is a uniformly distributed group element in $\mathbb{G}_1$ (resp. $\mathbb{G}_2$), exactly as in the output of $\mathsf{H.Gen}$, hence $\gamma = 0$. Third, we show that the algorithms allow for well distributed logarithms for the case $n = 1$. Let $X_1, \ldots, X_m, Z \in \mathcal{X}$ such that $Z \neq X_i$ for all i. From the m-cover-free property of F we have that there exist an index $\tau' \in F_Z$ such that $\tau' \notin \cup_{i=1}^m F_{X_i}$. Since τ is chosen uniformly at random in $[T]$, we have that $\tau = \tau'$ with probability $\delta = 1/T$. Now, assume that $\tau' = \tau = (i^*, j^*) \in [t] \times [t]$. Then for all $(i, j) \neq (i^*, j^*)$ it holds that the degree-0 coefficient of $c(y_1, y_2) = \alpha_i(y_1)\beta_j(y_2)$ is $c_0 = 0$, whereas for (i^*, j^*) the degree-0 coefficient of $c^*(y_1, y_2) = \alpha_{i^*}(y_1)\beta_{j^*}(y_2) = (a_{i^*} y_1 + 1)(b_{j^*} y_2 + 1)$, is $c_0^* = 1$. Therefore, we have that $c_{X_i,0} = 0, \forall i \in [m]$ and $c_{Z,0} = 1$ holds with probability δ. □

3.3 An Asymmetric Programmable Hash Function with Small Domain

In this section, we present the construction of an asymmetric hash function, $\mathsf{H}_{\mathsf{sqrt}}$, whose domain is of polynomial size T. $\mathsf{H}_{\mathsf{sqrt}}$ has a public key of length $O(\sqrt{T})$, and it turns out to be very important for obtaining our linearly-homomorphic signature scheme with short public key presented in Section 5. Somewhat interestingly, we show that this new function $\mathsf{H}_{\mathsf{sqrt}}$ satisfies several programmability properties, that make it useful in the context of various security proofs.

Let $\mathcal{G}(1^\lambda)$ be a bilinear group generator, let $T = \mathsf{poly}(\lambda)$ and $t = \lceil \sqrt{T} \rceil$. The hash function $\mathsf{H}_{\mathsf{sqrt}} = (\mathsf{H.Gen}, \mathsf{H.PriEval}, \mathsf{H.PubEval})$ that maps $\mathsf{H}_{\mathsf{sqrt}} : \mathcal{X} \to \mathbb{G}_1$ with $\mathcal{X} = [T]$ is defined as follows.

$\mathsf{H.Gen}(1^\lambda, \mathsf{bgp})$**:** for $i = 1$ to t, sample $\alpha_i, \beta_i \xleftarrow{\$} \mathbb{Z}_p$ and compute $A_i = g_1^{\alpha_i}, B_i = g_2^{\beta_i}$. Finally, set $\mathsf{sek} = \{\alpha_i, \beta_i\}_{i=1}^t$, $\mathsf{pek} = \{A_i, B_i\}_{i=1}^t$, and return$(\mathsf{sek}, \mathsf{pek})$.

$\mathsf{H.PriEval}(\mathsf{sek}, X)$**:** first, write $X \in [T]$ as a pair of integer $(i,j) \in [t] \times [t]$, and then return

$$Y = g_1^{\alpha_i \beta_j} \in \mathbb{G}_1$$

$\mathsf{H.PubEval}(\mathsf{pek}, X)$**:** let $X = (i,j)$. The public evaluation algorithm returns

$$\hat{Y} = e(A_i, B_j) = e(\mathsf{H}(X), g_2)$$

Programmable Pseudorandomness of $\mathsf{H}_{\mathsf{sqrt}}$. Here we show that $\mathsf{H}_{\mathsf{sqrt}}$ satisfies the programmable pseudo-randomness property of Definition 6.

Theorem 4. *Let $\mathbb{G}_1$ be a bilinear group of order p over which the XDDH assumption is ϵ'-hard. Then the asymmetric hash function $\mathsf{H}_{\mathsf{sqrt}}$ described above satisfies $(2,0,\epsilon)$-programmable pseudo-randomness with $\epsilon = T \cdot \epsilon'$. Furthermore, in the case when $h_1 = 1 \in \mathbb{G}_1$ or $h_1 = g_1$, $\mathsf{H}_{\mathsf{sqrt}}$ has $(1,0,\epsilon)$-programmable pseudo-randomness.*

Proof. First, we describe the trapdoor algorithms:

$\mathsf{H.TrapGen}(1^\lambda, g_1, h_1, g_2, h_2)$**:** first, sample $a_i, r_i, s_i, b_i \xleftarrow{\$} \mathbb{Z}_p$ for all $i \in [t]$ and then set $A_i = h_1^{r_i} g_1^{a_i}, B_i = h_2^{s_i} g_2^{b_i}$. Finally, set $\mathsf{td} = (\{a_i, r_i, s_i, b_i\}_{i=1}^t)$, $\mathsf{pek} = \{A_i, B_i\}_{i=1}^t$, and output $(\mathsf{td}, \mathsf{pek})$.
$\mathsf{H.TrapEval}(\mathsf{td}, X)$**:** let $X = (i,j)$, and then return the coefficients of the degree-2 polynomial

$$c_X(y_1, y_2) = (y_1 r_i + a_i)(y_2 s_j + b_j)$$

First, it is easy to see that the two algorithms satisfy the syntax and correctness properties. Also, in the case $h_1 = 1$ (i.e., $y_1 = 0$) or $h_1 = g_1$ (i.e., $y_1 = 1$), we obtain a degree-1 polynomial $c_X(y_2)$. Second, observe that each element A_i (resp. B_i) in pek is a uniformly distributed group element in $\mathbb{G}_1$ (resp. $\mathbb{G}_2$), as in $\mathsf{H.Gen}$, hence $\gamma = 0$. Third, we show that the function satisfies the pseudo-randomness property under the assumption that XDDH holds in $\mathbb{G}_1$. The main observation is that for every $X = (i,j)$, we have $c_{X,0} = a_i b_j$ where all the values b_i are uniformly distributed and information-theoretically hidden to an adversary who only sees pek. In particular, this holds even if $h_1 = 1$.

To prove the pseudo-randomness we make use of Lemma 2 below, which shows that for a uniformly random choice of $\boldsymbol{a}, \boldsymbol{b} \xleftarrow{\$} \mathbb{Z}_p^t, c \xleftarrow{\$} \mathbb{Z}_p^{t \times t}$ the distributions $(g_1^{\boldsymbol{a}}, g_1^{\boldsymbol{a} \cdot \boldsymbol{b}^\top}) \in \mathbb{G}_1^{t \times (t+1)}$ and $(g_1^{\boldsymbol{a}}, g_1^{c}) \in \mathbb{G}_1^{t \times (t+1)}$ are computationally indistinguishable.

Lemma 2. *Let $\boldsymbol{a}, \boldsymbol{b} \xleftarrow{\$} \mathbb{Z}_p^t, c \xleftarrow{\$} \mathbb{Z}_p^{t \times t}$ be chosen uniformly at random. If the XDDH assumption is ϵ'-hard in $\mathbb{G}_1$, then for any PPT $\mathcal{B}$ it holds $|\Pr[\mathcal{B}(g_1^{\boldsymbol{a}}, g_1^{\boldsymbol{a} \cdot \boldsymbol{b}^\top}) = 1] - \Pr[\mathcal{B}(g_1^{\boldsymbol{a}}, g_1^{c}) = 1]| \leq T \cdot \epsilon'$.*

We first show how to use Lemma 2 to prove that $\mathsf{H}_{\mathsf{sqrt}}$ has programmable pseudo-randomness. The proof of Lemma 2 appears slightly below.

Let $\mathcal{A}$ be an adversary that breaks the ϵ-programmable pseudo-randomness of $\mathsf{H}_{\mathsf{sqrt}}$. We construct a simulator $\mathcal{B}$ that can distinguish the two distributions $(g_1^{\boldsymbol{a}}, g_1^{\boldsymbol{a} \cdot \boldsymbol{b}^\top})$ and $(g_1^{\boldsymbol{a}}, g_1^{c})$ described above with advantage greater than ϵ.

$\mathcal{B}$'s input is a tuple $(\boldsymbol{A'}, C) \in \mathbb{G}_1^t \times \mathbb{G}_1^{t \times t}$ and its goal is to decide about the distribution of C. First, $\mathcal{B}$ runs $\mathcal{A}(\mathsf{bgp})$ which outputs the generators h_1, h_2. $\mathcal{B}$ then samples two random vectors $\boldsymbol{r}, \boldsymbol{\beta} \xleftarrow{\$} \mathbb{Z}_p^t$, computes $\boldsymbol{B} = g_2^{\boldsymbol{\beta}} \in \mathbb{G}_2^t$, $\boldsymbol{A} = h_1^{\boldsymbol{r}} \cdot \boldsymbol{A'} \in \mathbb{G}_1^t$, sets $\mathsf{pek} = (\boldsymbol{A}, \boldsymbol{B})$, and runs $\mathcal{A}(\mathsf{pek})$ Next,

for every oracle query (i,j) made by $\mathcal{A}$, $\mathcal{B}$ simulates the answer by returning to $\mathcal{A}$: $\mathsf{H}(i,j) = A_i^{\beta_j}$ and $C_{i,j}$. It is easy to see that if $C = g_1^{\boldsymbol{a}\cdot\boldsymbol{b}^\top}$ then $\mathcal{B}$ is perfectly simulating $\mathbf{Exp}^{PRH\text{-}0}_{\mathcal{A},\mathsf{H}_{\mathsf{sqrt}}}$, otherwise, if C is random and independent, then $\mathcal{B}$ is simulating $\mathbf{Exp}^{PRH\text{-}1}_{\mathcal{A},\mathsf{H}_{\mathsf{sqrt}}}$. As a final note, we observe that the above proof works even in the case $h_1 = 1$. □

Proof (Proof of Lemma 2). To prove the above lemma, we define $T+1$ hybrid distributions as follows. Let $\boldsymbol{a}, \boldsymbol{b} \xleftarrow{\$} \mathbb{Z}_p^t$ and $c \xleftarrow{\$} \mathbb{Z}_p^{t\times t}$ be randomly chosen. For every $0 \le k \le T$ we define the matrix $\mathcal{M}_k \in \mathbb{G}_1^{t\times t}$ by specifying the value $\mathcal{M}_k[i,j]$ of each entry $(i,j) \in [t]\times[t]$ of the matrix. For every $k' \in [T]$, let $(i,j) \in [t]\times[t]$ be such that $k' = j + (i-1)t$. Then:

- If $k' \le k$, $\mathcal{M}_k[i,j] = g_1^{c_{i,j}}$,
- If $k' > k$, $\mathcal{M}_k[i,j] = g_1^{a_i b_j}$.

Notice that $\mathcal{M}_0 = g_1^{\boldsymbol{a}\cdot\boldsymbol{b}^\top}$ while $\mathcal{M}_T = g_1^c$. Moreover,

$$\begin{aligned}|\Pr[\mathcal{B}(g_1^{\boldsymbol{a}},\mathcal{M}_0)=1] - \Pr[\mathcal{B}(g_1^{\boldsymbol{a}},\mathcal{M}_T)=1]| &\le \sum_{k=1}^{T} |\Pr[\mathcal{B}(g_1^{\boldsymbol{a}},\mathcal{M}_{k-1})=1] - \Pr[\mathcal{B}(g_1^{\boldsymbol{a}},\mathcal{M}_k)=1]| \\ &\le T\cdot|\Pr[\mathcal{B}(g_1^{\boldsymbol{a}},\mathcal{M}_{k-1})=1] - \Pr[\mathcal{B}(g_1^{\boldsymbol{a}},\mathcal{M}_k)=1]|\end{aligned}$$

We complete the proof of Lemma 2 by showing the following claim:

Claim 1 *For every $1 \le k \le T$, if XDDH is ϵ'-hard in $\mathbb{G}_1$, then*

$$|\Pr[\mathcal{B}(g_1^{\boldsymbol{a}},\mathcal{M}_{k-1}=1] - \Pr[\mathcal{B}(g_1^{\boldsymbol{a}},\mathcal{M}_k)=1]| \le \epsilon'$$

Assume by contradiction that $|\Pr[\mathcal{B}(g_1^{\boldsymbol{a}},\mathcal{M}_{k-1}=1] - \Pr[\mathcal{B}(g_1^{\boldsymbol{a}},\mathcal{M}_k)=1]| \ge \epsilon'$. Then it is possible to build a simulator $\mathcal{B}'$ which breaks the XDDH assumption in $\mathbb{G}_1$ with advantage greater than ϵ'. $\mathcal{B}'$ gets an XDDH instance $(g_1, g_1^\alpha, g_1^\beta, g_1^\gamma)$ and proceeds as follows:

- It samples $c_1,\ldots,c_{k-1} \xleftarrow{\$} \mathbb{Z}_p$.
- It samples $a_1,\ldots,a_{\hat{i}-1},a_{\hat{i}+1},\ldots,a_t,b_1,\ldots,b_{\hat{j}-1},b_{\hat{j}+1},\ldots,b_t \xleftarrow{\$} \mathbb{Z}_p$, where $(\hat{i},\hat{j})$ correspond to k, i.e., $k = \hat{j} + (\hat{i}-1)t$.
- It implicitly sets $\tilde{\boldsymbol{a}} = (a_1,\ldots,a_{\hat{i}-1},\alpha,a_{\hat{i}+1},\ldots a_t)$ and $\tilde{\boldsymbol{b}} = (b_1,\ldots,b_{\hat{j}-1},\beta,b_{\hat{j}+1},\ldots b_t)$.
- $\mathcal{B}'$ builds a matrix $\mathcal{M} \in \mathbb{G}_1^{t\times t}$ where:
  - If $k' \le k-1$, $\mathcal{M}[i,j] = g_1^{c_{i,j}}$,
  - If $k' = k$, $\mathcal{M}[i,j] = g_1^\gamma$,
  - If $k' > k$, $\mathcal{M}[i,j] = g_1^{\tilde{a}_i\cdot\tilde{b}_j}$. Notice that such value can be efficiently computed by $\mathcal{B}'$ as it knows $g_1^{\tilde{a}_{\hat{i}}} = g_1^\alpha$, $g_1^{\tilde{b}_{\hat{j}}} = g_1^\beta$, $\tilde{a}_i, \forall i \ne \hat{i}$, $\tilde{b}_j, \forall j \ne \hat{j}$, and $k' > k$ implies $(i,j) \ne (\hat{i},\hat{j})$.
- $\mathcal{B}'$ runs $b' \leftarrow \mathcal{B}(g_1^{\tilde{\boldsymbol{a}}},\mathcal{M})$ and returns the same bit b'.

As one can check, if $\gamma = \alpha\beta$, then $\mathcal{M}'$ is distributed as $\mathcal{M}_{k-1}$. Otherwise, if γ is random and independent, $\mathcal{M}'$ is distributed as $\mathcal{M}_k$. Therefore,

$$\begin{aligned}&|\Pr[\mathcal{B}'(g_1,g_1^\alpha,g_1^\beta,g_1^{\alpha\beta})=1] - \Pr[\mathcal{B}'(g_1,g_1^\alpha,g_1^\beta,g_1^\gamma)=1]| = \\ &= |\Pr[\mathcal{B}(g_1^{\tilde{\boldsymbol{a}}},\mathcal{M}_{k-1})=1] - \Pr[\mathcal{B}(g_1^{\tilde{\boldsymbol{a}}},\tilde{\mathcal{M}}_k)=1]| \ge \epsilon'\end{aligned}$$

□

$(\mathsf{poly}, 0, 2)$-programmability of $\mathsf{H}_{\mathsf{sqrt}}$. Below we show that $\mathsf{H}_{\mathsf{sqrt}}$ is $(\mathsf{poly}, 0, 2, \gamma, \delta)$-programmable for $\gamma = 0$ and $\delta = 1$. While such $(\mathsf{poly}, 0)$-programmability might look uninteresting at first, this property turns out to be useful in various security proofs, as shown in our application to homomorphic signatures of Section 5.

Theorem 5. *The asymmetric hash function $\mathsf{H}_{\mathsf{sqrt}}$ described above is $(\mathsf{poly}, 0, d, \gamma, \delta)$-programmable with $d = 2$, $\gamma = 0$ and $\delta = 1$. Furthermore, in the case when either $\hat{h}_1 = \hat{g}_1$ or $\hat{h}_2 = \hat{g}_2$, $\mathsf{H}_{\mathsf{sqrt}}$ is $(\mathsf{poly}, 0, d, \gamma, \delta)$-programmable with $d = 1$, $\gamma = 0$ and $\delta = 1$.*

Proof. The trapdoor algorithms are defined as follows:

$\mathsf{H.TrapGen}(1^\lambda, \hat{g}_1, \hat{h}_1, \hat{g}_2, \hat{h}_2)$**:** first, sample $r_i, s_i \xleftarrow{\$} \mathbb{Z}_p$ for all $i \in [t]$ and then set $A_i = \hat{h}_1^{r_i}, B_i = \hat{h}_2^{s_i}$. Finally, set $\mathsf{td} = (\{r_i, s_i\}_{i=1}^t)$, $\mathsf{pek} = \{A_i, B_i\}_{i=1}^t$, and output $(\mathsf{td}, \mathsf{pek})$.

$\mathsf{H.TrapEval}(\mathsf{td}, X)$**:** let $X = (i, j)$, and then return the coefficients of the degree-2 polynomial

$$c_X(y_1, y_2) = (y_1 y_2) r_i s_j$$

Syntax and correctness are easily seen by inspection. The public key generated by $\mathsf{H.TrapGen}$ is distributed identically to the one generated by $\mathsf{H.Gen}$, from which $\gamma = 0$. Also, it is clear that for any $X \in \mathcal{X}$, the degree-0 term of the polynomial c_X computed by $\mathsf{H.TrapEval}$ is always 0. It is straightforward to see that in the case $y_1 = 1$ (or $y_2 = 1$) the function satisfies the programmability with $d = 1$. □

Weak $(\mathsf{poly}, 1, 2)$-programmability of $\mathsf{H}_{\mathsf{sqrt}}$. Here we prove that $\mathsf{H}_{\mathsf{sqrt}}$ is weakly $(\mathsf{poly}, 1, 2, \gamma, \delta)$-programmable for $\gamma = 0$ and $\delta = 1$.

Theorem 6. *The asymmetric hash function $\mathsf{H}_{\mathsf{sqrt}}$ described above is weakly $(\mathsf{poly}, 1, d, \gamma, \delta)$-programmable with $d = 2$, $\gamma = 0$ and $\delta = 1$.*

Proof. The trapdoor algorithms are defined as follows:

$\mathsf{H.TrapGen}(1^\lambda, \hat{g}_1, \hat{h}_1, \hat{g}_2, \hat{h}_2, Z)$**:** let $Z = (i^*, j^*) \in [t] \times [t]$. First, sample $r_i, s_i \xleftarrow{\$} \mathbb{Z}_p$ for all $i \in [t]$. Next, compute $A_{i^*} = \hat{g}_1 \hat{h}_1^{r_{i^*}}, B_{j^*} = \hat{g}_2 \hat{h}_2^{s_{j^*}}$, $A_i = \hat{h}_1^{r_i}, \forall i \neq i^*$ and $B_j = \hat{h}_2^{s_j}$, $\forall j \neq j^*$. Finally, set $\mathsf{td} = (\{r_i, s_i\}_{i=1}^t)$, $\mathsf{pek} = \{A_i, B_i\}_{i=1}^t$, and output $(\mathsf{td}, \mathsf{pek})$.

$\mathsf{H.TrapEval}(\mathsf{td}, X)$**:** given $X = (i, j)$, return the coefficients of the degree-2 polynomial

$$c_X(y_1, y_2) = \alpha_i(y_1) \cdot \beta_j(y_2)$$

where $\alpha_i(y_1)$ (resp. $\beta_j(y_2)$) is the discrete logarithm of A_i (resp. B_j) in base $\hat{g}_1$ (resp. $\hat{g}_2$), viewed as a degree-1 polynomial in the unknown y_1 (resp. y_2).

Syntax and correctness are easily seen by inspection. The public key generated by $\mathsf{H.TrapGen}$ is distributed identically to the one generated by $\mathsf{H.Gen}$, from which $\gamma = 0$. Also, it is clear from the construction that for $Z = (i^*, j^*)$ we have $c_Z(y_1, y_2) = (y_1 r_1 + 1)(y_2 s_j + 1)$, and thus $c_{Z,0} = 1$, whereas for every $X \neq Z$ the degree-0 term of the polynomial $c_Z(y_1, y_2)$ computed by $\mathsf{H.TrapEval}$ is always 0. And this holds with probability $\delta = 1$. □

Weak $(\mathsf{poly}, 1, 2)$-degree-2-programmability of $\mathsf{H}_{\mathsf{sqrt}}$. Finally, we prove that $\mathsf{H}_{\mathsf{sqrt}}$ is also weakly $(\mathsf{poly}, 1, 2, \gamma, \delta)$-degree-2-programmable for $\gamma = 0$ and $\delta = 1$.

Theorem 7. *The asymmetric hash function $\mathsf{H}_{\mathsf{sqrt}}$ described above is weakly $(\mathsf{poly}, 1, d, \gamma, \delta)$-degree-2 programmable with $d = 2$, $\gamma = 0$ and $\delta = 1$.*

Proof. The proof of this theorem can be seen as the "dual" version of the one of Theorem 6. Instead of setting the simulated keys so that Z is the only input for which $c_{Z,0} = 1$, here the keys are simulated in such a way that Z is the only input in which the term $y_1 y_2$ appears. More precisely, the trapdoor algorithms work as follows:

$\mathsf{H.TrapGen}(1^\lambda, \hat{g}_1, \hat{h}_1, \hat{g}_2, \hat{h}_2, Z)$**:** let $Z = (i^*, j^*) \in [t] \times [t]$. First, sample $r_i, s_i \xleftarrow{\$} \mathbb{Z}_p$ for all $i \in [t]$ and then set $A_{i^*} = \hat{h}_1 \hat{g}_1^{r_{i^*}}, B_{j^*} = \hat{h}_2 \hat{g}_2^{s_{j^*}}$, $A_i = \hat{g}_1^{r_i}, \forall i \neq i^*$ and $B_j = \hat{g}_2^{s_j}, \forall j \neq j^*$. Finally, set $\mathsf{td} = (\{r_i, s_i\}_{i=1}^t)$, $\mathsf{pek} = \{A_i, B_i\}_{i=1}^t$, and output $(\mathsf{td}, \mathsf{pek})$.

$\mathsf{H.TrapEval}(\mathsf{td}, X)$**:** let $X = (i, j)$, and then return the coefficients of the degree-2 polynomial

$$c_X(y_1, y_2) = \alpha_i(y_1) \cdot \beta_j(y_2)$$

where $\alpha_i(y_1)$ (resp. $\beta_j(y_2)$) is the discrete logarithm of A_i (resp. B_j) in base $\hat{g}_1$ (resp. $\hat{g}_2$), viewed as a degree-1 polynomial in the unknown y_1 (resp. y_2).

Syntax and correctness are easily seen by inspection. The public key generated by $\mathsf{H.TrapGen}$ is distributed identically to the one generated by $\mathsf{H.Gen}$, from which $\gamma = 0$. By construction, we have that for $Z = (i^*, j^*)$, $c_Z(y_1, y_2) = (r_1 + y_1)(s_j + y_2)$, and thus $c_{Z,2} = 1$, whereas for every $X \neq Z$ the polynomial $c_X(y_1, y_2)$ has degree ≤ 1, and thus $c_{X,2} = 0$. This property holds with probability $\delta = 1$. □

4 Linearly-Homomorphic Signatures with Short Public Keys

In this section, we show a new linearly-homomorphic signature scheme that uses APHFs in a generic way. By instantiating the APHFs with our construction $\mathsf{H}_{\mathsf{sqrt}}$ given in Section 3, we obtain the *first* linearly-homomorphic signature scheme that is secure in the standard model, and whose public key has a size that is *sub-linear* in both the dataset size and the dimension of the signed vectors. Precisely, if the signature scheme supports datasets of maximal size N and can sign vectors of dimension T, then the public key of our scheme is of size $O(\sqrt{N} + \sqrt{T})$. All previously existing constructions in the standard model achieved only public keys of length $O(N + T)$. Furthermore, our scheme is adaptive secure and achieves the interesting property of *efficient verification* that allows to use the scheme for verifiable delegation of computation in the preprocessing model [16].

Before describing our scheme, in the next section we recall the definition of homomorphic signatures.

4.1 Homomorphic Signatures for Multi-Labeled Programs

In this section we recall the definition of homomorphic signatures as presented in [16]. This definition extends the one by Freeman in [18] in order to work with the general notion of multi-labeled programs [21,4].

Multi-Labeled Programs. A *labeled program* $\mathcal{P}$ is a tuple $(f, \tau_1, ..., \tau_n)$ such that $f : \mathcal{M}^n \to \mathcal{M}$ is a function of n variables (e.g., a circuit) and $\tau_i \in \{0,1\}^*$ is a label of the i-th input of f. Labeled

programs can be composed as follows: 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 it is denoted as $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$. The labeled inputs of $\mathcal{P}^*$ are all the distinct labeled inputs of $\mathcal{P}_1, \cdots \mathcal{P}_t$ (all the inputs with the same label are grouped together and considered as a unique input of $\mathcal{P}^*$).
Let $f_{id} : \mathcal{M} \to \mathcal{M}$ be the identity function and $\tau \in \{0,1\}^*$ be any label. We refer to $\mathcal{I}_\tau = (f_{id}, \tau)$ as the identity program with label τ. Note that a program $\mathcal{P} = (f, \tau_1, \cdots, \tau_n)$ can be expressed as the composition of n identity programs $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \cdots, \mathcal{I}_{\tau_n})$.

A *multi-labeled program* $\mathcal{P}_\Delta$ is a pair $(\mathcal{P}, \Delta)$ in which $\mathcal{P} = (f, \tau_1, \cdots, \tau_n)$ is a labeled program while $\Delta \in \{0,1\}^*$ is a *data set identifier*. Multi-labeled programs can be composed within the same data set in the most natural way: given $(\mathcal{P}_1, \Delta), \ldots, (\mathcal{P}_t, \Delta)$ which has 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 $g(\mathcal{P}_1, \cdots, \mathcal{P}_t)$, and Δ is the common data set identifier for all the $\mathcal{P}_i$. As for labeled programs, one can define the notion of a multi-labeled identity program as $\mathcal{I}_{(\Delta,\tau)} = ((f_{id}, \tau), \Delta)$.

Definition 8 (Homomorphic Signatures). *A homomorphic signature scheme* HSig *consists of a tuple of PPT algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{Eval})$ *satisfying the following four properties:* authentication correctness*,* evaluation correctness*,* succinctness *and* security*. The four algorithms work as follows:*

$\mathsf{KeyGen}(1^\lambda, \mathcal{L})$ *the key generation algorithm takes as input a security parameter λ, the description of the label space $\mathcal{L}$ (which fixes the 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.*

$\mathsf{Sign}(\mathsf{sk}, \Delta, \tau, m)$ *the signing algorithm takes as input 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.*

$\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_\Delta, m, \sigma)$ *the verification algorithm takes as input 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).*

$\mathsf{Eval}(\mathsf{vk}, f, \boldsymbol{\sigma})$ *the evaluation algorithm takes as input a public* 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.*

Below we describe the four properties mentioned above:

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

Evaluation Correctness. Intuitively, this property says that running the evaluation algorithm on signatures $(\sigma_1, \cdots, \sigma_n)$ such that each σ_i verifies for m_i as the output of a multi-labeled program (P_i, Δ), produces a signature σ which verifies for $f(m_1, \cdots, m_t)$ as the output of the composed program $(f(\mathcal{P}_1, \cdots, \mathcal{P}_n), \Delta)$. More formally, fix a key pair $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, \mathcal{L})$, 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 $m^* = g(m_1, \ldots, m_t)$, $\mathcal{P}^* = g(\mathcal{P}_1, \cdots, \mathcal{P}_t)$, 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.

SUCCINTNESS. A homomorphic signature scheme is said to be succint if, for a fixed security parameter λ, the size of signatures depends at most logarithmically on the data set size N.

SECURITY. To define the security notion of homomorphic signatures we define the following experiment $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HomSign}}(\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:

Key Generation $\mathcal{C}$ runs $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, \mathcal{L})$ and gives vk to $\mathcal{A}$.

Signing Queries $\mathcal{A}$ can adaptively submit queries of the form (Δ, τ, m), where Δ is a data set identifier, $\tau \in \mathcal{L}$, and $m \in \mathcal{M}$. The challenger $\mathcal{C}$ proceeds as follows: if (Δ, τ, m) is the first query with the data set identifier Δ, the challenger initializes an empty list $T_\Delta = \emptyset$ for Δ. If T_Δ does not already contain a tuple $(\tau, \cdot)$ (which means that $\mathcal{A}$ never asked for a query $(\Delta, \tau, \cdot)$), the challenger $\mathcal{C}$ 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 $(\tau, m) \in T_\Delta$ (which means that the adversary had already queried the tuple (Δ, τ, m)), then $\mathcal{C}$ 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 At the end $\mathcal{A}$ outputs a tuple $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$.

The experiment $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HomSign}}(\lambda)$ outputs 1 if the tuple returned by $\mathcal{A}$ is a forgery, and 0 otherwise.

To define what is a forgery in such a game we recall the notion of well defined program with respect to a list T_Δ [16].

Definition 9. *A labeled program $\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*)$ is well defined with respect to T_{Δ^*} if one of the two following cases holds:*

- $\exists\, m_1, \ldots, m_n$ *s.t.* $(\tau_i^*, m_i) \in T_{\Delta^*}\ \forall i = 1, \ldots, n$.
- $\exists\, i \in \{1, \cdots, n\}$ *s.t.* $(\tau_i, \cdot) \notin T_{\Delta^*}$ *and* $f^*(\{m_j\}_{(\tau_j, m_j) \in T_{\Delta^*}} \cup \{\tilde{m}_{(\tau_j, \cdot) \notin T_{\Delta^*}}\})$ *does not change for all possible choices of* $\tilde{m}_j \in \mathcal{M}$.

Intuitively, the first case says that the challenger has generated signatures for the entire input space of f for the data set Δ^*, while the second one means that the inputs that were not signed during the experiment do not contribute to the result of f.

Using this notion, it is then possible to define the three different types of forgeries that can occur in the experiment $\mathsf{HomUF\text{-}CMA}$:

Type 1: $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}^*_{\Delta^*}, m^*, \sigma^*) = 1$ and the list T_{Δ^*} was not initialized during the game (i.e., no message was ever signed w.r.t. data set identifier Δ^*).

Type 2: $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}^*_{\Delta^*}, m^*, \sigma^*) = 1$, $\mathcal{P}^*$ is well defined with respect to T_{Δ^*} and $m^* \neq f^*(\{m_j\}_{(\tau_j, m_j) \in T_{\Delta^*}})$ (i.e., m^* is not the correct output of $\mathcal{P}^*$ when executed over previously signed messages).

Type 3: $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}^*_{\Delta^*}, m^*, \sigma^*) = 1$ and $\mathcal{P}^*$ is *not* well defined with respect to T_{Δ^*}.

Then we say that HSig is a secure homomorphic signature if for any PPT adversary $\mathcal{A}$, we have that $\Pr[\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HomSign}}(\lambda) = 1] \leq \epsilon(\lambda)$ where $\epsilon(\lambda)$ is a negligible function.

We recall that, as proved by Freeman in [18], in a linearly-homomorphic signatures scheme any adversary who outputs a Type 3 forgery can be converted into one that outputs a Type 2 one.

Proposition 1 ([18]). *Let* HSig *be a linearly homomorphic signature scheme with message space $\mathcal{M} \subset \mathcal{R}^n$ for some ring $\mathcal{R}$. If* HSig *is secure against Type 2 forgeries, then* HSig *is secure against Type 3 forgeries.*

Homomorphic Signatures with Efficient Verification. We recall the notion of homomorphic signatures with efficient verification introduced in [16]. The property states that the verification algorithm can be split in two phases: an *offline* phase where, given the verification key vk and a labeled program $\mathcal{P}$, one precomputes a concise key $\mathsf{vk}_{\mathcal{P}}$; an *online* phase in which $\mathsf{vk}_{\mathcal{P}}$ can be used to verify signatures w.r.t. $\mathcal{P}$ and *any* dataset Δ. To achieve (amortized) efficiency, the idea is that $\mathsf{vk}_{\mathcal{P}}$ can be reused an unbounded number of times, and the online verification is cheaper than running $\mathcal{P}$. Below is the formal definition.

Definition 10. *Let* $\mathsf{HSig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{Eval})$ *be a homomorphic signature scheme for multi-labeled programs.* HSig satisfies efficient verification *if there exist two additional algorithms* ($\mathsf{VerPrep}$, 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* $t' = o(t(n))$*.*

4.2 Our Construction

Let $\Sigma' = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Ver}')$ be a regular signature scheme, and $F : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$ be a pseudorandom function with key space $\mathcal{K}$. Our linearly-homomorphic signature scheme signs T-dimensional vectors of messages in $\mathbb{Z}_p$, and supports datasets of size N, with both $N = \mathsf{poly}(\lambda)$ and $T = \mathsf{poly}(\lambda)$. Let $\mathsf{H} = (\mathsf{H.Gen}, \mathsf{H.PriEval}, \mathsf{H.PubEval})$ and $\mathsf{H}' = (\mathsf{H.Gen}', \mathsf{H.PriEval}', \mathsf{H.PubEval}')$ be two asymmetric programmable hash functions such that $\mathsf{H} : [N] \to \mathbb{G}_1$ and $\mathsf{H}' : [T] \to \mathbb{G}_1$.

We construct a homomorphic signature $\mathsf{HSig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{Eval})$ as follows:

$\mathsf{KeyGen}(1^\lambda, \mathcal{L}, T)$**.** Let λ be the security parameter, $\mathcal{L}$ be a set of admissible labels where $\mathcal{L} = \{1, \ldots, N\}$, and T be an integer representing the dimension of the vectors to be signed. The key generation algorithm works as follows.

- Generate a key pair $(\mathsf{vk}', \mathsf{sk}') \xleftarrow{\$} \mathsf{KeyGen}'(1^\lambda)$ for the regular scheme.
- Run $\mathsf{bgp} \xleftarrow{\$} \mathcal{G}(1^\lambda)$ to generate the bilinear groups parameters $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order $p \approx 2^\lambda$, $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ are generators and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable, non-degenerate bilinear map.
- Choose a random seed $K \xleftarrow{\$} \mathcal{K}$ for the PRF $F_K : \{0,1\}^* \to \mathbb{Z}_p$.
- Run $(\mathsf{sek}, \mathsf{pek}) \xleftarrow{\$} \mathsf{H.Gen}(1^\lambda, \mathsf{bgp})$ and $(\mathsf{sek}', \mathsf{pek}') \xleftarrow{\$} \mathsf{H.Gen}'(1^\lambda, \mathsf{bgp})$ to generate the keys of the asymmetric hash functions.
- Return $\mathsf{vk} = (\mathsf{vk}', \mathsf{bgp}, \mathsf{pek}, \mathsf{pek}')$ and $\mathsf{sk} = (\mathsf{sk}', K, \mathsf{sek}, \mathsf{sek}')$.

$\mathsf{Sign}(\mathsf{sk}, \Delta, \tau, \boldsymbol{m})$**.** The signing algorithm takes as input the secret key sk, a data set identifier $\Delta \in \{0,1\}^*$, a label $\tau \in [N]$ and a message vector $\boldsymbol{m} \in \mathbb{Z}_p^T$, and proceeds as follows:

1. Derive the integer $z \leftarrow F_K(\Delta)$ using the PRF, and compute $Z = g_2^z$.
2. Compute $\sigma_\Delta \leftarrow \mathsf{Sign}'(\mathsf{sk}', \Delta | Z)$ to bind Z to the dataset identifier Δ.
3. Choose a random $R \xleftarrow{\$} \mathbb{G}_1$ and compute

$$S = \left(\mathsf{H.PriEval}(\mathsf{sek}, \tau) \cdot R \cdot \prod_{j=1}^{T} \mathsf{H.PriEval}'(\mathsf{sek}', j)^{m_j} \right)^{1/z}$$

4. Return a signature $\sigma = (\sigma_\Delta, Z, R, S)$.

Essentially, the algorithm consists of two main steps. First, it uses the PRF F_K to derive a common parameter z which is related to the data set Δ, and it signs the public part, $Z = g_2^z$, of this parameter using the signature scheme Σ'. Second, it uses z to create the homomorphic component R, S of the signature, such that S is now related to all $(\Delta, \tau, \boldsymbol{m})$.

$\mathsf{Eval}(\mathsf{vk}, f, \boldsymbol{\sigma})$**.** The public evaluation algorithm takes as input the public key vk, a linear function $f : \mathbb{Z}_p^\ell \to \mathbb{Z}_p$ described by its vector of coefficients $\boldsymbol{f} = (f_1, \ldots, f_\ell)$, and a vector $\boldsymbol{\sigma}$ of ℓ signatures $\sigma_1, \ldots, \sigma_\ell$ where $\sigma_i = (\sigma_{\Delta,i}, Z_i, R_i, S_i)$ for $i = 1, \ldots, \ell$. Eval returns a signature $\sigma = (\sigma_\Delta, , Z, R, S)$ that is obtained by setting $Z = Z_1$, $\sigma_\Delta = \sigma_{\Delta,1}$, and by computing

$$R = \prod_{i=1}^{\ell} R_i^{f_i}, \quad S = \prod_{i=1}^{\ell} S_i^{f_i}$$

$\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_\Delta, \boldsymbol{m}, \sigma)$**.** Let $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_\ell), \Delta)$ be a multi-labeled program such that $f : \mathbb{Z}_p^\ell \to \mathbb{Z}_p$ is a linear function described by coefficients $\boldsymbol{f} = (f_1, \ldots, f_\ell)$. Let $\boldsymbol{m} \in \mathbb{Z}_p^T$ be a message-vector and $\sigma = (\sigma_\Delta, Z, R, S)$ be a signature.

First, run $\mathsf{Ver}'(\mathsf{vk}', \Delta | Z, \sigma_\Delta)$ to check that σ_Δ is a valid signature for Z and the dataset identifier Δ taken as input by the verification algorithm. If σ_Δ is not valid, stop and return 0 (reject).

Otherwise, output 1 if and only if the following equation is satisfied

$$e(S, Z) = \left(\prod_{i=1}^{\ell} \mathsf{H.PubEval}(\mathsf{pek}, \tau_i)^{f_i} \right) \cdot e(R, g_2) \cdot \left(\prod_{j=1}^{T} \mathsf{H.PubEval}'(\mathsf{pek}', j)^{m_j} \right) \tag{1}$$

Finally, we describe the algorithms for efficient verification:

$\mathsf{VerPrep}(\mathsf{vk}, \mathcal{P})$**.** Let $\mathcal{P} = (f, \tau_1, \ldots, \tau_\ell)$ be a labeled program for a linear function $f : \mathbb{Z}_p^\ell \to \mathbb{Z}_p$. The algorithm computes $H = \prod_{i=1}^{\ell} \mathsf{H.PubEval}(\mathsf{pek}, \tau_i)^{f_i}$, and returns the concise verification key $\mathsf{vk}_\mathcal{P} = (\mathsf{vk}', \mathsf{bgp}, H, \mathsf{pek}')$.

$\mathsf{EffVer}(\mathsf{vk}_\mathcal{P}, \Delta, \boldsymbol{m}, \sigma)$**.** The online verification is the same as Ver except that in the verification equation the value H has been already computed in the off-line phase (and is included in $\mathsf{vk}_\mathcal{P}$).

Clearly, running the combination of $\mathsf{VerPrep}$ and EffVer gives the same result as running Ver, and EffVer's running time is independent of f's complexity ℓ.

We formally show the correctness of our homomorphic signature scheme in Section 5.3. The following theorem states the security of the scheme. Its proof appears in Section 5.4.

Theorem 8. *Assume that Σ' is an unforgeable signature scheme, F is a pseudorandom function, and $\mathcal{G}$ is a bilinear group generator such that:* H *has* $(1, \gamma, \epsilon)$*-programmable pseudorandomness;* H' *is weakly* $(\mathsf{poly}, 1, 2, \gamma', \delta')$*-degree-2-programmable, weakly* $(\mathsf{poly}, 1, 2, \gamma', \delta')$*-programmable and* $(\mathsf{poly}, 0, 1, \gamma', \delta')$*-programmable; the* 2*-DHI and the FDHI assumptions hold. Then* HSig *is a secure linearly-homomorphic signature scheme.*

We note that our scheme HSig can be instantiated by instantiating both H and H' with two different instances of our programmable hash $\mathsf{H}_{\mathsf{sqrt}}$ described in Section 3.3. As one can check in Section 3.3, $\mathsf{H}_{\mathsf{sqrt}}$ allows for the multiple programmability modes required in our Theorem 10. Let us stress that requiring the same function to have multiple programmability modes is not contradictory, as such modes do not have to hold simultaneously. It simply means that for the same function there exist different pairs of trapdoor algorithms each satisfying programmability with different parameters.[7]

4.3 Proof of Correctness

Theorem 9. *If Σ' is a correct signature scheme, and* H, H' *are asymmetric hash functions for bilinear groups, then the scheme* HSig *satisfies the authentication correctness property.*

Proof. Let $(\mathsf{sk}, \mathsf{vk})$ be a pair of honestly generated keys and let $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \Delta, \tau, \boldsymbol{m})$ be a honestly generated signature, with $\sigma = (\sigma_\Delta, Z, R_\tau, S_\tau)$. In order to prove that the verification algorithm $\mathsf{Ver}(\mathsf{vk}, \mathcal{I}_{(\Delta,\tau)}, \boldsymbol{m}, \sigma)$ outputs 1 with all but negligible probability, the first observation to do is that by the correctness of Σ' the signature σ_Δ verifies correctly for Z and Δ. Then, by construction of HSig, we can see that

$$S_\tau = \left(\mathsf{H.PriEval}(\mathsf{sek}, \tau) \cdot R_\tau \cdot \prod_{j=1}^{T} \mathsf{H.PriEval}'(\mathsf{sek}', j)^{m_j} \right)^{1/z}$$

Hence, we have that

$$\begin{aligned} e(S_\tau, Z) &= e\left(\left(\mathsf{H.PriEval}(\mathsf{sek}, \tau) \cdot R_\tau \cdot \prod_{j=1}^{T} \mathsf{H.PriEval}'(\mathsf{sek}', j)^{m_j} \right)^{1/z}, Z \right) \\ &= e\left(\mathsf{H.PriEval}(\mathsf{sek}, \tau) \cdot R_\tau \cdot \prod_{j=1}^{T} \mathsf{H.PriEval}'(\mathsf{sek}', j)^{m_j}, g_2 \right) \\ &= e\left(\mathsf{H.PriEval}(\mathsf{sek}, \tau), g_2\right) \cdot e(R_\tau, g_2) \cdot e\left(\prod_{j=1}^{T} \mathsf{H.PriEval}'(\mathsf{sek}', j)^{m_j}, g_2 \right) \\ &= \mathsf{H.PubEval}(\mathsf{pek}, \tau) \cdot e(R_\tau, g_2) \cdot \prod_{j=1}^{T} \mathsf{H.PubEval}'(\mathsf{pek}', j)^{m_j} \end{aligned}$$

where the last equation holds by definition of $\mathsf{H.PubEval}$ and $\mathsf{H.PubEval}'$.

[7] We also stress that, by definition, the outputs of these trapdoor algorithms are statistically indistinguishable.

Theorem 10. *If Σ' is a correct signature scheme, and H, H' are asymmetric hash functions for bilinear groups, then the scheme HSig satisfies the evaluation correctness property.*

Proof. Let $(\mathsf{sk}, \mathsf{vk})$ be a pair of honestly generated keys, and let $\{\boldsymbol{m}^{(i)}, \mathcal{P}_{i,\Delta}, \sigma_i = (\sigma_\Delta, Z, R_i, S_i)\}_{i=1}^{\ell}$ be messages, labeled programs and signatures such that $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_{i,\Delta}, \boldsymbol{m}^{(i)}, \sigma_i) = 1$, for all $i = 1$ to ℓ. Let $\sigma \leftarrow \mathsf{Eval}(\mathsf{vk}, f, \boldsymbol{\sigma} = (\sigma_1, ..., \sigma_\ell))$ be a a signature obtained by running Eval on signatures $(\sigma_1, ..., \sigma_\ell)$, where $\sigma = (\sigma_\Delta, Z, R, S)$. By construction of Eval, we have $R = \prod_{i=1}^{\ell} R_i^{f_i}$ and $S = \prod_{i=1}^{\ell} S_i^{f_i}$. So, if we let $\boldsymbol{m} = f(\boldsymbol{m}^{(1)}, ..., \boldsymbol{m}^{(\ell)}) = \sum_{i=1}^{\ell} f_i \cdot \boldsymbol{m}^{(i)}$, for evaluation correctness we want to prove that the verification algorithm $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_\Delta, \boldsymbol{m}, \sigma)$ outputs 1.

The fact that σ_Δ verifies correctly for Z and Δ is immediate by correctness of Σ' and by construction of Eval (which simply copies one of these honestly-generated signatures).

Since each σ_i verifies correctly, for every $i = 1, \ldots, \ell$ we have

$$e(S_i, Z) = \mathsf{H.PubEval}(\mathsf{pek}, \tau_i) \cdot e(R_i, g_2) \cdot \prod_{j=1}^{T} \mathsf{H.PubEval}'(\mathsf{pek}', j)^{m_j^{(i)}}$$

Then, by the previous equations and the fact that H, H' are asymmetric hash functions for bilinear groups, we obtain the desired equation:

$$\begin{aligned}
e(S, Z) &= e\left(\prod_{i=1}^{\ell} S_i^{f_i}, Z\right) \\
&= e\left(\prod_{i=1}^{\ell} \left(\mathsf{H.PriEval}(\mathsf{sek}, \tau)^{f_i} \cdot R_i^{f_i} \cdot \prod_{j=1}^{T} \mathsf{H.PriEval}'(\mathsf{sek}', j)^{f_i m_j^{(i)}}\right)^{1/z}, Z\right) \\
&= \left(\prod_{i=1}^{\ell} \mathsf{H.PubEval}(\mathsf{pek}, \tau)^{f_i}\right) \cdot e(R, g_2) \cdot \left(\prod_{j=1}^{T} \mathsf{H.PubEval}'(\mathsf{pek}', j)^{\sum_{i=1}^{\ell} f_i m_j^{(i)}}\right) \\
&= \left(\prod_{i=1}^{\ell} \mathsf{H.PubEval}(\mathsf{pek}, \tau)^{f_i}\right) \cdot e(R, g_2) \cdot \left(\prod_{j=1}^{T} \mathsf{H.PubEval}'(\mathsf{pek}', j)^{m_j}\right)
\end{aligned}$$

4.4 Proof of Security

To prove Theorem 10, we show that for every PPT adversary $\mathcal{A}$ running in the security experiment $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HSig}}$, the probability that the experiment outputs 1 is negligible. We do the proof by describing a series of hybrid games. We write $G_i(\mathcal{A})$ to denote the event that a run of Game i with adversary $\mathcal{A}$ returns 1. Some of the games use some flag values bad_i that are initially set to false. If at the end of a game any of these values is set to true, the game simply outputs 0. We call Bad_i the event that bad_i is set to true during the run of an experiment. Essentially, whenever an event Bad_i occurs in Game i, the game may deviate its outcome.

Finally, we note that in the following proof we directly use the result of Proposition 1 so that we only have to deal with Type-1 and Type-2 forgeries, since Type-3 ones can be converted in Type-2.

Game 0 This game is the security experiment $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HSig}}$ (where $\mathcal{A}$ only outputs Type-1 or Type-2 forgeries).

Game 1 This game is defined as Game 0 apart from the fact that whenever $\mathcal{A}$ returns a forgery $\sigma^* = (\sigma^*_{\Delta}, Z^*, R^*, S^*)$ such that Z^* was not generated by the challenger in the signing query phase, then Game 1 sets $\mathsf{bad}_1 \leftarrow \mathsf{true}$. As we show in Lemma 7, any noticeable difference between Game 0 and Game 1 can be reduced to producing a forgery for the regular signature scheme Σ'. Furthermore, it is worth noting that after this change, the game never outputs 1 if the adversary returns a Type-1 forgery.

Game 2 This game is defined as Game 1 except that the pseudorandom function F is replaced by a random function $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$. It is easy to see that Game 1 is computationally indistinguishable from Game 2 under the assumption that F is pseudorandom.

Game 3 is defined as Game 2 except for the following change. Let $(\mathcal{P}^*_{\Delta^*}, \sigma^*, \boldsymbol{m}^*)$ be the forgery returned by the adversary where $\mathcal{P}^*_{\Delta^*} = (\boldsymbol{f}^*, \mathcal{L}^*)$, $\sigma^* = (\sigma^*_{\Delta}, Z^*, R^*, S^*)$ and $\Delta^* = \Delta_\mu$ for some $\mu \in [Q]$ where Q is the number of distinct datasets asked by $\mathcal{A}$ during the game (note that such μ must exist at this point since the adversary can win only with a Type-2 forgery). The challenger computes $\hat{S} = \prod_{\tau \in \mathcal{L}^*} (S_\tau)^{f^*_\tau}$, $\hat{R} = \prod_{\tau \in \mathcal{L}^*} (R_\tau)^{f^*_\tau}$ $\hat{\boldsymbol{m}} = \sum_{\tau \in \mathcal{L}^*} f^*_\tau \cdot \boldsymbol{m}_\tau$ where $\{R_\tau, S_\tau\}_\tau$ are the signature components generated by the challenger in all the signing queries $(\Delta_\mu, \tau, \boldsymbol{m}_\tau)$. If the forgery verifies correctly, i.e., $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}^*_{\Delta^*}, \boldsymbol{m}^*, \sigma^*) = 1$, *and* $\boldsymbol{m}^* \neq \hat{\boldsymbol{m}}$ *and* $S^* = \hat{S}$, then the challenger sets $\mathsf{bad}_3 \leftarrow \mathsf{true}$.

It is easy to see that $\Pr[G_2(\mathcal{A})] - \Pr[G_3(\mathcal{A})] \leq \Pr[\mathsf{Bad}_3]$. In Lemma 9 we show that any adversary for which Bad_3 occurs can be reduced to a solver for the 1-DHI problem.

Game 4 This game proceeds as Game 3 except for the following change: at the beginning, the challenger chooses a random index $\mu \xleftarrow{\$} [Q]$, where $Q = \mathsf{poly}(\lambda)$ is the number of signing queries made by $\mathcal{A}$ during the game. Let $\Delta_1, \ldots, \Delta_Q$ be all the datasets queried by $\mathcal{A}$. Then if the dataset Δ^* used by $\mathcal{A}$ in the forgery is *not* Δ_μ, the challenger sets $\mathsf{bad}_4 \leftarrow \mathsf{true}$. As one can check, we have that $\Pr[G_3(\mathcal{A})] = Q \cdot \Pr[G_4(\mathcal{A})]$.

Game 5 proceeds as Game 4 except that at the end the challenger runs the following additional check: if $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}^*_{\Delta^*}, \boldsymbol{m}^*, \sigma^*) = 1$ *and* $\boldsymbol{m}^* \neq \hat{\boldsymbol{m}}$ *and* $S^* \neq \hat{S}$ *and* $R^* = \hat{R}$, then the challenger sets $\mathsf{bad}_5 \leftarrow \mathsf{true}$. It is easy to see that $\Pr[G_4(\mathcal{A})] - \Pr[G_5(\mathcal{A})] \leq \Pr[\mathsf{Bad}_5]$. In Lemma 11 we show that any adversary for which Bad_5 occurs can be reduced to a solver for the 2-DHI problem.

Game 6 proceeds as Game 5 with the following modification. At the very beginning, the challenger chooses the value $z_\mu \xleftarrow{\$} \mathbb{Z}_p$ that will be used to generate the signatures for μ-th dataset Δ_μ. It sets $Z_\mu = g_2^{z_\mu}$. Second, instead of generating the key pek of the hash function H using $\mathsf{H.Gen}$, the challenger runs $(\mathsf{td}, \mathsf{pek}) \xleftarrow{\$} \mathsf{H.TrapGen}(1^\lambda, \mathsf{bgp}, g_1, g_1, g_2, Z_\mu)$ where $\mathsf{H.TrapGen}$ is the algorithm for which H has $(1, \gamma, \epsilon)$-programmable pseudo-randomness. Then the challenger uses td when it needs to compute $\mathsf{H}(\cdot)$ during the experiment.

If H hash $(1, \gamma, \epsilon)$-programmable pseudorandomness we immediately obtain that Game 5 and Game 6 are within statistical distance γ, i.e., $|\Pr[G_5(\mathcal{A})] - \Pr[G_6(\mathcal{A})]| \leq \gamma$.

Game 7 This game is the same as Game 6, except that in the signing queries $(\Delta, \tau, \boldsymbol{m})$ such that Δ is the μ-th distinct dataset queried by $\mathcal{A}$, the challenger first computes $\boldsymbol{c}_\tau \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, \tau)$ and then generates the signature component R_τ by setting $R_\tau = g_1^{-c_{\tau,0}}$, instead of choosing $R_\tau \xleftarrow{\$} \mathbb{G}_1$ randomly as done up to Game 6.

As we show in Lemma 12, Game 6 is computationally indistinguishable from Game 7 under the assumption that H has programmable pseudo-randomness. Moreover, note that due to the previous modifications, Game 7 can output 1 only if the adversary outputs a forgery $(\mathcal{P}^*_{\Delta^*}, \sigma^*, \boldsymbol{m}^*)$ such that $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}^*_{\Delta^*}, \boldsymbol{m}^*, \sigma^*) = 1$ *and* $\boldsymbol{m}^* \neq \hat{\boldsymbol{m}}$ *and* $S^* \neq \hat{S}$ *and* $R^* \neq \hat{R}$. We conclude the

proof by showing in Lemma 13 that an adversary that wins in Game 7 can be used to solve the FDHI problem (Definition 4).

We proceed with the proof by formally bounding the difference between each consecutive pair of games, and eventually the probability that an adversary wins in the last game. The proof of Theorem 10 is finally obtained by putting together all the bounds.

Before proceeding with the actual proof we recall that the function H' is defined over a domain of polynomial size. By Remark 1 (Section 3) this means that the programmability properties stated in the theorem hold with probability $\delta' = 1$. This fact is always implicitly considered when proving the lemmas below.

Lemma 3. *For every PPT $\mathcal{A}$ there exists a PPT forger $\mathcal{F}$ such that* $\Pr[G_0(\mathcal{A})] - \Pr[G_1(\mathcal{A})] \leq \mathbf{Adv}^{\mathsf{UF\text{-}CMA}}_{\sigma',\mathcal{F}}(\lambda)$.

Proof. The two games differ only if Bad_1 occurs in Game 1, i.e., $|\Pr[G_0(\mathcal{A})] - \Pr[G_1(\mathcal{A})]| \leq \Pr[\mathsf{Bad}_1]$. However, by the construction of HSig, if Bad_1 occurs, it means that the forgery returned by $\mathcal{A}$ includes a valid signature σ_{Δ^*} on $(\Delta^*|Z^*)$ although no signature on $(\Delta^*|\cdot)$ was ever returned by the challenger during the experiment. It is straightforward to show that, for any such a PPT $\mathcal{A}$, there exists a PPT forger algorithm $\mathcal{F}$ that breaks the unforgeability of the regular signature scheme Σ', i.e., $\Pr[\mathsf{Bad}_1] \leq \mathbf{Adv}^{\mathsf{UF\text{-}CMA}}_{\sigma',\mathcal{F}}(\lambda)$.

Lemma 4. *For every PPT $\mathcal{A}$ there exists a PPT distinguisher $\mathcal{D}$ such that* $|\Pr[G_1(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \leq \mathbf{Adv}^{PRF}_{F,\mathcal{D}}(\lambda)$.

Proof. Game 1 and Game 2 differ just for the fact that the PRF F is replaced by a random function $\mathcal{R}$. It is easy to do a reduction to the security of the PRF to show that for any adversary $\mathcal{A}$ such that $|\Pr[G_1(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \geq \epsilon$ is non-negligible it is possible to construct a PPT distinguisher $\mathcal{D}$ that archives advantage ϵ against the pseudo-randomness of F.

Lemma 5. *If H is simply $(1,\gamma)$-programmable, and H' is weakly $(\mathsf{poly}, 1, 2, \gamma', \delta')$-degree-2 programmable, then for every PPT $\mathcal{A}$ running in Game 3 there exists a PPT simulator $\mathcal{B}$ such that* $\Pr[\mathsf{Bad}_3] \leq T \cdot \mathbf{Adv}^{1\text{-}DHI}_{\mathcal{B}}(\lambda) + \gamma + \gamma'$.

Proof. Assume there exists a PPT adversary $\mathcal{A}$ such that $\Pr[\mathsf{Bad}_3] \geq \epsilon$. Then we show how to build a PPT simulator $\mathcal{B}$ that breaks the 1-DHI assumption with advantage greater than $\epsilon/T - \gamma - \gamma'$.

$\mathcal{B}$ takes as input a tuple (g_1, g_2, g_1^z, g_2^z), and its goal is to compute $g_1^{z^2}$. Precisely, here we use the fact that this problem is equivalent to the 1-DHI problem in which the adversary has to compute $g_1^{1/z}$. So, $\mathcal{B}$ proceeds as follows.

Setup: $\mathcal{B}$ starts by sampling a random $y \xleftarrow{\$} \mathbb{Z}_p$ and runs $(\mathsf{td}, \mathsf{pek}) \xleftarrow{\$} \mathsf{H.TrapGen}(1^\lambda, \mathsf{bgp}, g_1, g_1, g_2, g_2^y)$. Note that since $\mathcal{B}$ had set $h_1 = g_1$, the polynomials c_X generated by $\mathsf{H.TrapEval}(\mathsf{td}, X)$ will be univariate polynomials $c_X(y)$. Next, it chooses a random index $\nu \xleftarrow{\$} [T]$, which represents a guess on the index where the message vector $\boldsymbol{m}^*$ returned by the adversary in the forgery will differ from the "correct" result $\hat{\boldsymbol{m}}$. Then $\mathcal{B}$ sets $h_1 = g_1^z, h_2 = g_2^z$ and runs the trapdoor generation (for weakly degree-2 programmability) of the asymmetric hash function H' – $(\mathsf{td}', \mathsf{pek}') \xleftarrow{\$} \mathsf{H}'.\mathsf{TrapGen}(1^\lambda, \mathsf{bgp}, g_1, h_1, g_2, h_2, \nu)$ – by providing ν as the input on which the coefficient $c_{\nu,2} \neq 0$. Indeed, notice that by giving $h_1 = g_1^z, h_2 = g_2^z$ to $\mathsf{H}'.\mathsf{TrapGen}$, the polynomials generated $\mathsf{H}'.\mathsf{TrapEval}(\mathsf{td}', X)$ will be univariate polynomials $c_X(z)$.

Finally, the simulator generates the keys $(\mathsf{sk}', \mathsf{vk}')$ of the scheme Σ', sets $\mathsf{vk} = (\mathsf{vk}', \mathsf{pek}, \mathsf{pek}')$, stores $\mathsf{sk}', \mathsf{td}, \mathsf{td}'$, and returns vk to $\mathcal{A}$.

Signing queries: Let $k \leftarrow 1$ be a counter for the number of datasets queried by $\mathcal{A}$. For every new queried dataset Δ, $\mathcal{B}$ creates a list T_Δ of tuples $(\tau, \boldsymbol{m}, \sigma)$, which collects all the label/message pairs queried by the adversary on Δ, and the respectively generated signatures. Moreover, whenever the k-th new dataset Δ_k is queried, $\mathcal{B}$ samples a random $\xi_k \xleftarrow{\$} \mathbb{Z}_p$, computes $Z_k = (g_2^z)^{\xi_k}$ and stores ξ_k. Note that all the values $\{Z_k\}_{k\in[Q]}$ are random in $\mathbb{G}_2$ and thus are distributed exactly as in Game 3.

Given a signing query $(\Delta, \tau, \boldsymbol{m})$ such that $\Delta = \Delta_k$ is the k-th dataset, $\mathcal{B}$ proceeds as follows. First, it runs $\boldsymbol{c}_\tau \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, \tau)$, and $\boldsymbol{c}'_j \leftarrow \mathsf{H}'.\mathsf{TrapEval}(\mathsf{td}', j)$ for all $j = 1$ to T. Notice that by the weak $(\mathsf{poly}, 1, 2)$ degree-2 programmability of H' we have $c'_{j,2} = 0$ for all $j \neq \nu$, whereas $c'_{\nu,2} \neq 0$.

Therefore, $\mathcal{B}$ samples a random $\rho_\tau \xleftarrow{\$} \mathbb{Z}_p$ and computes

$$R_\tau = g_1^{-c_\tau(y) - \sum_{j=1}^T c'_{j,0} m_j} \cdot (g_1^z)^{\rho_\tau}, \quad S_\tau = \left(g_1^{\rho_\tau} \cdot g_1^{\sum_{j=1}^T c'_{j,1} m_j} \cdot (g_1^z)^{c'_{\nu,2} m_\nu} \right)^{\frac{1}{\xi_k}}$$

As one can see, the value R_τ is a uniformly distributed $\mathbb{G}_1$ element as in Game 3. Moreover, S_τ is a correctly distributed signature since

$$\begin{aligned}
S_\tau &= \left(g_1^{\rho_\tau} \cdot g_1^{\sum_{j=1}^T c'_{j,1} m_j} \cdot (g_1^z)^{c'_{\nu,2} m_\nu} \right)^{\frac{1}{\xi_k}} = \left(g_1^{z\rho_\tau} \cdot g_1^{\sum_{j=1}^T z\, c'_{j,1} m_j} \cdot (g_1^{z^2})^{c'_{\nu,2} m_\nu} \right)^{\frac{1}{z_k}} \\
&= \left(g_1^{c_\tau(y)} \cdot g_1^{-c_\tau(y) - \sum_{j=1}^T c'_{j,0} m_j} \cdot (g_1^z)^{\rho_\tau} \cdot g_1^{\sum_{j=1}^T c'_{j,0} m_j} \cdot g_1^{\sum_{j=1}^T z c'_{j,1} m_j} \cdot (g_1^{z^2})^{c'_{\nu,2} m_\nu} \right)^{\frac{1}{z_k}} \\
&= \left(\mathsf{H}(\tau) \cdot R_\tau \cdot g_1^{\sum_{j=1}^T (c'_{j,0} + c'_{j,1} z + c'_{j,2} z^2) m_j} \right)^{\frac{1}{z_k}} \\
&= \left(\mathsf{H}(\tau) \cdot R_\tau \cdot \prod_{j=1}^T g_1^{c'_j(z) m_j} \right)^{\frac{1}{z_k}} = \left(\mathsf{H}(\tau) \cdot R_\tau \cdot \prod_{j=1}^T \mathsf{H}'(j)^{m_j} \right)^{\frac{1}{z_k}}
\end{aligned}$$

Finally, $\mathcal{B}$ returns to $\mathcal{A}$ the signature $\sigma = (\sigma_\Delta, Z_k, R_\tau, S_\tau)$, where $\sigma_\Delta \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}', \Delta | Z_k)$.

Forgery: Let $(\mathcal{P}^*_{\Delta^*}, \sigma^*, \boldsymbol{m}^*)$ be the forgery returned by the adversary. $\mathcal{B}$ proceeds exactly as the challenger in Game 3 in order to compute $\hat{R}, \hat{S}, \hat{\boldsymbol{m}}$. If Bad_3 occurs, since $(\mathcal{P}^*_{\Delta^*}, \sigma^*, \boldsymbol{m}^*)$ verifies correctly the following two equations hold

$$e(S^*, Z_\mu) = \Lambda \cdot e(R^*, g_2) \cdot \prod_{j=1}^T \mathsf{H.PubEval}'(\mathsf{pek}', j)^{m^*_j},$$

$$e(\hat{S}, Z_\mu) = \Lambda \cdot e(\hat{R}, g_2) \cdot \prod_{j=1}^T \mathsf{H.PubEval}'(\mathsf{pek}', j)^{\hat{m}_j}$$

where $\Lambda = \prod_{\tau \in \mathcal{L}^*} \mathsf{H.PubEval}(\mathsf{pek}, \tau)^{f^*_\tau}$. If we divide the two equations and consider that, by definition of Bad_3, it holds $S^* = \hat{S}$, then we obtain

$$\frac{\hat{R}}{R^*} = \prod_{j=1}^T \mathsf{H}'(j)^{m^*_j - \hat{m}_j}$$

By correctness of the trapdoor algorithms of H' we know that $\mathsf{H}'(j) = g_1^{c'_{j,0}+c'_{j,1}z+c'_{j,2}z^2}$ where $c'_{j,2} = 0$ for all $j \neq \nu$ whereas $c'_{\nu,2} \neq 0$. Furthermore, since the simulation provided to $\mathcal{A}$ so far was distributed statistically close to the real execution of Game 3 (close by a factor $\gamma + \gamma'$ due to the use of $\mathsf{TrapGen}$ in H and H'), ν is information-theoretically hidden to $\mathcal{A}$. Hence,

$$\frac{\hat{R}}{R^*} = g_1^{\sum_{j=1}^{T}(c'_{j,0}+c'_{j,1}z+c'_{j,2}z^2)(m^*_j-\hat{m}_j)} = g_1^{\sum_{j=1}^{T}(c'_{j,0}+c'_{j,1}z)(m^*_j-\hat{m}_j)+c'_{\nu,2}z^2(m^*_\nu-\hat{m}_\nu)}$$

Since $\boldsymbol{m}^* \neq \hat{\boldsymbol{m}}$ there must exist an index $\nu' \in [T]$ such that $m^*_{\nu'} \neq \hat{m}_{\nu'}$. If $\nu' \neq \nu$ then $\mathcal{B}$ aborts, otherwise it computes

$$g_1^{z^2} = \left(\frac{\hat{R} \cdot g_1^{-\sum_{j=1}^{T}(c'_{j,0}+c'_{j,1}z)(m^*_j-\hat{m}_j)}}{R^*} \right)^{\frac{1}{c'_{\nu,2}(m^*_\nu-\hat{m}_\nu)}}$$

It is easy to see that if $\mathcal{B}$ does not abort, $\mathcal{B}$ is able to compute the solution $g_1^{z^2}$ of the 1-DHI problem. The probability that $\mathcal{B}$ does not abort is $\Pr[\nu' = \nu] = 1/T$ since ν is uniformly distributed and completely hidden from the view of $\mathcal{A}$. In conclusion, we have that if $\Pr[\mathsf{Bad}_3] \geq \epsilon$ then $\mathcal{B}$ has advantage at least $\epsilon/T - \gamma - \gamma'$. □

Lemma 6. $\Pr[G_3(\mathcal{A})] = Q \cdot \Pr[G_4(\mathcal{A})]$

Proof. First, note that $\Pr[G_4(\mathcal{A})] = \Pr[G_4(\mathcal{A}) \wedge \mathsf{Bad}_4] + \Pr[G_4(\mathcal{A}) \wedge \neg\mathsf{Bad}_4] = \Pr[G_4(\mathcal{A}) | \neg\mathsf{Bad}_4] \Pr[\neg\mathsf{Bad}_4]$ since Game 4 outputs 0 whenever Bad_4 occurs. Second, observe that when Bad_4 does not occur (i.e., the challenger guesses correctly the query index μ of the dataset Δ^*) then the outcome of Game 4 is identical to the one of Game 3, i.e., $\Pr[G_4(\mathcal{A}) | \neg\mathsf{Bad}_4] = \Pr[G_3(\mathcal{A})]$. Since μ is chosen uniformly at random and is completely hidden to $\mathcal{A}$ we have that $\Pr[\neg\mathsf{Bad}_4] = 1/Q$, from which the lemma follows.

Lemma 7. *If* H *is simply* $(1, \gamma)$*-programmable, and* H' *is weakly* $(\mathsf{poly}, 1, 2, \gamma', \delta')$*-programmable, then for every PPT* $\mathcal{A}$ *running in Game 5 there exists a PPT simulator* $\mathcal{B}$ *such that* $\Pr[\mathsf{Bad}_5] \leq T \cdot \mathbf{Adv}^{2\text{-}DHI}_{\mathcal{B}}(\lambda) + \gamma + \gamma'$.

Proof. Assume there exists a PPT adversary $\mathcal{A}$ such that $\Pr[\mathsf{Bad}_5] \geq \epsilon$. Then we show how to build a PPT simulator $\mathcal{B}$ that breaks the 2-DHI assumption in $\mathbb{G}_1$ with advantage greater than $\epsilon/T - \gamma - \gamma'$.

$\mathcal{B}$ takes as input a tuple $(g_1, g_2, g_1^z, g_2^z, g_1^{z^2}, g_2^{z^2})$, and its goal is to compute $g_1^{1/z}$. To do so $\mathcal{B}$ proceeds as follows.

Setup: $\mathcal{B}$ proceeds as the challenger in Game 5 by choosing a random index $\mu \xleftarrow{\$} [Q]$. Second, $\mathcal{B}$ picks a random $y \xleftarrow{\$} \mathbb{Z}_p$ and runs $(\mathsf{td}, \mathsf{pek}) \xleftarrow{\$} \mathsf{H}.\mathsf{TrapGen}(1^\lambda, \mathsf{bgp}, g_1, g_1, g_2, g_2^y)$. Note that since $\mathcal{B}$ had set $h_1 = g_1$, the polynomials c_X generated by $\mathsf{H}.\mathsf{TrapEval}(\mathsf{td}, X)$ will be univariate, degree-1, polynomials $c_X(y)$. Next, it chooses a random index $\nu \xleftarrow{\$} [T]$, which represents a guess on the index where the message vector $\boldsymbol{m}^*$ returned by the adversary in the forgery will differ from the "correct" result $\hat{\boldsymbol{m}}$. It runs the trapdoor generation (for weak $(\mathsf{poly}, 1, 2)$-programmability) of the asymmetric hash function H' – $(\mathsf{td}', \mathsf{pek}') \xleftarrow{\$} \mathsf{H}'.\mathsf{TrapGen}(1^\lambda, \mathsf{bgp}, g_1, g_1^z, g_2, g_2^z, \nu)$ – providing ν as the input on which the coefficient $c_{\nu,0} \neq 0$. Notice that by giving $h_1 = g_1^z, h_2 = g_2^z$

to $\mathsf{H}'.\mathsf{TrapGen}$, the polynomials generated $\mathsf{H}'.\mathsf{TrapEval}(\mathsf{td}', X)$ will be univariate polynomials $c_X(z) = c_{X,0} + c_{X,1}z + c_{X,2}z^2$.

Finally, it generates the keys $(\mathsf{sk}', \mathsf{vk}')$ of the scheme Σ', sets $\mathsf{vk} = (\mathsf{vk}', \mathsf{pek}, \mathsf{pek}')$, stores $\mathsf{sk}', \mathsf{td}, \mathsf{td}'$, and returns vk to $\mathcal{A}$.

Signing queries: Let $k \leftarrow 1$ be a counter for the number of datasets queried by $\mathcal{A}$. For every new queried dataset Δ, $\mathcal{B}$ creates a list T_Δ of tuples $(\tau, \boldsymbol{m}, \sigma)$, which collects all the label/message pairs queried by the adversary on Δ and the respectively generated signatures. Moreover, whenever the k-th new dataset Δ_k is queried, $\mathcal{B}$ does the following: if $k = \mu$ it samples a random $\xi \xleftarrow{\$} \mathbb{Z}_p$, computes $Z_\mu = (g_2^z)^\xi$ and stores ξ; if $k \neq \mu$, $\mathcal{B}$ samples directly a random $z_k \xleftarrow{\$} \mathbb{Z}_p$, computes $Z_k = g_2^{z_k}$ and stores z_k. Note that all the values $\{Z_k\}_{k \in [Q]}$ are random in $\mathbb{G}_2$ and thus are distributed exactly as in Game 5.

Given a signing query $(\Delta, \tau, \boldsymbol{m})$ such that $\Delta = \Delta_k$ is the k-th dataset, $\mathcal{B}$ first computes $\sigma_{\Delta_k} \leftarrow \mathsf{Sign}(\mathsf{sk}', \Delta_k, Z_k)$, and then proceeds as follows.

- If $k \neq \mu$, $\mathcal{B}$ runs $\boldsymbol{c}_\tau \leftarrow \mathsf{H}.\mathsf{TrapEval}(\mathsf{td}, \tau)$, and $\boldsymbol{c}'_j \leftarrow \mathsf{H}'.\mathsf{TrapEval}(\mathsf{td}', j)$ for all $j = 1$ to T. It samples $R_\tau \xleftarrow{\$} \mathbb{G}_1$, and computes

$$S_\tau = \left(g_1^{c_\tau(y)} \cdot R_\tau \cdot \prod_{j=1}^{T} g_1^{c'_j(z) m_j} \right)^{\frac{1}{z_k}}$$

  In particular, note that every $g_1^{c'_j(z)}$ can be computed by $\mathcal{B}$ using the values $g_1^z, g_1^{z^2}$.
- If $k = \mu$, $\mathcal{B}$ runs $\boldsymbol{c}_\tau \leftarrow \mathsf{H}.\mathsf{TrapEval}(\mathsf{td}, \tau)$, and $\boldsymbol{c}'_j \leftarrow \mathsf{H}'.\mathsf{TrapEval}(\mathsf{td}', j)$ for all $j = 1$ to T. Notice that by the weak $(\mathsf{poly}, 1, 2)$-programmability of H', $c'_{j,0} = 0$ for all $j \neq \nu$, whereas $c'_{\nu,0} \neq 0$. Therefore, $\mathcal{B}$ samples a random $\rho_\tau \xleftarrow{\$} \mathbb{Z}_p$ and computes

$$R_\tau = g_1^{-c_\tau(y) - c_{\nu,0} m_\nu} \cdot (g_1^z)^{\rho_\tau}, \quad S_\tau = \left(g_1^{\rho_\tau} \cdot g_1^{\sum_{j=1}^{T} (c'_{j,1} + c'_{j,2} z) m_j} \right)^{\frac{1}{\xi}}$$

  As one can see, the value R_τ is a uniformly distributed $\mathbb{G}_1$ element as in Game 5. Moreover, S_τ is a correctly distributed signature since

$$\begin{aligned}
S_\tau &= \left(g_1^{\rho_\tau} \cdot g_1^{\sum_{j=1}^{T} (c'_{j,1} + c'_{j,2} z) m_j} \right)^{\frac{1}{\xi}} = \left(g_1^{z\rho_\tau} \cdot g_1^{\sum_{j=1}^{T} (c'_{j,1} z + c'_{j,2} z^2) m_j} \right)^{\frac{1}{z\xi}} \\
&= \left(g_1^{c_\tau(y)} \cdot g_1^{-c_\tau(y) - c'_{\nu,0} m_\nu} \cdot g_1^{z\rho_\tau} \cdot g_1^{\sum_{j=1}^{T} (c'_{j,0} + z c'_{j,1} + c'_{j,2} z^2) m_j} \right)^{\frac{1}{z\xi}} \\
&= \left(\mathsf{H}(\tau) \cdot R_\tau \cdot g_1^{\sum_{j=1}^{T} (c'_{j,0} + z c'_{j,1} + c'_{j,2} z^2) m_j} \right)^{\frac{1}{z\xi}} \\
&= \left(\mathsf{H}(\tau) \cdot R_\tau \cdot \prod_{j=1}^{T} \mathsf{H}'(j)^{m_j} \right)^{\frac{1}{z_\mu}}
\end{aligned}$$

Finally, $\mathcal{B}$ returns to $\mathcal{A}$ the signature $\sigma = (\sigma_{\Delta_k}, Z_k, R_\tau, S_\tau)$.

Forgery: Let $(\mathcal{P}^*_{\Delta^*}, \sigma^*, \boldsymbol{m}^*)$ be the forgery returned by the adversary. $\mathcal{B}$ proceeds exactly as the challenger in Game 5 in order to compute $\hat{R}, \hat{S}, \hat{\boldsymbol{m}}$. If Bad_5 occurs, since $(\mathcal{P}^*_{\Delta^*}, \sigma^*, \boldsymbol{m}^*)$ verifies correctly the following two equations hold

$$e(S^*, Z_\mu) = \Lambda \cdot e(R^*, g_2) \cdot \prod_{j=1}^{T} \mathsf{H.PubEval}'(\mathsf{pek}', j)^{m^*_j},$$

$$e(\hat{S}, Z_\mu) = \Lambda \cdot e(\hat{R}, g_2) \cdot \prod_{j=1}^{T} \mathsf{H.PubEval}'(\mathsf{pek}', j)^{\hat{m}_j}$$

where $\Lambda = \prod_{\tau \in \mathcal{L}^*} \mathsf{H.PubEval}(\mathsf{pek}, \tau)^{f^*_\tau}$. If we divide the two equations and consider that by definition of Bad_3, $S^* \neq \hat{S}$ but $R^* = \hat{R}$ we obtain

$$\frac{S^*}{\hat{S}} = \left(\prod_{j=1}^{T} \mathsf{H}'(j)^{m^*_j - \hat{m}_j} \right)^{\frac{1}{z\xi}}$$

By using the $\mathsf{H}'.\mathsf{TrapEval}$ algorithm we know that $\mathsf{H}'(j) = g_1^{c'_{j,0} + z c'_{j,1} + c'_{j,2} z^2}$ where $c'_{j,0} = 0$ for all $j \neq \nu$. Hence,

$$\frac{S^*}{\hat{S}} = \left(g_1^{1/(z\xi)}\right)^{\sum_{j=1}^{T} (c'_{j,0} + z c'_{j,1} + c'_{j,2} z^2)(m^*_j - \hat{m}_j)} = \left(g_1^{1/(z\xi)}\right)^{c'_{\nu,0}(m^*_\nu - \hat{m}_\nu)} g_1^{\sum_{j=1}^{T} (c'_{j,1} + c'_{j,2} z)(m^*_j - \hat{m}_j)/\xi}$$

Since $\boldsymbol{m}^* \neq \hat{\boldsymbol{m}}$ there must exist an index $\nu' \in [T]$ such that $m^*_{\nu'} \neq \hat{m}_{\nu'}$. If $\nu' \neq \nu$ then $\mathcal{B}$ aborts, otherwise it computes

$$g_1^{1/z} = \left(\frac{S^* \cdot g_1^{-\sum_{j=1}^{T} (c'_{j,1} + c'_{j,2} z)(m^*_j - \hat{m}_j)/\xi}}{\hat{S}} \right)^{\frac{\xi}{c'_{\nu,0}(m^*_\nu - \hat{m}_\nu)}}$$

Note that the simulation of Game 5 provided by $\mathcal{B}$ to $\mathcal{A}$ is statistically close (by a factor $\gamma + \gamma'$ due to the use of $\mathsf{TrapGen}$ in H and H') to the real execution of Game 5. Then, it is easy to see that if $\mathcal{B}$ does not abort, it is able to compute the solution of the 2-DHI problem $g_1^{1/z}$. The probability that $\mathcal{B}$ does not abort is $\Pr[\nu' = \nu] = 1/T$ since ν is uniformly distributed and completely hidden from the view of $\mathcal{A}$. In conclusion, we have that if $\Pr[\mathsf{Bad}_5] \geq \epsilon$ then $\mathcal{B}$ has advantage at least $\epsilon/T - \gamma - \gamma'$. □

Lemma 8. *If the asymmetric hash function* H *has* $(1, \gamma, \epsilon)$*-programmable pseudo-randomness then* $|\Pr[G_6(\mathcal{A})] - \Pr[G_7(\mathcal{A})]| \leq \epsilon$.

Proof. We do the proof by contradiction. Assume there exists a PPT adversary $\mathcal{A}$ such that $|\Pr[G_1] - \Pr[G_2]| \geq \epsilon$. Then we show how to build a PPT simulator $\mathcal{B}$ that breaks the programmable pseudo-randomness of H with advantage ϵ. We build such a simulator $\mathcal{B}$ as follows:

Setup: $\mathcal{B}$ first receives the bilinear group parameters bgp, which includes the two generators g_1, g_2. $\mathcal{B}$ proceeds as the challenger in Game 6 by choosing a random index $\mu \xleftarrow{\$} [Q]$ and a random $z_\mu \xleftarrow{\$} \mathbb{Z}_p$. It also prepares $Z_\mu = g_2^{z_\mu}$. Then it sets $h_1 = g_1 \in \mathbb{G}_1$, $h_2 = Z_\mu$ and returns (h_1, h_2) to

its challenger. It receives back a public key pek for H, and also gets access to an oracle that on input τ outputs $\mathsf{H}(\tau)$ and either $g_1^{c_{\tau,0}}$ or $g_1^{r_\tau}$.
$\mathcal{B}$ then queries its oracle on all inputs $\tau \in [N]$ and stores all the answers $\{Y_\tau, C_\tau\}_{\tau\in[N]}$. Moreover, $\mathcal{B}$ chooses in advance the values $z_k \xleftarrow{\$} \mathbb{Z}_p$, $\forall k \in [Q] \setminus \{\mu\}$, and stores $\{z_k, Z_k = g_2^{z_k}\}$. Next, it generates the keys $(\mathsf{sk}', \mathsf{vk}')$ of the scheme Σ', the keys $(\mathsf{sek}', \mathsf{pek}')$ of the asymmetric hash H', it sets $\mathsf{vk} = (\mathsf{vk}', \mathsf{pek}, \mathsf{pek}')$, stores $\mathsf{sk}', \mathsf{sek}'$, and returns vk to $\mathcal{A}$.

Signing queries: Let $k \leftarrow 1$ be a counter for the number of datasets queried by $\mathcal{A}$. For every new queried dataset Δ, $\mathcal{B}$ creates a list T_Δ of tuples $(\tau, \boldsymbol{m}, \sigma)$, which collects all the label/message pairs queried by the adversary on Δ and the respectively generated signatures.
On the k-th query $(\Delta, \tau, \boldsymbol{m})$, $\mathcal{B}$ proceeds as follows:
- If $k \neq \mu$, $\mathcal{B}$ first generates the signature σ_Δ on (Δ_k, Z_k) using the secret key sk'. Next, it chooses a random value $R_\tau \xleftarrow{\$} \mathbb{G}_1$ and computes $S_\tau = (Y_\tau \cdot R_\tau \prod_{j=1}^{T} \mathsf{H.PriEval}'(\mathsf{sk}', j)^{m_j})^{1/z_k}$.
- If $k = \mu$, $\mathcal{B}$ works exactly as above except that it sets $R_\tau = C_\tau^{-1}$.

Finally, $\mathcal{B}$ returns to $\mathcal{A}$ the signature $\sigma = (\sigma_\Delta, Z_k, R_\tau, S_\tau)$, where $\sigma_\Delta \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}', \Delta | Z_k)$.

Forgery: Let $(\mathcal{P}^*_{\Delta^*}, \sigma^*, \boldsymbol{m}^*)$ be the forgery returned by the adversary. $\mathcal{B}$ proceeds exactly as the challenger in Game 7 in order to determine the outcome of the experiment, and outputs 0 or 1 accordingly.

It is easy to see that if $\mathcal{B}$ receives from its oracle values C_τ with the pseudorandom distribution, then $\mathcal{B}$ is perfectly simulating Game 7 to $\mathcal{A}$. Otherwise, if the values C_τ are random then $\mathcal{B}$ is simulating Game 6. Therefore,

$$|\Pr[\mathbf{Exp}^{PRH-0}_{\mathcal{B},\mathsf{H}} = 1] - \Pr[\mathbf{Exp}^{PRH-1}_{\mathcal{B},\mathsf{H}} = 1]| = |\Pr[G_7(\mathcal{A})] - \Pr[G_6(\mathcal{A})]| \;=\; \epsilon$$

□

To conclude the proof, we are left with showing that any PPT adversary has negligible probability of winning in Game 7. We show this in the following lemma where we prove that this holds under the Flexible Diffie-Hellman Inversion Assumption (FDHI) given in Definition 4.

Lemma 9. *If* H *has* $(1, \gamma, \epsilon)$*-programmable pseudo-randomness and* H' *is* $(\mathsf{poly}, 0, 1, \gamma', \delta')$*-programmable, then for any PPT* $\mathcal{A}$ *running in Game 7 there is a PPT* $\mathcal{B}$ *against the FDHI assumption such that* $\Pr[G_7(\mathcal{A})] = \mathbf{Adv}^{FDHI}_{\mathcal{B}}(\lambda) + \gamma + \gamma'$.

Proof. Assume that $\mathcal{A}$ is a PPT adversary such that $\Pr[G_7(\mathcal{A})] = \epsilon$. Then we show how to build a PPT simulator $\mathcal{B}$ which uses $\mathcal{A}$ to solve the FDHI problem with advantage ϵ. $\mathcal{B}$ receives an FDHI instance $(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}})$ and works as follows.

Setup: $\mathcal{B}$ proceeds as the challenger in Game 6 by choosing a random index $\mu \xleftarrow{\$} [Q]$. Next, it runs the trapdoor generation algorithm for the programmable pseudo-randomness of H, $(\mathsf{td}, \mathsf{pek}) \xleftarrow{\$} \mathsf{H.TrapGen}(1^\lambda, \mathsf{bgp}, g_1, g_1, g_2, g_2^z)$, and the trapdoor generation algorithm for the $(\mathsf{poly}, 0, 1)$-programmability of H', $(\mathsf{td}', \mathsf{pek}') \xleftarrow{\$} \mathsf{H}'.\mathsf{TrapGen}(1^\lambda, \mathsf{bgp}, g_1, g_1, g_2, g_2^z)$.
Finally, it generates the keys $(\mathsf{sk}', \mathsf{vk}')$ of the scheme Σ', sets $\mathsf{vk} = (\mathsf{vk}', \mathsf{pek}, \mathsf{pek}')$, stores $\mathsf{sk}', \mathsf{td}, \mathsf{td}'$, and returns vk to $\mathcal{A}$.

Signing queries: Let $k \leftarrow 1$ be a counter for the number of datasets queried by $\mathcal{A}$. For every new queried dataset Δ, $\mathcal{B}$ creates a list T_Δ of tuples $(\tau, \boldsymbol{m}, \sigma)$, which collects all the label/message pairs queried by the adversary on Δ and the respectively generated signatures.

Moreover whenever the k-th new dataset Δ_k is queried, $\mathcal{B}$ does the following: if $k = \mu$ it samples a random $\xi_\mu \xleftarrow{\$} \mathbb{Z}_p$, computes $Z_\mu = (g_2^z)^{\xi_\mu}$ and stores Z_μ, ξ_μ; if $k \neq \mu$, $\mathcal{B}$ samples a random $\xi_k \xleftarrow{\$} \mathbb{Z}_p$, and computes $Z_k = (g_2^v)^{\xi_k}$ and stores Z_k, ξ_k. Note that all the values $\{Z_k\}_{k\in[Q]}$ are random in $\mathbb{G}_2$ and thus are distributed exactly as in Game 7.

Given a signing query $(\Delta, \tau, \boldsymbol{m})$ such that $\Delta = \Delta_k$ is the k-th dataset, $\mathcal{B}$ first computes $\sigma_{\Delta_k} \leftarrow \mathsf{Sign}(\mathsf{sk}', \Delta_k, Z_k)$, and then proceeds as follows.

– If $k \neq \mu$: $\mathcal{B}$ runs $\boldsymbol{c}_\tau \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, \tau)$, and $\boldsymbol{c}'_j \leftarrow \mathsf{H'.TrapEval}(\mathsf{td}', j)$ for all $j = 1$ to T. Notice that by the $(1, \gamma, \epsilon)$-programmability of H we have that $\boldsymbol{c}_\tau$ is a degree-1 polynomial in z: $c_\tau(z) = c_{\tau,0} + c_{\tau,1} z$. Similarly, the polynomials $\boldsymbol{c}'_j$ generated by $\mathsf{H'.TrapEval}$ are also of degree 1 in the sole variable z, and by the $(\mathsf{poly}, 0, 1)$-programmability of H' they are all such that $c'_{j,0} = 0$, i.e., $c'_j(z) = c'_{j,1} z$.

Next, it samples $\rho_\tau \xleftarrow{\$} \mathbb{Z}_p$, computes

$$R_\tau = g_1^{-c_{\tau,0}} \cdot (g_1^r)^{\rho_\tau}, \quad S_\tau = \left((g_1^{\frac{z}{v}})^{c_{\tau,1}} \cdot (g_1^{\frac{r}{v}})^{\rho_\tau} \cdot (g_1^{\frac{z}{v}})^{\sum_{j=1}^T c'_{j,1} m_j} \right)^{\frac{1}{\xi_k}}$$

and returns $\sigma = (Z_k, \sigma_{\Delta_k}, R_\tau, S_\tau)$ to $\mathcal{A}$.

Note that the signature is correctly distributed as in Game 7, since R_τ is a uniformly distributed $\mathbb{G}_1$ element, and

$$\begin{aligned}
S_\tau &= \left((g_1^{\frac{z}{v}})^{c_{\tau,1}} \cdot (g_1^{\frac{r}{v}})^{\rho_\tau} \cdot (g_1^{\frac{z}{v}})^{\sum_{j=1}^T c'_{j,1} m_j} \right)^{\frac{1}{\xi_k}} \\
&= \left((g_1^z)^{c_{\tau,1}} \cdot (g_1^r)^{\rho_\tau} \cdot (g_1^z)^{\sum_{j=1}^T c'_{j,1} m_j} \right)^{\frac{1}{v\xi_k}} \\
&= \left(g_1^{c_{\tau,0}} \cdot (g_1^z)^{c_{\tau,1}} \cdot g_1^{-c_{\tau,0}} \cdot (g_1^r)^{\rho_\tau} \cdot g_1^{\sum_{j=1}^T (c'_{j,1} z) m_j} \right)^{\frac{1}{z_k}} \\
&= \left(\mathsf{H}(\tau) \cdot R_\tau \cdot g_1^{\sum_{j=1}^T c'_j(z) m_j} \right)^{\frac{1}{z_k}} \;=\; \left(\mathsf{H}(\tau) \cdot R_\tau \cdot \prod_{j=1}^T g_1^{c'_j(z) m_j} \right)^{\frac{1}{z_k}} \\
&= \left(\mathsf{H}(\tau) \cdot R_\tau \cdot \prod_{j=1}^T \mathsf{H}'(j)^{m_j} \right)^{\frac{1}{z_k}}
\end{aligned}$$

– If $k = \mu$: $\mathcal{B}$ runs $\boldsymbol{c}_\tau \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, \tau)$, and $\boldsymbol{c}'_j \leftarrow \mathsf{H'.TrapEval}(\mathsf{td}', j)$ for all $j = 1$ to T. It sets $R_\tau = g_1^{-c_{\tau,0}}$ and computes

$$S_\tau = \left(g_1^{c_{\tau,1}} \cdot g_1^{\sum_{j=1}^T c'_{j,1} m_j} \right)^{\frac{1}{\xi_\mu}}$$

and returns $\sigma = (Z_\mu, \sigma_{\Delta_\mu}, R_\tau, S_\tau)$ to $\mathcal{A}$.

As one can check, such signature is distributed as a signature in Game 7: $R_\tau = g_1^{-c_{\tau,0}}$ as in the definition of Game 7 (for the μ-th dataset) while for S_τ we have

$$\begin{aligned}
S_\tau &= \left(g_1^{c_{\tau,1}} \cdot g_1^{\sum_{j=1}^{T} c'_{j,1} m_j}\right)^{\frac{1}{\xi_\mu}} = \left(g_1^{z c_{\tau,1}} \cdot g_1^{z \sum_{j=1}^{T} c'_{j,1} m_j}\right)^{\frac{1}{z\xi_\mu}} \\
&= \left(g_1^{c_{\tau,0}} \cdot g_1^{z c_{\tau,1}} \cdot g_1^{-c_{\tau,0}} \cdot g_1^{\sum_{j=1}^{T} (c'_{j,1} z) m_j}\right)^{\frac{1}{z_\mu}} \\
&= \left(\mathsf{H}(\tau) \cdot R_\tau \cdot g_1^{\sum_{j=1}^{T} c'_j(z) m_j}\right)^{\frac{1}{z_\mu}} = \left(\mathsf{H}(\tau) \cdot R_\tau \cdot \prod_{j=1}^{T} g_1^{c'_j(z) m_j}\right)^{\frac{1}{z_\mu}} \\
&= \left(\mathsf{H}(\tau) \cdot R_\tau \cdot \prod_{j=1}^{T} \mathsf{H}'(j)^{m_j}\right)^{\frac{1}{z_\mu}}
\end{aligned}$$

Forgery: Let $(\mathcal{P}^*_{\Delta^*}, \sigma^*, \boldsymbol{m}^*)$ be the forgery returned by the adversary. $\mathcal{B}$ proceeds exactly as the challenger in Game 7 in order to compute $\hat{R}, \hat{S}, \hat{\boldsymbol{m}}$.

By definition, if Game 7 outputs 1, since $(\mathcal{P}^*_{\Delta^*}, \sigma^*, \boldsymbol{m}^*)$ verifies correctly, the following two equations hold

$$e(S^*, Z_\mu) = \Lambda \cdot e(R^*, g_2) \cdot \prod_{j=1}^{T} \mathsf{H.PubEval}'(\mathsf{pek}', j)^{m^*_j},$$

$$e(\hat{S}, Z_\mu) = \Lambda \cdot e(\hat{R}, g_2) \cdot \prod_{j=1}^{T} \mathsf{H.PubEval}'(\mathsf{pek}', j)^{\hat{m}_j}$$

where $\Lambda = \prod_{\tau \in \mathcal{L}^*} \mathsf{H.PubEval}(\mathsf{pek}, \tau)^{f^*_\tau}$. If we divide the two equations and consider that by definition of Game 7, it must be $S^* \neq \hat{S}$ and $R^* \neq \hat{R}$, then we obtain:

$$\begin{aligned}
\frac{S^*}{\hat{S}} &= \left(\frac{R^*}{\hat{R}} \cdot \prod_{j=1}^{T} \mathsf{H}'(j)^{m^*_j - \hat{m}_j}\right)^{\frac{1}{z\xi_\mu}} = \left(\frac{R^*}{\hat{R}} \cdot \prod_{j=1}^{T} g_1^{c'_{j,1} z (m^*_j - \hat{m}_j)}\right)^{\frac{1}{z\xi_\mu}} \\
&= \left(\frac{R^*}{\hat{R}}\right)^{\frac{1}{z\xi_\mu}} \cdot \left(\prod_{j=1}^{T} g_1^{c'_{j,1}(m^*_j - \hat{m}_j)}\right)^{\frac{1}{\xi_\mu}} \qquad (2)
\end{aligned}$$

Therefore $\mathcal{B}$ can compute

$$W = \frac{R^*}{\hat{R}}, \quad W' = \left(\frac{S^*}{\hat{S}}\right)^{\xi_\mu} \cdot g_1^{\sum_{j=1}^{T} c'_{j,1}(\hat{m}_j - m^*_j)}$$

and returns (W, W') as a solution for the FDHI assumption.

To see that (W, W') is a solution for the FDHI assumption, i.e., $W' = W^{1/z}$, observe that by equation (2) it holds

$$(W')^z = \left(\frac{S^*}{\hat{S}}\right)^{z\xi_\mu} \cdot g_1^{\sum_{j=1}^{T} z c'_{j,1}(\hat{m}_j - m_j^*)}$$

$$= \left[\left(\frac{R^*}{\hat{R}}\right)^{\frac{1}{z\xi_\mu}} \cdot \left(\prod_{j=1}^{T} g_1^{c'_{j,1}(m_j^* - \hat{m}_j)}\right)^{\frac{1}{\xi_\mu}}\right]^{z\xi_\mu} \cdot g_1^{\sum_{j=1}^{T} z c'_{j,1}(\hat{m}_j - m_j^*)} = \frac{R^*}{\hat{R}} = W$$

Note that the simulation of Game 7 provided by $\mathcal{B}$ to $\mathcal{A}$ is statistically close (by a factor $\gamma + \gamma'$ due to the use of $\mathsf{TrapGen}$ in H and H') to the real execution of Game 7. Then, it is easy to see that if Game 7 outputs 1, $\mathcal{B}$ is able to compute the solution of the FDHI problem, as described above. In conclusion, if $\Pr[G_7(\mathcal{A})] \geq \epsilon$ then $\mathcal{B}$ has advantage at least $\epsilon - \gamma - \gamma'$ in solving FDHI. □

5 Short Signatures with Shorter Public Keys from Bilinear Maps

In this section we describe how to use APHFs to construct in a generic fashion standard-model signature schemes over bilinear groups. We propose two constructions that are provably-secure under the q-Strong Diffie-Hellman [7] and the q-Diffie-Hellman [6] assumptions. These constructions are the analogues of the schemes in [26] and [25] respectively. The basic idea behind the constructions is to replace a standard $(m, 1)$-PHF with an $(m, 1, d)$-APHF. In fact, in this context, having a secretly-computable H does not raise any issue when using H in the signing procedure as the signer already uses a secret key. At the same time, for verification purposes, computing the (public) isomorphic copy of H in the target group is also sufficient. Our proof confirms that the $(m, 1, d)$-programmability can still be used to control the size of the randomness in the same way as in [26,25]. One difference in the security proof is that the schemes in [26,25] are based on the q-(S)DH assumption, where q is the number of signing queries made by the adversary, whereas ours have to rely on the $(q + d - 1)$-(S)DH problem. Since our instantiations use $d = 2$, the difference (when considering concrete security) is very minor.

When plugging into these generic constructions our new APHF, $\mathsf{H}_{\mathsf{acfs}}$, described in Section 3.2, which is $(m, 1, 2)$-programmable, we obtain schemes that, for signing ℓ-bits messages, allow for public keys of length $O(m\sqrt{\ell})$ as in [**?**].

We describe the scheme based on q-SDH in Section 4.1, and the one based on q-DH in Section 4.2. As discussed in [25], the advantage of the scheme from q-DH compared to the one from q-SDH is to be based on a weaker assumption.

5.1 A q-Strong Diffie-Hellman Based Solution

In this section we revisit the q-SDH based solution of [26]. The signature $\Sigma_{\mathsf{qSDH}} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ is as follows:

$\mathsf{KeyGen}(1^\lambda)$**.** Let λ be the security parameter, and let $\ell = \ell(\lambda)$ and $\rho = \rho(\lambda)$ be arbitrary polynomials. Our scheme can sign messages in $\{0,1\}^\ell$ using randomness in $\{0,1\}^\rho$. The key generation algorithm works as follows:

- Run $\mathsf{bgp} \xleftarrow{\$} \mathcal{G}(1^\lambda)$ to generate the bilinear groups parameters $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are asymmetric groups of prime order $p \approx 2^\lambda$, $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ are generators and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable, non-degenerate bilinear map.
- Run $(\mathsf{sek}, \mathsf{pek}) \xleftarrow{\$} \mathsf{H.Gen}(1^\lambda, \mathsf{bgp})$ to generate the keys of the asymmetric hash function.
- Choose a random $x \xleftarrow{\$} \mathbb{Z}_p^*$ and set $X \leftarrow g_2^x$. Return $\mathsf{vk} = (\mathsf{bgp}, \mathsf{pek}, X)$ and $\mathsf{sk} = (\mathsf{sek}, x)$.

$\mathsf{Sign}(\mathsf{sk}, M)$**.** The signing algorithm takes as input the secret key sk, and a message $M \in \{0,1\}^\ell$. It starts by generating a random $r \xleftarrow{\$} \{0,1\}^\rho$. Next, it computes $\sigma = \mathsf{H.PriEval}(\mathsf{sek}, M)^{\frac{1}{x+r}}$ and outputs (σ, r).

$\mathsf{Ver}(\mathsf{vk}, M, (\sigma, r))$**.** To check that (σ, r) is a valid signature, check that r is of length ρ and that

$$e(\sigma, X \cdot g_2^r) = \mathsf{H.PubEval}(\mathsf{pek}, M)$$

We state the security of the scheme in the following theorem. We note that for simplicity our proof assumes an $(m, 1, d)$-APHF for $d = 2$, which matches our realization. A generalization of the theorem for a generic d can be immediately obtained, in which case one would rely on the $(q + d - 1)$-SDH assumption.

Theorem 11. *Assume that $\mathcal{G}$ is a bilinear group generator such that the $(q+1)$-SDH assumption holds in $\mathbb{G}_1$ and H is $(m, 1, 2, \gamma, \delta)$-programmable, then Σ_{qSDH} is a secure signature scheme. More precisely, let $\mathcal{B}$ be an efficient (probabilistic) algorithm that runs in time t, asks (up to) q signing queries and produces a valid forgery with probability ϵ, then there exists an equally efficient algorithm $\mathcal{A}$ that confutes the $(q+1)$-SDH assumption with probability*

$$\epsilon' \geq \frac{\delta}{q}\left(\epsilon - \gamma - \frac{q}{p} - \frac{q^{m+1}}{2^{\rho m}}\right)$$

Proof. This proof is almost identical to the corresponding one from [26], we rewrite it here mainly to show how to use APHFs in place of standard PHFs.

Let $\mathcal{B}$ be an adversary against the signature scheme. Assuming that $\mathcal{B}$ asks (up to) q signing queries, and denote with M_i the i-th queried message and with (σ_i, r_i) the corresponding signature. Also, let $M^*, (\sigma^*, r^*)$ be the produced forgery. We distinguish two types of forgeries:

Type I forgery: It holds that $r^* = r_i$ for some $i \in [q]$.
Type II forgery: It holds that $r^* \neq r_i \ \forall i \in [q]$.

Notice that these two cases are mutually exclusive and completely cover the set of possible forgeries. Now we show that both types of forgeries can be used to violate the $(q+1)$-SDH assumption.

Lemma 10 (Type I forgeries). *Let $\mathcal{B}$ be a type I forger that breaks the signature scheme with advantage ϵ_1 (and making up to q signature queries). Then there exists an (equally efficient) adversary $\mathcal{A}$ that breaks the $(q+1)$-SDH assumption with advantage ϵ', where*

$$\epsilon' \geq \frac{\delta}{q}\left(\epsilon_1 - \gamma - \frac{q}{p} - \frac{q^{m+1}}{2^{\rho m}}\right)$$

We prove the lemma via a sequence of games. We denote with G_i the event that Game i outputs 1, i.e., that $\mathcal{B}$ (successfully) forges in Game i.

Game 0 This game is the standard existential unforgeability experiment $\mathbf{Exp}^{\mathsf{UF\text{-}CMA}}_{\mathcal{B},\Sigma}$. Clearly,

$$Pr[G_0] = \epsilon_1$$

Game 1 This is the same as the previous game but the parameter of the APHF are generated using $\mathsf{H.TrapGen}$ (rather than $\mathsf{H.Gen}$). More precisely, $\mathcal{A}$ runs $\mathsf{H.TrapGen}(1^\lambda, g_1, h_1, g_2, h_2)$, where g_1, g_2 are, randomly chosen, generators and $h_1 = g_1^\alpha, h_2 = g_2^\alpha$ for a randomly chosen $\alpha \xleftarrow{\$} \mathbb{Z}_p$. By the γ-closeness of the trapdoor keys, we have:

$$\Pr[G_1] \geq \Pr[G_0] - \gamma$$

Game 2 In this game we do the following changes. First, we choose the r_i's used to answer signing queries all in advance (rather than one by one when needed). Since the r_i's are chosen at random and independently anyway this change cannot affect $\mathcal{B}$'s advantage at all. Second, we modify the way g_1, h_1, g_2, h_2 are chosen when executing $\mathsf{H.TrapGen}$. Specifically, let $\hat{g}_1$ be a generator of $\mathbb{G}_1$ and $\hat{g}_2$ be a generator of $\mathbb{G}_2$. We choose $i^* \in_R \{1, \ldots, q\}$ and we set $r^* = r_{i^*}$, $R = \cup_{i=1}^q r_i$, $R^* = R \setminus \{r^*\}$ and $R^{*,i} = R \setminus \{r^*, r_i\}$. Next, we define the polynomials $p^*(z) = \prod_{r \in R^*}(z + r) \bmod p$ and $p(z) = p^*(z)(z + r^*) \bmod p$. Notice that both polynomials are of degree $\leq q$. Thus, from $\hat{g}_1, \hat{g}_1^x, \ldots, \hat{g}_1^{x^q}$ it is possible to compute $g_1 = \hat{g}_1^{p^*(x)}$ and $h_1 = \hat{g}_1^{p(x)}$. Next we set $g_2 = \hat{g}_2$, $X = g_2^x$ and $h_2 = g_2^{(x+r^*)}$. The distribution of g_1, g_2 is identical to the one in Game 1. The only difference might occur in the case $p(x) = 0$, as in this case g_1, h_1 would not be generators. By the Schwartz-Zippel lemma, however, this happens only with probability at most q/p. Thus

$$\Pr[G_2] \geq \Pr[G_1] - \frac{q}{p}$$

Game 3 Let Bad_3 be the event that the same r_i is used to sign more than m different messages. This means that if Bad_3 occurs there are, at least, $m + 1$ indices $i_1, \ldots, i_{m+1}$ such that $r_{i_1} = \ldots = r_{i_{m+1}}$. On q signing queries there might be up to $\binom{q}{m+1} \leq q^{m+1}$ such tuples. Moreover, a given tuple is of the form $r_{i_1} = \ldots = r_{i_{m+1}}$ with probability $2^\rho / 2^{\rho(m+1)}$. This means that

$$\Pr[\mathsf{Bad}_3] \leq \frac{q^{m+1}}{2^{\rho m}}$$

We modify Game 2, by assuming that the simulation aborts if Bad_3 occurs. Thus,

$$\Pr[G_3] \geq \Pr[G_2] - \frac{q^{m+1}}{2^{\rho m}}$$

Game 4 Let Bad_4 be the event that $\mathcal{B}$ outputs a value r^* such that $r^* = r_i$, but $i \neq i^*$. We modify the previous game by imposing that the simulation aborts if Bad_4 occurs. Thus,

$$\Pr[G_4] = \Pr[G_4 \wedge \neg\mathsf{Bad}_4] = \frac{1}{q}\Pr[G_3]$$

Game 5 Let Bad_5 be the event that either there is an index $i \in [q]$ such that $r_i = r^*$ such that $c_{M_i,0} \neq 0$, or it occurs $c_{M^*,0} = 0$. Game 5 proceeds as Game 4 except that it aborts if Bad_5 occurs. Using the programmability of H, we can bound the probability of Bad_5. Precisely, we have that $\Pr[\neg\mathsf{Bad}_5] \geq \delta$, from which we have

$$\Pr[G_5] = \Pr[G_5 \wedge \neg\mathsf{Bad}_5] = \delta \Pr[G_4]$$

Game 6 We further modify the simulation by using the alternative signing mechanism, from [7]. In particular, to sign the i-th queried message M_i, one proceeds as follows. First, compute $\boldsymbol{c}_{M_i} \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, M_i)$. Notice that by the setting of h_1, h_2 and by the definition of $\mathsf{TrapEval}$, $\boldsymbol{c}_{M_i}$ is a degree-2 polynomial $c_{M_i}(x + r^*)$. Let us write $c_{M_i}(x + r^*) = c_{M_i,0} + c'_{M_i}(x + r^*)$ by removing the degree-0 term. Hence, one computes

$$\begin{aligned}\sigma_i = \mathsf{H.PriEval}(\mathsf{sek}, M_i)^{\frac{1}{x+r_i}} &= \left(g_1^{c_{M_i,0}+c'_{M_i}(x+r^*)}\right)^{\frac{1}{x+r_i}} \\ &= \left(\hat{g}_1^{p^*(x)c_{M_i,0}}\hat{g}_1^{p^*(x)c'_{M_i}(x+r^*)}\right)^{\frac{1}{x+r_i}} \\ &= \hat{g}_1^{c_{M_i,0}\prod_{r\in R^*,i}(x+r)}\hat{g}_1^{c'_{M_i}(x+r^*)\prod_{r\in R^*,i}(x+r)}\end{aligned}$$

Moreover, for all the signing queries that do not cause Bad_5, notice that $c_{M_i,0} = 0$ and thus such signing queries can be answered without any explicit knowledge of x. As a consequence,

$$\Pr[G_6] = \Pr[G_5]$$

Notice also that in Game 6, we are assuming that neither Bad_5 nor Bad_4 occur. This means that, for the the forged signature (M^*, σ^*, r^*) one has that $\mathsf{H.PriEval}(\mathsf{sek}, M^*) = g_1^{c_{M^*}(x+r^*)}$ and $c_{M^*,0} \neq 0$. Using the same notation as above, using the q-SDH instance we can compute

$$y = g_1^{\frac{c'_{m_i}(x+r^*)}{x+r^*}}$$

as $c'_{m_i}(x + r^*)$ is a polynomial of degree ≤ 2 without the constant term, i.e, $c'_{m_i}(x + r^*)$ is divisible by $(x + r^*)$. We set

$$\sigma' = \left(\sigma^* \cdot y^{-1}\right)^{1/c_{m^*,0}} = g_1^{\frac{1}{x+r^*}} = \hat{g}_1^{\frac{p^*(x)}{x+r^*}}$$

Using standard techniques [7,26], σ' can be used to extract the required $\hat{g}_1^{\frac{1}{x+r^*}}$. This means that $\Pr[G_6] \leq \epsilon$. Finally, putting together the bounds from the games above yields the lemma.

Lemma 11 (Type II forgeries). *Let $\mathcal{B}$ be a type II forger that breaks the signature scheme with advantage ϵ_2 (and making up to q signature queries). Then there exist (equally efficient) adversaries $\mathcal{A}_1$, that breaks the $(q + 1)$-SDH assumption with advantage ϵ, and $\mathcal{A}_2$ that breaks the discrete logarithm problem with advantage $\epsilon_{\mathtt{DL}}$, where*

$$\epsilon + \epsilon_{\mathtt{DL}} \geq \epsilon_2 - q/p - \gamma$$

Again we prove the lemma via a sequence of games, and use G_i to denote the event that $\mathcal{B}$ (successfully) forges in Game i.

Game 0 This game is the standard existential unforgeability experiment $\mathbf{Exp}^{\mathsf{UF\text{-}CMA}}_{\mathcal{B},\Sigma}$. Clearly,

$$Pr[G_0] = \epsilon_2$$

Game 1 This is the same as Game 1 above, (i.e. the parameter of the programmable hash function are generated using $\mathsf{H.TrapGen}$ (rather than $\mathsf{H.Gen}$). Thus,

$$\Pr[G_1] \geq \Pr[G_0] - \gamma$$

Game 2 In this game we do the following changes. First, we choose the r_i's used to answer signing queries all in advance (rather than one by one when needed). Second, we modify the way g_1, h_1, g_2, h_2 are chosen when executing $\mathsf{H.TrapGen}$. Specifically, let $\hat{g}_1$ be a generator of $\mathbb{G}_1$ and $\hat{g}_2$ be a generator of $\mathbb{G}_2$. We we set $R = \cup_{i=1}^{q} r_i$. Next we define the (degree-q) polynomial $p(z) = \prod_{r \in R}(z+r) \bmod p$. From $\hat{g}_1, \hat{g}_1^x, \ldots, \hat{g}_1^{x^q}$ it is possible to compute $g_1 = \hat{g}_1^{p(x)}$ and $h_1 = \hat{g}_1^{\alpha}$ (for random $\alpha \xleftarrow{\$} \mathbb{Z}_p$). Next we set $g_2 = \hat{g}_2$, $X = g_2^x$, and $h_2 = g_2^{\alpha}$ Note that the distribution of g_1, g_2 is identical with respect to Game 1. Again, the only difference might occur in the case when $p(x) = 0$, as in this case g_1 would not be a generator. Thus

$$\Pr[G_2] \geq \Pr[G_1] - \frac{q}{p}$$

Game 3 Let M^* be the message used in the forgery, and let $c_{M^*} \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, M^*)$. We define Bad_3 as the event that $c_{M^*}(\alpha) = 0$. Then, if Bad_3 happens Game 3 aborts. It is not hard to show, that if Bad_3 occurs, then one can break the discrete log assumption (which in turn is implied by the $(q+1)$-SDH). Thus

$$\Pr[G_3] \geq \Pr[G_2] - \epsilon_{\mathtt{DL}}$$

Game 4 We further modify the simulation by using the alternative signing mechanism, from [7]. In particular, to sign the i-th message M_i one obtains $c_{M_i} \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, M_i)$ and then computes

$$\begin{aligned}\sigma_i &= \mathsf{H.PriEval}(\mathsf{sek}, M_i)^{\frac{1}{x+r_i}} = \left(g_1^{c_{M_i}(\alpha)}\right)^{\frac{1}{x+r_i}} = \left(\hat{g}_1^{p(x)c_{M_i}(\alpha)}\right)^{\frac{1}{x+r_i}} \\ &= \hat{g}_1^{c_{M_i}(\alpha)\prod_{r \in R \setminus \{r_i\}}(x+r)}\end{aligned}$$

Since all the signing queries, can be answered without any explicit knowledge of x we have that

$$\Pr[G_4] = \Pr[G_3]$$

Notice also that since we are assuming that Bad_3 does not occur we have that, from the produced forgery on M^* we can extract

$$\sigma' = (\sigma^*)^{1/c_{M^*}(\alpha)} = g_1^{\frac{1}{x+r^*}} = \hat{g}_1^{\frac{p(x)}{x+r^*}}$$

Again, by using standard techniques [7,26], σ' can be used to extract the required $\hat{g}_1^{\frac{1}{x+r^*}}$. Hence, $\Pr[G_4] \leq \epsilon$.

Finally, putting together the bounds from the games above yields the lemma.

5.2 A q-Diffie Hellman based solution

In this section we show how to revisit the q-DH based scheme of [25] in order to work with APHFs. Our construction uses a standard PHF as an additional building block. We construct a signature $\Sigma_{\mathsf{qDH}} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ as follows:

$\mathsf{KeyGen}(1^\lambda)$**.** Let λ be the security parameter, and let $\ell = \ell(\lambda)$ and $\rho = \rho(\lambda)$ be arbitrary polynomials. The scheme can sign messages in $\{0,1\}^\ell$ using randomness in $\{0,1\}^\rho$. The key generation algorithm works as follows:

- Run $\mathsf{bgp} \xleftarrow{\$} \mathcal{G}(1^\lambda)$ to generate the bilinear groups parameters $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order $p \approx 2^\lambda$, $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ are generators and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable, non-degenerate bilinear map.
- Run $(\mathsf{sek}, \mathsf{pek}) \xleftarrow{\$} \mathsf{H.Gen}(1^\lambda, \mathsf{bgp})$ to generate the keys of the asymmetric hash function.
- Let $\mathsf{D} = (\mathsf{PHF.Gen}, \mathsf{PHF.Eval})$ be a group hash function [26] over $\mathbb{G}_2$ with input length ρ such that D is programmable using the algorithms $(\mathsf{PHF.TrapGen}, \mathsf{PHF.TrapEval})$. Run $(\kappa, \tau) \leftarrow \mathsf{PHF.TrapGen}(1^\lambda, g_1, g_1^y)$, for a random $y \xleftarrow{\$} \mathbb{Z}_p$.
- Return $\mathsf{vk} = (\mathsf{bgp}, \mathsf{pek}, \kappa)$ and $\mathsf{sk} = (\mathsf{sek}, \tau, y)$. In what follows, we use the same notation of [25], and use $\mathsf{d}(r)$ as a shorthand for $(a, b) \leftarrow \mathsf{PHF.TrapEval}(\tau, r)$, $\mathsf{d}(r) = a + yb$.

$\mathsf{Sign}(\mathsf{sk}, M)$**.** The signing algorithm takes as input the secret key sk, and a message $M \in \{0,1\}^\ell$. It starts by generating a random $r \in \{0,1\}^\rho$. Next, it computes $\sigma = \mathsf{H.PriEval}(\mathsf{sek}, M)^{\frac{1}{\mathsf{d}(r)}}$ and outputs (σ, r).

$\mathsf{Ver}(\mathsf{vk}, M, (\sigma, r))$**.** To verify that (σ, r) is a valid signature, check that r is of length ρ, that $\mathsf{d}(r) \neq 0$ and that

$$e(\sigma, \mathsf{PHF.Eval}(r)) = \mathsf{H.PubEval}(\mathsf{pek}, M)$$

We prove the security of the scheme in the following theorem. We note that for simplicity our proof assumes an $(m, 1, d)$-APHF for $d = 2$, which matches our realization. A generalization of the theorem for a generic d can be immediately obtained, in which case one would rely on the $(q + d - 1)$-DH assumption.

Theorem 12. *Assume that $\mathcal{G}$ is a bilinear group generator such that the $(q+1)$-DH assumption holds in $\mathbb{G}_1$, H is an asymmetric $(m, 1, 2, \gamma, \delta)$-programmable hash function, D is a $(1, \mathsf{poly}, \gamma', \delta')$ programmable hash function over $\mathbb{G}_2$ then Σ_{qDH} is a secure signature scheme. More precisely let $\mathcal{B}$ be an efficient (probabilistic) algorithm that runs in time t, asks (up to) q signing queries and produces a valid forgery with probability ϵ_1, then there exists an equally efficient algorithm $\mathcal{A}$ that confutes the $(q+1)$-DH assumption with probability*

$$\epsilon' \geq \delta\delta' \left(\frac{\epsilon_1}{q} - \gamma - \frac{q^m}{2^{\rho m}} \right)$$

Proof. Again the proof is almost identical to the corresponding one from [26], we rewrite it here for completeness. Let $\mathcal{B}$ be an adversary against the signature scheme. Assuming that $\mathcal{B}$ asks (up to) q signing queries we denote with M_i the i-th queried message and with (σ_i, r_i) the corresponding signature. Also, let $M^*, (\sigma^*, r^*)$ be the produced forgery. We distinguish two types of forgeries

Type I forgery : It holds that $r^* = r_i$ for some $i \in [q]$.
Type II forgery : It holds that $r^* \neq r_i \ \forall i \in [q]$.

Notice that these two cases are mutually exclusive and completely cover the set of possible forgeries. Now we show that both types of forgeries can be used to violate the $(q+1)$-DH assumption.

Lemma 12 (Type I forgeries). *Let $\mathcal{B}$ be a type I forger that breaks the signature scheme with advantage ϵ_1 (and making up to q signature queries). Then there exists an (equally efficient) adversary $\mathcal{A}$ that breaks the $(q+1)$-DH assumption with advantage ϵ', where*

$$\epsilon' \geq \delta\delta' \left(\frac{\epsilon_1}{q} - \gamma - \frac{q^m}{2^{\rho m}} \right)$$

Again we prove the lemma via a sequence of games, and use G_i to denote the event that $\mathcal{B}$ (successfully) forges in Game i.

Game 0 This game is the standard existential unforgeability experiment $\mathbf{Exp}^{\mathsf{UF\text{-}CMA}}_{\mathcal{B},\Sigma}$. Clearly,

$$Pr[G_0] = \epsilon_1$$

Game 1 Let Bad_1 be the event that the same r_i is used more than m times. We change the simulation by forcing an abort if Bad_1 occurs. As done in lemma 3 we have that

$$\Pr[G_1] \geq \Pr[G_0] - \frac{q^{m+1}}{2^{\rho m}}$$

Game 2 In this game we do the following changes. First, we choose the r_i's used to answer signing queries all in advance (rather than one-by-one when needed). Since the r_i's are chosen randomly and independently anyway, this change cannot affect $\mathcal{B}$'s advantage at all. Second, we guess the index i such that $i = i^*$ and we abort if this does not happen (i.e. $\mathcal{B}$ outputs an $r^* \neq r_i$). Clearly,

$$\Pr[G_2] \geq \frac{1}{q} \Pr[G_1]$$

Game 3 In this game we do the following changes. First, the parameter of the asymmetric programmable hash function are generated using $\mathsf{H.TrapGen}$ (rather than $\mathsf{H.Gen}$). Next, we modify the way g_1, h_1, g_2, h_2 are chosen when executing $\mathsf{H.TrapGen}$. Specifically, let $\hat{g}_1$ be a generator of $\mathbb{G}_1$ and $\hat{g}_2$ be a generator of $\mathbb{G}_2$. Let $R = \cup_{i=1}^{q} r_i$, $R^* = R \setminus \{r^*\}$ and $R^{*,i} = R \setminus \{r^*, r_i\}$. We set

$$g_1 = \hat{g}_1^{\prod_{r \in R^*} \mathsf{d}(r)} \qquad h_1 = \hat{g}_1^{\prod_{r \in R} \mathsf{d}(r)} \qquad g_2 = \hat{g}_2 \qquad h_2 = \hat{g}_2^{\mathsf{d}(r^*)}$$

The γ-statistical closeness of H's trapdoor keys implies

$$\Pr[G_3] \geq \Pr[G_2] - \gamma$$

Game 4 Let Bad_4 be the event that, letting $\boldsymbol{c}_{M_i} \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, M_i)$, the following happens. Either $c_{M_i,0} \neq 0$ for some i for which $r_i = r^*$, or $c_{M^*,0} = 0$ (where $\boldsymbol{c}_{M^*} \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, M^*)$). Game 4 proceeds as Game 3 except that it it aborts if Bad_4 occurs. The programmability of H implies that

$$\Pr[G_4] = \Pr[G_3 \wedge \neg \mathsf{Bad}_4] \geq \delta \Pr[G_3]$$

Game 5 Now we change the way signing queries are answered. Whenever a message M_i is queried, the simulator computes $c_{M_i} \leftarrow \mathsf{H.TrapEval}(\mathsf{td}, m_i)$ and sets

$$\sigma_i = \hat{g}_1^{c_{M_i}(\mathsf{d}(r^*)) \prod_{r \in R^*, i} \mathsf{d}(r)}$$

Notice that it is possible to sign all the received signing queries as, by Game 4, for all $r_i = r^*$, it holds $c_{M_i,0} = 0$. Game 5 is perfectly indistinguishable from Game 4, from $\mathcal{B}$'s perspective, i.e.,

$$\Pr[G_5] = \Pr[G_4]$$

Game 6 Now for each r_i we compute $(a_i, b_i) \leftarrow \mathsf{PHF.TrapEval}(\tau, r_i)$ (for the received forgery we would get (a^*, b^*)). Let Bad_6 be the event that, either $a_i \equiv 0 \bmod p$ for some i such that $r^* = r_i$, or $a^* \neq 0$. If Bad_6 occurs, Game 6 aborts. By the $(1, \mathsf{poly}, \gamma', \delta')$ programmability of D

$$\Pr[G_6] \geq \delta' \Pr[G_5]$$

Now we embed the received $(q+1)$-DH challenge $(\hat{g}_1, \hat{g}_1^y, \ldots, \hat{g}_1^{y^{q+1}}, \hat{g}_2, \hat{g}_2^y)$ as input, and we proceed as before (but using the fact that we do not explicitly know y). It is easy to check that all signing queries can be answered. Moreover, once the forgery (M^*, σ^*, r^*) is produced, we can extract a solution of the $(q+1)$-DH challenge as follows. First, since by Game 4 $c_{M^*,0} \neq 0$, we can write

$$z = \left(\frac{\sigma^*}{\hat{g}_1^{c'_{M^*}(\mathsf{d}(r^*)) \prod_{r \in R^*}(a_r + y b_r)}} \right)^{b^*/c_{M^*,0}}$$

where, $c'_{M_*}(\mathsf{d}(r^*))$ is the polynomial obtained from $c_{M^*}(\mathsf{d}(r^*))$ by removing the constant term $c_{M^*,0}$ and dividing by $\mathsf{d}(r^*)$).

$$\begin{aligned}
z &= \left(\frac{\mathsf{H.PriEval}(\mathsf{sek}, M^*)^{\frac{1}{\mathsf{d}(r^*)}}}{\hat{g}_1^{c'_{M^*}(\mathsf{d}(r^*)) \prod_{r \in R^*}(a_r + y b_r)}} \right)^{b^*/c_{M^*,0}} = \left(\frac{g_1^{c_{M^*}(\mathsf{d}(r^*))/\mathsf{d}(r^*)}}{\hat{g}_1^{c'_{M^*}(\mathsf{d}(r^*)) \prod_{r \in R^*}(a_r + y b_r)}} \right)^{b^*/c_{m^*,0}} \\
&= \left(\frac{g_1^{c_{M^*,0}/\mathsf{d}(r^*)} \hat{g}_1^{c'_{M^*}(\mathsf{d}(r^*)) \prod_{r \in R^*}(a_r + y b_r)}}{\hat{g}_1^{c'_{M^*}(\mathsf{d}(r^*)) \prod_{r \in R^*}(a_r + y b_r)}} \right)^{b^*/c_{M^*,0}} = \left(g_1^{c_{M^*,0}/\mathsf{d}(r^*)} \right)^{b^*/c_{M^*,0}} \\
&= \left(g_1^{c_{M^*,0}/(y b^*)} \right)^{b^*/c_{M^*,0}} = g_1^{1/y}
\end{aligned}$$

Finally, by using techniques from [7] one can easily get the desired result $\hat{g}_1^{1/y}$.

Lemma 13 (Type II forgeries). *Let $\mathcal{B}$ be a type II forger that breaks the signature scheme with advantage ϵ_2 (and making up to q signature queries). Then there exists an (equally efficient) adversary $\mathcal{A}$ that breaks the $(q+1)$-DH assumption with advantage ϵ' and an adversary that breaks the discrete logarithm assumption with advantage ϵ'' where*

$$\epsilon' + \delta' \epsilon'' \geq \delta'(\epsilon_2 - \gamma)$$

This lemma can be proved by easily adapting the proof of lemma 4 to this setting.

References

1. 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.
2. 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.
3. N. Attrapadung, B. Libert, and T. Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In K. Kurosawa and G. Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 386–404. Springer, Feb. / Mar. 2013.
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. C. Mitchell, A. Scedrov, and B. Schmidt. Automated analysis of cryptographic assumptions in generic group models. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 95–112. Springer, Aug. 2014.
6. D. Boneh and X. Boyen. Efficient selective-ID secure identity based encryption without random oracles. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 223–238. Springer, May 2004.
7. D. Boneh and X. Boyen. Short signatures without random oracles. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 56–73. Springer, May 2004.
8. D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456. Springer, May 2005.
9. 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.
10. 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.
11. 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. Springer, Mar. 2011.
12. X. Boyen, X. Fan, and E. Shi. Adaptively secure fully homomorphic signatures based on lattices. Cryptology ePrint Archive, Report 2014/916, 2014. `http://eprint.iacr.org/2014/916`.
13. 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.
14. 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.
15. 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.
16. D. Catalano, D. Fiore, and B. Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 371–389. Springer, Aug. 2014.
17. P. Erdös, P. Frankel, and Z. Furedi. Families of finite sets in which no set is covered by the union of r others. *Israeli Journal of Mathematics*, 51:79–89, 1985.
18. 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.
19. E. S. V. Freire, D. Hofheinz, K. G. Paterson, and C. Striecks. Programmable hash functions in the multilinear setting. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 513–530. Springer, Aug. 2013.
20. 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.
21. 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.
22. S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. Cryptology ePrint Archive, Report 2014/897, 2014. `http://eprint.iacr.org/2014/897`.
23. M. Green and S. Hohenberger. Practical adaptive oblivious transfer from simple assumptions. In Y. Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 347–363. Springer, Mar. 2011.

24. G. Hanaoka, T. Matsuda, and J. C. N. Schuldt. On the impossibility of constructing efficient key encapsulation and programmable hash functions in prime order groups. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 812–831. Springer, Aug. 2012.
25. D. Hofheinz, T. Jager, and E. Kiltz. Short signatures from weaker assumptions. In D. H. Lee and X. Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 647–666. Springer, Dec. 2011.
26. D. Hofheinz and E. Kiltz. Programmable hash functions and their applications. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 21–38. Springer, Aug. 2008.
27. D. Hofheinz and E. Kiltz. Programmable hash functions and their applications. *Journal of Cryptology*, 25(3):484–527, July 2012.
28. 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. Springer, Feb. 2002.
29. R. Kumar, S. Rajagopalan, and A. Sahai. Coding constructions for blacklisting problems without computational assumptions. In M. J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 609–623. Springer, Aug. 1999.
30. B. Libert, T. Peters, M. Joye, and M. Yung. Linearly homomorphic structure-preserving signatures and their applications. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 289–307. Springer, Aug. 2013.
31. S. Mitsunari, R. Saka, and M. Kasahara. A new traitor tracing. *IEICE Transactions*, E85-A(2):481–484, Feb. 2002.
32. B. R. Waters. Efficient identity-based encryption without random oracles. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 114–127. Springer, May 2005.

A Digital Signatures

A digital signature scheme consists of three algorithms $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ such that:

$\mathsf{KeyGen}(1^\lambda)$ the key generation takes as input a security parameter λ and returns a secret key sk and a public verification key vk.

$\mathsf{Sign}(\mathsf{sk}, m)$ on input a secret key sk and a message m, the signing algorithm generates a signature σ.

$\mathsf{Ver}(\mathsf{vk}, m, \sigma)$ given a triple vk, m, σ the verification algorithm outputs 1 (accept) if σ is a valid signature on m for verification key vk, and 0 (reject) otherwise.

The security of a signature scheme, called *existential unforgeability against chosen message attacks* ($\mathsf{UF\text{-}CMA}$) is defined via the following experiment:

Experiment $\mathbf{Exp}^{\mathsf{UF\text{-}CMA}}_{\mathcal{A},\Sigma}(\lambda)$
$(\mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{vk})$
If $\mathsf{Ver}(\mathsf{vk}, m^*, \sigma^*) = 1$ and $m^* \neq m_i$ for all m_i queried to $\mathsf{Sign}(\mathsf{sk}, \cdot)$, output 1
Else Output 0

The advantage of $\mathcal{A}$ in breaking the $\mathsf{UF\text{-}CMA}$-security of Σ is $\mathbf{Adv}^{\mathsf{UF\text{-}CMA}}_{\mathcal{A},\Sigma}(\lambda) = \Pr[\mathbf{Exp}^{\mathsf{UF\text{-}CMA}}_{\mathcal{A},\Sigma}(\lambda) = 1]$. Then we say that $\mathcal{A}$ (t, Q, ϵ)-breaks the $\mathsf{UF\text{-}CMA}$-security of Σ if $\mathcal{A}$ runs in time t, makes at most Q signature queries, and $\mathbf{Adv}^{\mathsf{UF\text{-}CMA}}_{\mathcal{A},\Sigma}(\lambda) = \epsilon$.

A digital signature scheme Σ is $\mathsf{UF\text{-}CMA}$-secure if for any PPT $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{UF\text{-}CMA}}_{\mathcal{A},\Sigma}(\lambda)$ is negligible.

B On the Hardness of the FDHI Assumption

To gain confidence in the FDHI assumption we show that FDHI is implied by the following decisional assumption:

Definition 11 (Decisional Assumption 1). *Let $\mathcal{G}$ be a generator of asymmetric bilinear groups, let* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$ *where g_1, g_2 are two random generators. The Decisional Assumption 1 is ϵ-hard for $\mathcal{G}$ if for every PPT adversary $\mathcal{A}$:*

$$|\Pr[\mathcal{A}(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}}, g_2^{1/z})] - \Pr[\mathcal{A}(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}}, g_2^t)]| \leq \epsilon$$

where $z, v, r, t \xleftarrow{\$} \mathbb{Z}_p$.

Proposition 2. *For any $\mathcal{A}$ which ϵ-breaks the FDHI assumption, there is $\mathcal{B}$ which ϵ'-breaks Assumption 1 where $\epsilon' \geq \epsilon - 1/p$*

Proof (Sketch). Let $(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}}, T)$ be the input of $\mathcal{B}$ where T can be either $g_2^{1/z}$ or g_2^t for a random and independent t. $\mathcal{B}$ runs $(W, Y) \leftarrow \mathcal{A}(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}})$. If $e(Y, g_2^z) = e(W, g_2)$ (i.e., $\mathcal{A}$ succeeds), then $\mathcal{B}$ returns 1 if $e(W, T) = e(Y, g_2)$ holds, and 0 otherwise.

Clearly, if $T = g_2^{1/z}$, $e(W, T) = e(W, g_2^{1/z}) = e(W^{1/z}, g_2) = e(Y, g_2)$. Instead, if T is random and independent, the equation holds only with negligible probability $1/p$.

As a next step, we show that Assumption 1 can be equivalently re-written in the following Assumption 2 without rational exponents:

Definition 12 (Decisional Assumption 2). *Let $\mathcal{G}$ be a generator of asymmetric bilinear groups, let* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$*. Let $h_1 \in \mathbb{G}_1, h_2 \in \mathbb{G}_2$ be two random generators. The Decisional Assumption 2 is ϵ-hard for $\mathcal{G}$ if for every PPT adversary $\mathcal{A}$:*

$$|\Pr[\mathcal{A}(h_1, h_2, h_2^x, h_2^u, h_1^u, h_1^{ru}, h_1^{rx}, h_2^{x^2})] - \Pr[\mathcal{A}(h_1, h_2, h_2^x, h_2^u, h_1^u, h_1^{ru}, h_1^{rx}, h_2^t)]| \leq \epsilon$$

where $x, u, r, t \xleftarrow{\$} \mathbb{Z}_p$.

Proof. The equivalence between the assumptions is obtained by setting the following equalities:

$$g_1 = h_1^u, g_2 = h_2^x, g_2^z = h_2, g_2^v = h_2^u, g_1^{z/v} = h_1, g_1^r = h_1^{ru}, g_1^{r/v} = h_1^{rx}, T = T$$

Finally, it is not hard to see that Assumption 2 is hard in the generic bilinear group model. When framing the assumption according to the master theorem in [8], the polynomial x^2 (in the group $\mathbb{G}_2$) is in fact linearly-independent from the other polynomials representing the instance of the assumption. To confirm the validity of Assumption 2, we also automatically tested it using the generic group tool of [5].[8]

[8] The simple script describing the assumption is available upon request.