smklhs: Succinct Multi key Linearly Homomorphic Signatures for Certified Statistics

Diego F. Aranha
Aarhus University
Aarhus, Denmark
dfaranha@eng.au.dk

Cecilia Boschini
ETH
Zurich, Switzerland
cecilia.boschini@inf.ethz.ch

Hanna Ek
Chalmers University of Technology and
University of Gothenburg
Gothenburg, Sweden
hanna.ek@chalmers.se

Elena Pagnin
Chalmers University of Technology and
University of Gothenburg
Gothenburg, Sweden
elenap@chalmers.se

Abstract

We study the problem of certifying statistical claims over datasets contributed by multiple independent sources. In this setting, an untrusted server aggregates signed data records and publishes claims such as sums, averages, or rates, while any third party can verify that these claims are correct with respect to the authenticated input data, without needing access to the underlying records. A central challenge is to achieve public verifiability without requiring trust in the aggregator, while keeping both the proof size and the verification cost small enough for practical deployment. This problem is motivated by applications in which reliable and scalable certification of published statistics is essential, including official health and demographic reporting.

In this work, we present smklhs, a multi-key linearly homomorphic signature scheme for this setting. Compared to the state of the art, smklhs is the first practical construction to enjoy evaluated signatures of size logarithmic in the number of distinct signers involved in the computation, and else independent on the total number of input messages. We prove smklhs secure against fully adaptive adversaries in the random oracle and algebraic group models, under well-studied hardness assumptions in bilinear groups.

We implement our scheme using the high-performance pairing library RELIC and compare it with prior work. To demonstrate practicality, we consider a case study on authenticated mortality statistics related to the impact of COVID-19 in Spain. At the 128-bit security level, our experiments show that an authenticated claim covering a 180-day nationwide dataset with over 300,000 signed records generated by 190 distinct signers can be verified in approximately 22 seconds on a commodity desktop machine. These results indicate that our approach is fast, lightweight, and practical for real-world deployment.

1 Introduction

Modern data processing systems tend to outsource storage and computation to remote servers. In many settings, however, the data is contributed by multiple independent sources, and the server cannot be fully trusted. Users querying the server for aggregate statistics on the stored records may therefore demand publicly verifiable guarantees that the reported results are correct. Our goal is to design a practical solution to such verifiable aggregation with lower communication cost than existing alternatives for computations involving data by several sources.

We address this problem using multi key homomorphic signatures (MKHS) [20]. With MKHS, each source independently signs and uploads its data to a server, which can then answer numerical queries together with a compact evaluated signature certifying the correctness of the result. As a result, any verifier can detect whether the server has injected false data or returned an incorrect aggregation. Figure 1 illustrates this setting. This source authentication layer is particularly relevant to federated data analysis and sensor based monitoring, where aggregate outputs must be certified with respect to data contributed by multiple parties.

At present, only two MKHS constructions admit efficient implementations: *The Simplest* [6], and Schabhüser *et al.*'s [37]. In both cases the size of an evaluated signature grows linearly with the number of signers contributing to the computation. This may be acceptable when only a few data providers are involved, but becomes a limitation in larger statistical reporting (*e.g.,* Spain has 19 territorial units, EU has 27 member states, USA 50, WHO 194).

1.1 Our contribution

Our main contribution is the design and implementation of the first concrete constructions of efficient, **s**uccinct, **m**ulti-**k**ey, **l**inearly **h**omomorphic **s**ignature schemes suitable for certifying the outcome of statistics over data contributed by multiple sources. At a high level, our realizations are a careful combination of an *improved* version of the *The Simplest* multi key linearly homomorphic signature by Aranha and Pagnin (aka mklhs) [6] with a *simplified* version of the succinct, non interactive, inner product argument in Bulletproofs [13]. The main technical challenge we address is designing the (asymptotically) shortest MKHS signatures to date, while avoiding non falsifiable assumptions.

Theoretical contributions. We provide the *first efficient* constructions of multi key linearly homomorphic signature scheme where the size of evaluated signatures grows sublinearly –in fact *logarithmically*– with the number of *signers* (data sources) involved in the evaluation, and is otherwise independent of the total number of inputs to the evaluated function (intuition in subsection 1.2).

We prove security under variants of the discrete logarithm problem (coCDH* and dl-rel), in the algebraic group model (AGM) and

the programmable random oracle (RO) model. We present two constructions: smklhs, secure against *weakly* adaptive adversaries (i.e., some queries need to be declared at the start of the security game); and adpt.smklhs, secure against *fully adaptive* adversaries but 2x slower and with 2x larger signatures compared to smklhs.

Along the way we prove a sequence of auxiliary results of independent interest, including security of the inner product argument in Bulletproofs in the AGM and under the presence of additional group elements, and a detailed proof for the non interactive version of the argument.

Practical contributions. We provide an open-source implementation of the non interactive inner product argument from Bulletproofs [13], smklhs and adpt.smklhs within the RELIC library, together with tests and benchmarks. While minor, this contribution gives a stand-alone building block which may serve as a foundation for future work. We extensively compare our scheme with previous work implemented over the same building blocks, achieving large savings in computation (up to 3.4x) and signature size (8x) in comparison to Schabhüser *et al.* [37]; and a trade-off between modest slowdown (1.25x) with shorter signatures in comparison to [6].

A concrete use case study: certified COVID-19 statistics. We compute authenticated statistics using the COVID-19 data published by the government of Spain as part of the EuroMOMO platform [34]. Figure 1 illustrates the data flow in this setting. In our case study we emulate territorial units of Spain that independently report daily updates of case counts and mortality to a central database, signed under independent private keys. This naturally suggests a multi source setting where each region uploads data signed with a different secret key. This multi key setting serves two main purposes: it allows to cryptographically identify the data sources, and it limits damages due to key exposure or leaks. Finally, we show how a database user, e.g., journalists, researchers, or public agencies, can use that data to compute Standardized Mortality Rate (SMR) and Excess Mortality (EM) metrics in an *authenticated* way by simply evaluating linear functions over the signed data. This ensures transparency and accountability: database users should be able to verify that the published aggregate results are computed correctly and only from authentic records, even when the database is untrusted. The efficiency of our scheme allows to perform this computation for a 180-day period for all 19 autonomous communities of Spain, such that verification of the overall computation involving more than 30,000 individual data records and signatures takes less than 2.2 seconds in a standard Desktop computer. We scale this experiment up to 190 signers and more than 300,000 records to achieve a verification time of 22 seconds.

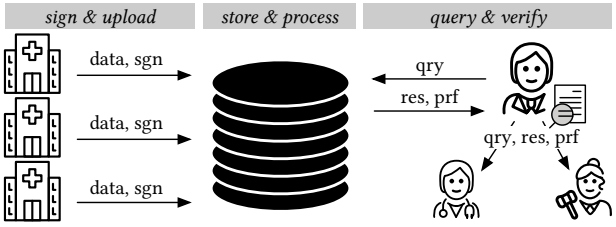

Figure 1: Certified statistics: data sources (signers) upload signed data to a database (untrusted server). The server processes queries from end users (verifiers), who receive responses that include a result and a short proof that certifies the returned result is correctly derived from the signed data stored in the database.

1.2 Technical overview

A generic approach. A natural way to design succinct MKHS is to combine any digital signature scheme with a SNARK (succinct non interactive argument of knowledge). This would entail proving the verification of n signatures (one per input/message) as well as that a given function f on the chosen inputs evaluates to $y = f(x_1, \dots, x_n)$. This approach is flawless in theory; in practice, however, the signature verification algorithm needs to be translated into a circuit, which imposes a considerable computational overhead, even from SNARK-friendly signatures. Moreover, one needs to be careful on instantiating the SNARK to avoid common knowledge of exponent assumptions (which are non falsifiable) and impossibility results in the presence of oracles [21].

Our approach. We follow the same blueprint of combining a MKHS and a succinct proof system, but do so in a tailored and constrained way. First, we focus on a MKHS construction that is linearly homomorphic, i.e., it supports evaluations of functions of the form $f(x_1, \dots, x_n) = \sum_{i=1}^{n} a_i x_i$. This means that one homomorphic evaluation can be split into the sum of t evaluations, one per different signer contributing data to the function, e.g., if $x_1, \dots x_A$ are inputs signed by Alice and $x_{A+1}, \dots, x_B$ are input signed by Bob then $f(x_1, \dots x_B) = \sum_{i=1}^{A} f_i x_i + \sum_{i=A+1}^{B} f_i x_i$ (by commutativity and associativity of addition). Second, we isolate the core building block of a succinct proof system, namely an inner product argument, to prove in a succinct way that the t addends $(a_1, a_2) = (\sum_{i=1}^{A} f_i x_i, \sum_{i=A+1}^{B} f_i x_i)$ sum to $y = \langle (a_1, a_2), (1, 1) \rangle = \sum_{i=1}^{2} a_i$.

In more detail, our starting point is the most efficient multi key linearly homomorphic signature to date: *The Simplest* (mklhs) [6]. *The Simplest* is a natural adaptation of Boneh et al.'s linearly homomorphic signature scheme from [8] to the multi key setting. A mklhs signature is of the form

$$\sigma = (\mu, \Gamma), \quad \Gamma = (\mathcal{H}(\ell) \cdot g^{\mu})^{\mathsf{sk}}$$

where μ is the message (a $\mathbb{Z}_p$ value), ℓ is a label clarifying the "context" of the data input μ (this is necessary in MKHS), and $\mathcal{H}$ is a hash-to-point function (modeled as a random oracle). Homomorphic evaluation of linear functions happens naturally in the exponent: given two signatures σ_1, σ_2 on messages μ_1, μ_2 generated by two different signers with keys $\mathsf{sk}_1, \mathsf{sk}_2$, the aggregated signature on $\mu = f(\mu_1, \mu_2) = f_1\mu_1 + f_2\mu_2$ (i.e., μ is a linear function f of μ_1, μ_2) has to include both the aggregated signature $\Gamma_1^{f_1} \cdot \Gamma_2^{f_2}$ $(= \mathcal{H}(\ell_1)^{f_1 \mathsf{sk}_1} \cdot \mathcal{H}(\ell_2)^{f_2 \mathsf{sk}_2} \cdot g^{f_1\mu_1 \mathsf{sk}_1 + f_2\mu_2 \mathsf{sk}_2})$ and the partial values $\mu' = f_1\mu_1$ and $\mu'' = f_2\mu_2$ to be verifiable. This makes the resulting aggregated signature *linear in the number of signers*.

As the name suggests, mklhs is simple (and efficient) but comes with a major security drawback: the security proof restricts adversarial access to the random oracle to "only after the sign-query phase is over". This setting is clearly unrealistic.

Table 1: Comparison of our constructions with prior MKHS schemes, ordered by publication year. λ is the security parameter, n the number of messages, t the number of contributing signers, and d is the depth of the arithmetic circuit. Note that $t \leq n$ and usually $t << n$.

CONSTRUCTION	SIZE OF EVALUATED SIGNATURES	IMPLEMENTATION	HOMOMORPHISM	ASSUMPTIONS	ADAPTIVE SECURITY
MK-HAuth [20]	$O(d + t \cdot (\lambda + 2^{\omega(\log \lambda)d}))$	No	Arithmetic circuits	SIS & exponential module	Weak
Lai *et al.* [32]	$O(\text{poly}(\lambda))$	No	Functions	OWF & O-SNARK	Yes
Matrioska [22]	$O(t \cdot \text{poly}(\lambda))$	No	Arbitrary circuits	unforgeability of HS	Yes
Schabhüser *et al.* [37]	$O(t \cdot \lambda)$	Yes (in [6])	Linear	DDH & dl	Yes
The Simplest [6]	$O(t \cdot \lambda)$	Yes	Linear	RO & coCDH	No
Samarin *et al.* [18]	$O(t \cdot \text{poly}(\lambda))$	No	Turing machines	unforgeability of HS	Yes
Anthoine *et al.* [3, Corollary 2]	$\text{poly}(\log(n) \cdot \lambda) \cdot d$	No	Boolean circuits	DDH or LWE	Yes
Anthoine *et al.* [3, Corollary 3]	$O(\lambda \cdot d^2) + \text{poly}(\lambda)$	No	Arithmetic circuits	HiKer & k-Lin	Yes
smklhs [this work]	$O(\log(t) \cdot \lambda)$	Yes	Linear	RO, AGM, coCDH* & dl	Weak
adpt.smklhs [this work]	$O(\log(t) \cdot \lambda)$	Yes	Linear	RO, AGM, coCDH* & dl	Yes

So we set out to address the aforementioned limitations of *The Simplest* while preserving its lean structure. First, we improve efficiency by migrating the construction from Type-2 to Type-3 following [16], plus some minor modifications to preserve malleability without undermining unforgeability. The resulting scheme is not yet succinct because evaluated signatures still carry one component per contributing signer, as in [6]. To circumvent that, we use a simplified version of the succinct, non interactive, inner product argument in Bulletproofs [13]: now the aggregator can just include a succinct proof of knowledge of the partial messages; this yields signatures of logarithmic size in the number of both signers (and else independent on the total number of messages). This trick, however, forces us to impose algebraic adversaries (AGM model) in order to extract one-signer forgeries from a forged multi-signer aggregated signature in an efficient way. Indeed, consider the setting from the example above where the adversary returns a forged Γ^* for the message $\mu^* \neq \mu = f(\mu_1, \mu_2)$. After compacting Γ^* with the inner product argument we have a proof π instead of the addends $f_1\mu_1^*$ and $f_2\mu_2^*$. We use the AGM in order to efficiently identify which message (hence signature) was tampered with.

Avoiding non falsifiable assumptions. AGM proofs typically reduce to non falsifiable assumptions: they can be written as demanding the existence of an extractor that outputs the discrete logs of the adversary's proof so long as the proof satisfies certain pairing relations (i.e., KoE style assumptions). However, Agrikola et al. [2] showed that the AGM can be instantiated from falsifiable assumptions if the scheme allows an algebraic wrapper which includes an encryption of the representation of each element output by the algorithms. Our proofs translate to this setting, but our simulation incurs a tightness loss linear in the number of signers, RO queries and signing queries (due to the lack of statistical indistinguishability when simulating the signatures and the output of the RO).

Limitations. Our schemes support *linear functions* evaluations, and security reduces to group-based hardness assumptions. In particular, our constructions do not handle more expressive computations such as higher degree polynomials or circuits and are *not post quantum secure*. That said, these limitations are less restrictive in practice than it may appear. Our case study already covers meaningful statistics such as Excess Mortality (EM) and Standardized Mortality Rate (SMR), as well as weighted sums arising in federated learning and sensor monitoring. In addition, linearly homomorphic signatures have been used to prove correct shuffling for scalable mix-nets [29], and construct the first constant-size non-malleable commitment to group elements [33]. Regarding post quantum security, pairing-based constructions currently provide a substantially better tradeoff in terms of compactness and efficiency, and our envisioned application scenarios require near- to medium-term authenticity of published aggregates, rather than long-term confidentiality. Moreover, the only MKHS schemes with practical implementations to date are linearly homomorphic and based on bilinear groups [6, 37]. All existing MKHS that support more expressive proposals are purely theoretical results since they rely either on incredibly expensive underlying HS schemes [18, 22]; or on building blocks for which no concrete instantiation exists [3]; or on lattices of exponential-size module [20]; or on non falsifiable assumptions [32] (cf. Table 1 for overview).

Finally, a few MKHS schemes [3, 32, 37] achieve *context-hiding*, namely that evaluated signatures leak no extra information about the signed messages, their positions, or other hidden structure, beyond the output. Our constructions do not provide this property. This choice keeps the design lean and signature size short(er), and aligns with our target applications where the underlying data is public or broadly accessible, as in platforms such as EuroMOMO. In such settings, the main goal is to certify the correctness of published statistics, rather than to hide the input data.

1.3 Related work

Johnson *et al.* [31] conceived the idea of homomorphic signatures (HS), mainly as a mechanism that allows for authenticated redactions of signed messages. Subsequent work expanded on the set of operations allowed on signed data, the most relevant for us are *linearly* HS [8, 10, 14, 36], HS for *polynomial* functions [9, 15], *fully homomorphic* HS from exponential modulo lattices (levelled) [28], or from subexponentially-secure indistinguishability obfuscation and more (unlevelled) [25], and HS for *non deterministic computations* [23]. Agrawal *et al.* [1] extended HS to the multi-source

setting, opening the door to collaborative applications. This line of work led to the first *multi key* leveled fully homomorphic signature scheme [20], a nontrivial extension of [28]. Subsequent work has followed two directions: generic compilers (which remain impractical), and concrete constructions. Table 1 provides a comparison among all existing MKHS proposals.

Generic compilers construct MKHS from other cryptographic primitives in a black-box way. Matrioska [22] and Samarin *et al.* [35] show how to achieve multi key homomorphism information theoretically from very expressive (hence unrealistic) single-key HS. Schabhüser *et al.* [37] realize perfectly context-hiding MKHS from any signature scheme and bilinear groups. Still the size of evaluated signatures is linear in t, the number of signers contributing data to the computation, and the constructions are not context-hiding. Anthoine *et al.* [3] beat these limitations and propose constructions of context-hiding succinct MKHS for circuits of unbounded depth from standard signatures, batch arguments for NP (BARGs), and functional commitments (FC). The size of evaluated signatures in the first construction [3, Corollary 2 in the full version] is logarithmic in n, the *total number of data input* to the evaluated boolean circuit; while in the second construction [3, Corollary 3 in the full version] is independent of both n and t. At present, however, these result remain theoretical, since they requires encoding several non-trivial components inside the BARG circuit, including signature verification, the FC update procedure, and consistency checks with a somewhere-extractable vector commitment. Lai *et al.* [32] also propose a context-hiding MKHS scheme with evaluated signatures of size independent of both n and t. Succinctness and privacy are trivially achieved by employing succinct non interactive argument of knowledge (SNARK); the security proof, however, requires non-standard extra features including strong and oracle SNARKs (O-SNARK) [21] as well non falsifiable assumptions (as proven in [32]), which cannot be efficiently tested or refuted.

Concrete constructions to date are only linearly homomorphic and based on bilinear groups. The first is Schabhüser *et al.*'s [37] compiler as implemented in [6]. The second is *the Simplest* [6] which is the most efficient multi key homomorphic signature to date. We remark that [6] sacrifices message privacy layer and a realistic attacker model in favor of efficiency, and a lean design.

2 Preliminaries

Notation. We denote the security parameter by λ. We use $a \leftarrow n$ to assign the value n to the variable a. Similarly, $a \xleftarrow{\$} \mathbb{Z}_p$ indicates that a is sampled uniformly at random from $\mathbb{Z}_q$. We write == for checks of equality and := to state defined equal.

Let $n \geq 1$ be a positive integer. We denote by $[n]$ the discrete interval set $\{1, 2, \dots, n\}$. When n is clear from the context, we refer to the left (resp. right) part of $[n]$ as $[:i] := \{1, \dots, i\}$ (resp. $[i:] := \{i+1, \dots, n\}$). Vectors are represented using bold font, for example $\mathbf{g} := (g_1, \dots, g_n)$. For multiplication of vectors we use the following notation: let $\mathbf{g}, \mathbf{h}, \mathbf{a}$ be vectors of the same length $n \geq 1$, then $\mathbf{g}^{\mathbf{a}} := \prod_{i=1}^{n} g_i^{a_i}$ and $\mathbf{gh} := \prod_{i=1}^{n} g_i h_i$. We also use the following operations: $\mathbf{a} \circ \mathbf{b} = (a_1 b_1, \dots, a_n b_n)$ (Hadamard product) and $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_i a_i b_i$ (inner product).

Groups are denoted as $\mathbb{G}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, and are of large prime order p, cyclic, and multiplicative. All group elements that are sampled at random, given as a challenge or some other public input to the protocol are denoted by lower case letters, while group elements that have been computed from these group elements are denoted by upper case letters. With a slight abuse of notation, when x is a group element, $[x]$ denotes its algebraic representation.

Negligible functions (in the security parameter) are denoted as $\varepsilon(\lambda)$. All algorithms are assumed to be PPT, i.e., probabilistic and to run in polynomial time in the security parameter, λ. The adversary is a non-uniform PPT algorithm denoted as $\mathscr{A}$.

2.1 Computational assumptions

The discrete logarithm assumption. The advantage of an adversary in solving the discrete logarithm problem $\mathrm{G}^{\mathsf{dl}}_{\mathbb{G}_1}$ is negligible, i.e., $\mathrm{Adv}^{\mathsf{dl}}_{\mathbb{G}_1}(\mathscr{A}, \lambda) = \Pr\left[\mathrm{G}^{\mathsf{dl}}_{\mathbb{G}_1}(\mathscr{A}, \lambda) == 1\right] \leq \varepsilon(\lambda)$.

The non-trivial discrete relation assumption. The advantage of an adversary in $\mathrm{G}^{\mathsf{coCDH}^*}_{\mathbb{G}_1,\mathbb{G}_2}$ is negligible, i.e., $\mathrm{Adv}^{\mathsf{dl\text{-}rel}}_{\mathbb{G}}(\mathscr{A}, \lambda) = \Pr\left[\mathrm{G}^{\mathsf{dl\text{-}rel}}_{\mathbb{G},n}(\mathscr{A}, \lambda) == 1\right] \leq \varepsilon(\lambda)$.

The co-computational Diffie-Hellman assumption. The advantage of an adversary in $\mathrm{G}^{\mathsf{coCDH}^*}_{\mathbb{G}_1,\mathbb{G}_2}$ is negligible, i.e., $\mathrm{Adv}^{\mathsf{coCDH}^*}_{\mathbb{G}_1,\mathbb{G}_2}(\mathscr{A}, \lambda) = \Pr\left[\mathrm{G}^{\mathsf{coCDH}^*}_{\mathbb{G}_1,\mathbb{G}_2}(\mathscr{A}, \lambda) == 1\right] \leq \varepsilon(\lambda)$.

Figure 2 collects the description of the discrete logarithm (dl), the non-trivial discrete relation(dl-rel) and the co-computational Diffie-Hellman (coCDH*) problem. We recall the computational assumptions for each problem, which essentially state that the probability the game returns 1 is negligible in λ (if p is of size $2^{2\lambda}$).

$\mathrm{G}^{\mathsf{dl}}_{\mathbb{G}}(\mathscr{A}, \lambda)$	$\mathrm{G}^{\mathsf{dl\text{-}rel}}_{\mathbb{G},n}(\mathscr{A}, \lambda)$	$\mathrm{G}^{\mathsf{coCDH}^*}_{\mathbb{G}_1,\mathbb{G}_2}(\mathscr{A}, \lambda)$
$g \xleftarrow{\$} \mathbb{G}^*; h \xleftarrow{\$} \mathbb{G}$	$\mathbf{g} := (g_1, \dots, g_n) \xleftarrow{\$} \mathbb{G}$	$g_1, v \xleftarrow{\$} \mathbb{G}_1; g_2 \xleftarrow{\$} \mathbb{G}_2$
$a \xleftarrow{\$} \mathscr{A}(g, h)$	$(a_1, \dots, a_n) \xleftarrow{\$} \mathscr{A}_\lambda(\mathbf{g})$	$\mathrm{x} \xleftarrow{\$} \mathbb{Z}_p$
return $(g^a == h)$	**if** $(a_1, \dots, a_n) \notin (\mathbb{Z}_p^*)^n$	$h_1 := g_1^{\mathrm{x}}, h_2 := g_2^{\mathrm{x}}$
	return 0	$y \leftarrow \mathscr{A}(\{g_i, h_i\}_{i=1}^2, v)$
	return $\left(\prod g_i^{a_i} == 1\right)$	**return** $(y == v^{\mathrm{x}})$

Figure 2: Computational problems over finite groups.

2.2 The algebraic group model

An *algebraic* algorithm $\mathcal{A}_{\mathsf{alg}}$ is an algorithm that, whenever it outputs a group element $Z \in \mathbb{G}$, it also outputs its *representation*, i.e., a vector of exponents $\mathbf{z} \leftarrow (z_1, \dots, z_t) \in \mathbb{Z}_p^t$ such that $Z = \prod_{i=1}^{t} K_i^{z_i}$, where $\mathbf{K} \leftarrow (K_1, \dots, K_t)$ is the list of all group elements given to $\mathcal{A}_{\mathsf{alg}}$ during its execution. For simplicity, we denote the representation of an element as $[Z] \leftarrow (Z, z_1, \dots, z_t)$.

2.3 Bilinear pairings

A bilinear pairing is a non-degenerate efficiently computable map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ defined over groups of prime order p. The core property of the map e is linearity in both arguments, i.e., $e(g^x, h) =$

$e(g,h)^x = e(g,h^x)$. For efficiency, we consider only pairings of Type-3 over pairing friendly curves. Security typically relies directly or indirectly on the hardness of solving the Bilinear Diffie-Hellman problem, which asks to compute $e(g,h)^{abc}$ given $g, g^a, g^b, g^c, h, h^a, h^b, h^c$ for $g \in \mathbb{G}_1$, $h \in \mathbb{G}_2$ and $a, b, c \in \mathbb{Z}_p^*$.

2.4 Inner product arguments

An inner product argument [12] is an interactive protocol between a Prover and a Verifier, at the end of which the Prover produces a logarithmic sized transcript that should cryptographically convince the Verifier that $(\mathbf{g}, \mathbf{h}, u, P, c)$ satisfies the following relation:

$$\mathsf{R}^{\mathsf{InPrd}} = \{ (\mathbf{g}, \mathbf{h} \in \mathbb{G}^n,\ P \in \mathbb{G},\ c \in \mathbb{Z}_p\ ; \mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n) : P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}} \wedge \langle \mathbf{a}, \mathbf{b} \rangle = c \} \quad (1)$$

Protocol 1 in [13], converts a proof system for the relation in (1), to a proof system for the relation in (2) with the same complexity,

$$\{(\mathbf{g}, \mathbf{h} \in \mathbb{G}^n,\ P, u \in \mathbb{G}\ ; \mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n) : P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}} u^{\langle \mathbf{a}, \mathbf{b} \rangle}\} \quad (2)$$

Throughout this paper we will state the relations we wish to prove in the form of (1), than transform them into (2) following Protocol 1 in [13] and give all protocols this relation.

A non interactive version of the inner product argument in [12] can be obtained via the Fiat-Shamir heuristic [7]. The Prover directly outputs a proof, π, that convinces any verifier that a commitment P satisfies the relation $\mathsf{R}^{\mathsf{InPrd}}$. The non interactive inner product argument considered is denoted as InPrd, and its full description is deferred to Appendix A.

2.5 Multi key homomorphic signatures

Multi key homomorphic signatures [20] are a natural generalization of homomorphic signatures. To be meaningful, the homomorphic authentication of messages needs to happen in a "context", i.e., a signer not only signs a value, say "10", but she also signs that "10" is the value assigned to a certain entry in the database. *Labels* are used to make the connection between messages (values) and context. Therefore, on top of the input values and signatures, one also needs the respective labels and the function to evaluate. Together they compose the labeled program.

Definition 2.1 (Labeled Program [26]). A labeled program $\mathcal{P}$ is a tuple of the form $(f, \ell_1, \ldots, \ell_n)$ such that, $f : \mathcal{M}^n \to \mathcal{M}$ is a function that takes n inputs and returns a single value, while $\ell_1, \ldots, \ell_n \in \{0,1\}^*$ are the n labels identifying each input.

When moving to the multi key setting, the mainstream approach is to include the signer's identity in the label [6, 20]. This produces labels ℓ of the form $\ell = (\mathsf{id}, \tau)$, where id is the user's identifier and τ is a tag uniquely identifying the data item (for user id). We refer to [20] for further details on labeled programs and their composition. The final ingredient to define the context is a database identifier, typically denoted as $\Delta \in \{0,1\}^*$, see [20].

*Definition 2.2 (Multi key Homomorphic Signatures (*MKHS*) [3, 20]).* A multi key homomorphic signature scheme is a tuple of five PPT algorithms (SetUp, KeyGen, Sign, Eval, Verify) such that:

$\mathsf{SetUp}(\lambda) \to \mathsf{pp}$ Takes as input the security parameter and outputs public parameters containing (at least) the description of the identity space ID, the tag space $\mathcal{T}$, the message space $\mathcal{M}$ and the set of admissible functions $\mathcal{F}$. The label space is $\mathcal{L} := \mathsf{ID} \times \mathcal{T}$. pp is an implicit input to all other algorithms.

$\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{id}, \mathsf{sk}, \mathsf{pk})$ Outputs a unique identity id $\in$ ID, and a secret-public key pair (sk, pk).

$\mathsf{Sign}(\mathsf{sk}, \Delta, \ell, \mu) \to \sigma$ On input a signing-key sk, a dataset identifier Δ, a label $\ell = (\mathsf{id}, \tau) \in \mathcal{L}$, and a message $\mu \in \mathcal{M}$, it outputs a signature σ.

$\mathsf{Eval}(\mathcal{P}, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, \{\mu_i\}_{i\in[n]}, \{\sigma_i\}_{i\in[n]}) \to \hat{\sigma}$ Takes as input a labeled program $\mathcal{P}$, a set of public keys $\{\mathsf{pk}\}_{\mathsf{id}}$, a set of messages, a set of signatures, and outputs a signature $\hat{\sigma}$.

$\mathsf{Verify}(\mathcal{P}, \Delta, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, \mu, \hat{\sigma}) \to 0/1$ Takes as input a labeled program $\mathcal{P}$, a dataset identifier Δ, set of public keys $\{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}$, a message μ, and a signature $\hat{\sigma}$. Outputs $0/1$.

When referring to a MKHS scheme, we implicitly assume it satisfies Definition 2.2 as well as *Authentication Correctness, Evaluation Correctness, Succinctness* and *Security* recalled below (see [20] for details).

Authentication correctness. This property states that every signature output by Sign–for a tuple $(\mathsf{sk}_{\mathsf{id}}, \Delta, \ell, \mu)$– is accepted by the verification algorithm against the tuple $(\ell, \Delta, \mathsf{pk}_{\mathsf{id}}, \mu)$ for the identity function f.

Evaluation correctness. This property states that if the signatures input to Eval satisfy authentication correctness, then Eval outputs signatures that verify for the appropriate message $\hat{\mu} = f(\mu_1, \ldots, \mu_n)$ and labeled program $\mathcal{P} := (f, \ell_1, \ldots, \ell_n)$. In this work we are interested in single-hop signatures, that is, the output of Eval is not a valid input to Eval (and cannot be further computed on).

Succinctness. This property states that the size of the signature output by Eval for a labeled program $\mathcal{P} := (f, \ell_1, \ldots, \ell_n)$ should depend at most logarithmically on n (the number of messages input to f) and t (the number of distinct signers involved in the computation). More formally, a multi key homomorphic signature $\hat{\sigma} \leftarrow \mathsf{Eval}(\mathcal{P}, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, \{\mu_i\}_{i\in[n]}, \{\sigma_i\}_{i\in[n]})$ is succinct if $|\hat{\sigma}| = \mathcal{O}(\log(\lambda, t, n))$.

Security. Following [20], we consider homomorphic unforgeability under chosen message attack with non-adaptive corruptions. This means that $\mathcal{A}$ can only corrupt an identity id if id had not been already queried to the signing or the verification oracles.

We consider two types of adversaries: adversaries that perform only *weakly-adaptive* queries, and adversaries that are *fully-adaptive*. The difference lies in that weakly-adaptive security (wHUFCMA) requires $\mathcal{A}$ to declare all queries to the signing oracle before performing any identity (new key generation) or signing query, yet during the game $\mathcal{A}$ can adaptively choose in which order to perform the declared queries. On the other hand, fully-adaptive security (HUFCMA) lets $\mathcal{A}$ chose the input to the signing query i after having received responses for all signing queries $j < i$. In both cases, we only allow corruptions upon a fresh identity query, as done, e.g., in [6, 20, 37]. While weakly-adaptive adversaries may seem artificially restricted, we remark that adaptive security is often considerably more expensive, and the overhead comes mainly from technical challenges in the security proof which not always reflect real-world attacks against the proven weakly-adaptive scheme.

Since the difference between weakly- and fully-adaptive adversaries are minor, we present both games in a single definition.

Definition 2.3 (wHUFCMA). Let MKHS be a multi key homomorphic signature scheme The security game for homomorphic unforgeability under chosen message attack (wHUFCMA) with weakly / fully-adaptive queries is as follows. Steps applicable solely for weakly-adaptive security have a gray background.

Weakly-Adaptive Adversaries: A the beginning of the game $\mathscr{A}$ declares the set of dataset, label and message triplets to be queried to the signing oracle. Specifically, $\mathscr{A}$ sends a collection $L_{(\Delta,\ell,\mu)} = \{(\Delta,\ell,\mu)_i\}_{i\in[Q]}$, where $[Q]$ is the total number of signing queries to be performed during the game.

Setup: The challenger $\mathscr{C}$ runs $\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda)$ and returns pp to the adversary $\mathscr{A}$. In addition, $\mathscr{C}$ initializes three empty lists: L_σ to track all signatures, L_{ID} to track all generated identities, and L_{corr} to track all corrupted identities. Upon receiving pp, $\mathscr{A}$ gets access to any random oracle in pp.

Identity Queries: The adversary can submit queries on the form $\mathsf{id} \in \mathsf{ID}$. If $\mathsf{id} \notin L_{ID}$, the challenger creates a new honest user id: first, it generates a new key pair by running $(\mathsf{sk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, then adds the identity together with the key pair to the list of generated identities, $L_{ID} \leftarrow L_{ID} \cup \{(\mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}})\}$ and returns $\mathsf{pk}_{\mathsf{id}}$ to $\mathscr{A}$.
If $\mathsf{id} \in L_{ID}$, $\mathscr{C}$ aborts if there is an entry $(\cdot,(\mathsf{id},\cdot),\cdot) \in L_\sigma$. Otherwise, it interprets the query as a corruption query: it updates the list of corrupted identities $L_{\mathsf{corr}} \leftarrow L_{\mathsf{corr}} \cup \{\mathsf{id}\}$, it retrieves the triple $(\mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}})$ from L_{ID} and finally returns $\mathsf{sk}_{\mathsf{id}}$ to $\mathscr{A}$.

Signing Queries: $\mathscr{A}$ adaptively submits queries of the form $(\Delta, \mathsf{id}, \mu)$, where Δ is a dataset identifier, $\ell = (\mathsf{id}, \tau)$ is a label in $\mathsf{ID} \times \mathcal{T}$ and $\mu \in \mathcal{M}$ is a message. If $\{\cdot,\ell,\cdot,\cdot\} \in L_\sigma$ $/(\Delta, \mathsf{id}, \mu) \notin L_{(\Delta,\ell,\mu)}$, the challenger ignores the query.

If $(\mathsf{id},\cdot,\cdot) \notin L_{ID}$, i.e. the identity specified by the label has not been initialized, $\mathscr{C}$ interprets it as an identity query: first it generates a new key pair by running $(\mathsf{sk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, then adds the identity together with the key pair to the list of generated identities, $L_{ID} \leftarrow L_{ID} \cup \{(\mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}})\}$. Once the keys for id have been initialized (or recovered from L_{ID}) $\mathscr{C}$ computes the signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{id}}, \Delta, \ell = (\tau, \mathsf{id}), \mu)$ and stores $(\Delta, \ell = (\mathsf{id}, \tau), \mu, \sigma) \in L_\sigma$ and returns σ to the adversary.

End of Game: At the end of the game, $\mathscr{A}$ outputs

$$(\mathcal{P}^* := (f, \{\ell_i\}_{i\in[n]}), \Delta^*, \{\mathsf{pk}^*_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, \hat{\mu}^*, \hat{\sigma}^*).$$

If the output tuple is a valid forgery (see Definition 2.4), the adversary wins, and the game returns 1. Else, it returns 0.

Definition 2.4 (Forgeries [20]). Let $(\mathcal{P}^*, \Delta^*, \{\mathsf{pk}^*_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, \hat{\mu}^*, \hat{\sigma}^*)$ be the tuple returned by $\mathscr{A}$ at the end of the wHUFCMA security game (Definition 2.3). Let $\mathcal{P}^* := (f, \{\ell_i\}_{i\in[n]})$. We say that $\mathscr{A}$ outputs a valid forgery if

$$\mathsf{Verify}(\mathcal{P}^*, \Delta^*, \{\mathsf{pk}^*_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, \hat{\mu}^*, \hat{\sigma}^*) == 1$$

and (at least) one of the following conditions holds:

Type-1 forgery The dataset was not initialized, i.e. $(\Delta^*,\cdot,\cdot) \notin L_\sigma$.

$\widetilde{\mathsf{InPrd}}.\mathsf{P}\big(\mathsf{pp} = (n, \mathbf{g}, u), P; \mathbf{a}, \mathbf{b} := \mathbf{1}^n\big) \rightarrow \tilde{\pi}$

$\mathbf{g}^{(0)} \leftarrow \mathbf{g},\ \mathbf{a}^{(0)} \leftarrow \mathbf{a},\ \mathbf{b}^{(0)} \leftarrow \mathbf{b},\ m \leftarrow n$
for $i = 1, \ldots, \log(n)$
  $m \leftarrow m/2$
  $c_L \leftarrow \langle \mathbf{a}^{(i-1)}_{[:m]}, \mathbf{b}^{(i-1)}_{[m:]} \rangle, \quad c_R \leftarrow \langle \mathbf{a}^{(i-1)}_{[m:]}, \mathbf{b}^{(i-1)}_{[:m]} \rangle$
  $L_i \leftarrow (\mathbf{g}^{(i-1)}_{[m:]})^{\mathbf{a}^{(i-1)}_{[:m]}} u^{c_L}, \quad R_i \leftarrow (\mathbf{g}^{(i-1)}_{[:m]})^{\mathbf{a}^{(i-1)}_{[m:]}} u^{c_R}$
  $x_i \leftarrow \mathcal{H}_2(\mathsf{pp}, P^{(0)}, \{L_j, R_j\}^i_{j=1}, \{x_j\}^{i-1}_{j=1})$
  $\mathbf{g}^{(i)} \leftarrow (\mathbf{g}^{(i-1)}_{[:m]})^{x_i^{-1}} \circ (\mathbf{g}^{(i-1)}_{[m:]})^{x_i}$
  $\mathbf{a}^{(i)} \leftarrow \mathbf{a}^{(i-1)}_{[:m]} \cdot x_i + \mathbf{a}^{(i-1)}_{[m:]} \cdot x_i^{-1}, \quad \mathbf{b}^{(i)} \leftarrow \mathbf{b}^{(i-1)}_{[:m]} \cdot x_i^{-1} + \mathbf{b}^{(i-1)}_{[m:]} \cdot x_i$
$a \leftarrow \mathbf{a}^{(\log n)}$
$\tilde{\pi} \leftarrow \big((L_1, R_1), \ldots, (L_{\log(n)}, R_{\log(n)}), a\big)$

$\widetilde{\mathsf{InPrd}}.\mathsf{V}\big(\mathsf{pp} = (n, \mathbf{g}, u), P, \tilde{\pi}\big) \rightarrow ver$

parse $\tilde{\pi}$, $\mathbf{g}^{(0)} \leftarrow \mathbf{g},\ \mathbf{b}^{(0)} \leftarrow \mathbf{1}^n,\ m \leftarrow n,\ P^{(0)} \leftarrow P$
for $i = 1, \ldots, \log(n)$
  $m \leftarrow m/2$
  $x_i \leftarrow \mathcal{H}_2(\mathsf{pp}, P^{(0)}, \{L_j, R_j\}^i_{j=1}, \{x_j\}^{i-1}_{j=1})$
  $P^{(i)} \leftarrow L_i^{x_i^2} \cdot P^{(i-1)} \cdot R_i^{x_i^{-2}}$
  $\mathbf{g}^{(i)} \leftarrow (\mathbf{g}^{(i-1)}_{[:m]})^{x_i^{-1}} \circ (\mathbf{g}^{(i-1)}_{[m:]})^{x_i}, \quad \mathbf{b}^{(i)} \leftarrow \mathbf{b}^{(i-1)}_{[:m]} \cdot x_i^{-1} + \mathbf{b}^{(i-1)}_{[m:]} \cdot x_i$
$g \leftarrow \mathbf{g}^{(\log(n))}, b \leftarrow \mathbf{b}^{(\log(n))}$
$ver \leftarrow (P^{(\log n)} == g^a u^{ab})$

Figure 3: $\widetilde{\mathsf{InPrd}}$: a simple non interactive inner product argument.

Type-2 forgery The dataset and all labels in $\mathcal{P}^*$ have been initialized during the game, i.e. $(\Delta^*, \ell_i^*, \mu_i) \in L_\sigma$ for all $i \in [n]$, but $\hat{\mu} \neq f^*(\mu_1, \ldots, \mu_n)$.

Type-3 forgery There exists (at least) one index $i \in [n]$ such that ℓ_i^* was never queried for signing, i.e. $(\Delta^*, \ell_i^*, \cdot) \notin L_\sigma$.

Definition 2.5 (Unforgeability). A mkhs scheme satisfies wHUFCMA with non-adaptive corruptions if the advantage of any $\mathscr{A}$ in winning the security game in Definition 2.3 is negligible, i.e.,

$$\mathsf{Adv}^{\mathrm{wHUFCMA}}_{\mathrm{MKHS}}(\mathscr{A}, \lambda) := \Pr[\mathrm{wHUFCMA}(\lambda) == 1] \leq \varepsilon(\lambda).$$

3 Our simplified inner product argument

In this section we present $\widetilde{\mathsf{InPrd}}$, a special instance of the non interactive inner product argument of Bulletproofs [13] where the vector $\mathbf{b}$ (using the notation in [13]) is publicly known and set to $\mathbf{1}^n$ (the vector of all 1s). $\widetilde{\mathsf{InPrd}}$ is the building block that renders our signatures succinct. Pseudocode description is given in Figure 3. $\widetilde{\mathsf{InPrd}}$ returns a proof, $\tilde{\pi}$, that the commitment P satisfies the following relation:

$$\mathsf{R}^{\widetilde{\mathsf{InPrd}}} = \big\{ (\mathbf{g} \in \mathbb{G}_1^n,\ P \in \mathbb{G}_1, c \in \mathbb{Z}_p\ ; \mathbf{a} \in \mathbb{Z}_p^n) : P = \mathbf{g}^{\mathbf{a}} \wedge c = \langle \mathbf{a}, \mathbf{1} \rangle \big\}. \tag{3}$$

We prove the security of $\widetilde{\mathsf{InPrd}}$ through the following steps. First, we present the non-simplified but *non interactive* version of the inner product argument of Bulletproofs obtained by the Fiat-Shamir transform (see InPrd in Appendix A); and prove it secure in the AGM. The proof is heavily inspired by the Bulletproof proof in the AGM given in [27], with the obvious differences that InPrd is non interactive, and we only consider the inner product argument subroutine instead of the full Bulletproof. Then, we reduce the security of $\widetilde{\mathsf{InPrd}}$ to that of InPrd (Appendix B).

Finally, in Appendix C we give a parallel version of the inner product argument, $\widetilde{\mathsf{ParInPrd}}$, needed for our adaptively secure signature.

4 Our weakly secure construction

Figure 4 presents a pseudocode description of smklhs, our succinct multi key linearly homomorphic signature scheme secure against weakly-adaptive adversaries. For readability, n and t denote the number of labels (hence messages) and of unique signers (hence identities) in a labeled program. In addition, I_{id} identifies the subset of $[n]$ containing only the indices of messages queried for the identity id, more formally: $\mathsf{I}_{\mathsf{id}} := \{i \in [n] \ : \ \ell_i \in \mathcal{P} \ \wedge \ \ell_i = (\mathsf{id}, \cdot)\}$.

4.1 Intuition of our construction

We combine a Type-3 version of *the Simplest* [6] with simplified version of Bulletproof inner product argument [13], to achieve succinct evaluated signatures.

In detail, **keygen** outputs public keys that consist of three components $\mathsf{pk} = (\mathrm{Y}^{(1)}, \mathrm{Y}^{(2)}, \Omega) = (g_1^s, g_2^s, g_1^{\omega})$, where the $\mathbb{G}_2$ component corresponds to the public key in *the Simplest*, and $g_1^s \in \mathbb{G}_1$, $g_2^s \in \mathbb{G}_2$ will be needed for the inner product argument.

Including g_1^s in pk enables trivial forgeries of signatures of the form $\sigma = (\mu, \Gamma = \mathcal{H}(\cdot) g_1^{\mu})^s$, by setting $\sigma^* = (\mu + \delta, \Gamma \cdot (\mathrm{Y}^{(1)})^{\delta})$ for any $\delta \in \mathbb{Z}_p$. To prevent this, we introduce the group elements Θ_1, Θ_2 among the parameters generated in the (trusted) **setup** and compute the signature by multiplying the output of the $\mathcal{H}_1$ with Θ_1^{μ} before raise it to the secret key s. Having $g_1^s \in \mathbb{G}_1$, $g_2^s \in \mathbb{G}_2$ in pk moreover prevents programming g_1 to depend on the coCDH* challenge in the security proof, as done in *the Simplest*, since this change would be detectable through the pairing. Thus, we introduce the (third) component Ω of pk and update the signing procedure to also involve the corresponding secret, ω. As a result, signatures output by **sign** are if the form $\Gamma = (\mathcal{H}_1(\Delta, \ell, \mathsf{pk}) \cdot \Theta_1^{\mu})^s \cdot \Psi_1^{\omega \cdot \mu}$.

Note that the signature contains a new group element Ψ_1 (output by **setup** together with Ψ_2 to simplify the extraction of a coCDH* solution in the reduction).

The **homomorphic evaluation** draws inspiration from [6, 8], but with adaptations to compact evaluated signatures. Evaluated signatures from a single signer are aggregated as in the Simplest [6], since in this case they are fully succinct (just one group element). For computations involving two or more signers, evaluated signatures have three components: a signature $\Gamma \in \mathbb{G}_1$ which is as in [6], and two proofs $\tilde{\pi}_M, \tilde{\pi}_\Omega$. The first proof, $\tilde{\pi}_M$, is for the relation R_M (see below) about a commitment M being constructed using the expected public keys $\mathrm{Y}_{\mathsf{id}}^{(1)}$ and some messages $\hat{\mu}_{\mathsf{id}} = \sum_{i \in \mathsf{I}_{\mathsf{id}}} f_i \mu_i$ that add up to $\hat{\mu} = \sum_{\mathsf{id}} \hat{\mu}_{\mathsf{id}}$. The second proof, $\tilde{\pi}_\Omega$, is for the relation

SetUp(λ) $\to$ pp

$(p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e) = \mathcal{G} \leftarrow \mathit{BilinGroup}(\lambda)$
$\mathsf{ID}, \mathcal{T} \leftarrow \{0,1\}^{\lambda}, \mathcal{M} \leftarrow \mathbb{Z}_p$
$u \xleftarrow{\$} \mathbb{G}_1$ / for InnerProduct proofs
$\theta, \psi \xleftarrow{\$} \mathbb{Z}_p : \Theta_1 := g_1^{\theta}, \Theta_2 := g_2^{\theta}, \Psi_1 := g_1^{\psi}, \Psi_2 := g_2^{\psi}$
$\mathcal{H}_1 : \{0,1\}^* \to \mathbb{G}_1, \mathcal{H}_2 : \{0,1\}^* \to \mathbb{Z}_p^*$ / hash functions
$\mathsf{pp} \leftarrow \left(\mathcal{G}, \mathsf{ID}, \mathcal{M}, \mathcal{T}, (\Theta_1, \Theta_2), (\Psi_1, \Psi_2), u, \mathcal{H}_1, \mathcal{H}_2\right)$

KeyGen(pp) $\to$ (id, sk, pk)

$\mathsf{id} \xleftarrow{\$} \mathsf{ID}$
$\mathsf{sk} := (s, \omega) \xleftarrow{\$} \mathbb{Z}_p^* \times \mathbb{Z}_p^*$
$\mathsf{pk} := \left(\mathrm{Y}^{(1)} \leftarrow g_1^s, \mathrm{Y}^{(2)} \leftarrow g_2^s, \Omega \leftarrow g_1^{\omega}\right)$

Sign($\mathsf{sk} = (s, \omega), \Delta, \ell, \mu$) $\to \sigma$

$\sigma \leftarrow \left(\mathcal{H}_1(\Delta, \ell, \mathsf{pk}) \cdot \Theta_1^{\mu}\right)^s \cdot \Psi_1^{\omega \cdot \mu}$

Eval($\mathcal{P}, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, \{\mu_i\}_{i \in [n]}, \{\sigma_i\}_{i \in [n]}$) $\to \hat{\sigma}$

parse $\mathcal{P}$ as $(\{f_i\}_{i=1}^n, \cdot) \in \mathbb{Z}_p^n$
$\Gamma \leftarrow \prod_{i=1}^n \sigma_i^{f_i}$ / evaluated signature
for $\mathsf{id} \in \mathcal{P}$ / consider only distinct id
  $\hat{\mu}_{\mathsf{id}} \leftarrow \sum_{i \in \mathsf{I}_{\mathsf{id}}} \mu_i \cdot f_i \mod p$ / each signer's contribution
if $(t == 1)$ / single-signer signature
  $M \leftarrow \perp, \Omega \leftarrow \perp, \tilde{\pi}_M \leftarrow \perp, \tilde{\pi}_\Omega \leftarrow \perp$ / already succinct
else / run the simplified succinct inner product arguments
  $M \leftarrow \prod_{\mathsf{id} \in \mathcal{P}} (\mathrm{Y}_{\mathsf{id}}^{(1)})^{\hat{\mu}_{\mathsf{id}}}, \Omega \leftarrow \prod_{\mathsf{id} \in \mathcal{P}} \Omega_{\mathsf{id}}^{\hat{\mu}_{\mathsf{id}}}$
  $\mathsf{pp}_M \leftarrow (t, \{\mathrm{Y}_{\mathsf{id}}^{(1)}\}_{\mathsf{id} \in \mathcal{P}}, u), x_M \leftarrow \mathcal{H}_2(M, \mathsf{pp}_M)$
  $\mathsf{pp}_\Omega \leftarrow (t, \{\Omega_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, u), x_\Omega \leftarrow \mathcal{H}_2(\Omega, \mathsf{pp}_M)$
  $\tilde{\pi}_M \leftarrow \widetilde{\mathsf{InPrd}}.\mathrm{P}\left(\mathsf{pp}_M, M \cdot u^{x_M \cdot \hat{\mu}}, \{\hat{\mu}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}\right)$
  $\tilde{\pi}_\Omega \leftarrow \widetilde{\mathsf{InPrd}}.\mathrm{P}\left(\mathsf{pp}_\Omega, \Omega \cdot u^{x_\Omega \cdot \hat{\mu}}, \{\hat{\mu}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}\right)$
$\hat{\sigma} \leftarrow (\Gamma, M, \Omega, \tilde{\pi}_M, \tilde{\pi}_\Omega)$

Verify($\mathcal{P}, \Delta, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, \hat{\mu}, \hat{\sigma}$) $\to$ *ver*

parse $\mathcal{P}$ as $(\{f_i\}_{i=1}^n, \{\ell_i\}_{i=1}^n)$ and $\hat{\sigma}$ as $(\Gamma, M, \Omega, \tilde{\pi}_M, \tilde{\pi}_\Omega)$
if $(t == 1)$ / single-signer signature
  $ver \leftarrow e(\Gamma, g_2) == e\left((\mathrm{Y}_{\mathsf{id}}^{(1)})^{\hat{\mu}}, \Theta_2\right) \cdot e\left(\Omega_{\mathsf{id}}^{\hat{\mu}}, \Psi_2\right) \cdot e\left(\prod_{i=1}^n \mathcal{H}_1(\Delta, \ell_i, \mathsf{pk}_{\mathsf{id}})^{f_i}, \mathrm{Y}_{\mathsf{id}}^{(2)}\right)$
else / $\tilde{\pi}_M = (M, \cdots), \tilde{\pi}_\Omega = (\Omega, \cdots)$
  $\mathsf{pp}_M \leftarrow (t, \{\mathrm{Y}_{\mathsf{id}}^{(1)}\}_{\mathsf{id} \in \mathcal{P}}, u), x_M \leftarrow \mathcal{H}_2(M, \mathsf{pp}_M)$
  $\mathsf{pp}_\Omega \leftarrow (t, \{\Omega_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, u), x_\Omega \leftarrow \mathcal{H}_2(\Omega, \mathsf{pp}_\Omega)$
  $ver_1 \leftarrow \widetilde{\mathsf{InPrd}}.\mathrm{V}\left(\mathsf{pp}_M, M \cdot u^{x_M \cdot \hat{\mu}}, \tilde{\pi}_M\right)$
  $ver_2 \leftarrow \widetilde{\mathsf{InPrd}}.\mathrm{V}\left(\mathsf{pp}_\Omega, \Omega \cdot u^{x_\Omega \cdot \hat{\mu}}, \tilde{\pi}_\Omega\right)$
  $ver_3 \leftarrow e(\Gamma, g_2) == e(M, \Theta_2) \cdot e(\Omega, \Psi_2) \cdot \prod_{\mathsf{id} \in \mathcal{P}} e\left(\prod_{i \in \mathsf{I}_{\mathsf{id}}} \mathcal{H}_1(\Delta, \ell_i, \mathsf{pk}_{\mathsf{id}})^{f_i}, \mathrm{Y}_{\mathsf{id}}^{(2)}\right)$
  $ver \leftarrow (ver_1 \wedge ver_2 \wedge ver_3)$

Figure 4: Our smklhs scheme. $\widetilde{\mathsf{InPrd}}$ is described in Figure 3.

R_Ω (see below) about a commitment Ω being constructed using the public keys Ω_{id} and some messages $\hat{\mu}_{\mathsf{id}}$, such that $\hat{\mu} = \sum_{\mathsf{id}} \hat{\mu}_{\mathsf{id}}$.

$$\mathsf{R}_M := \left\{ \{\mathrm{Y}^{(1)}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, M, \hat{\mu}; \{\hat{\mu}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}} \;:\; M = \prod_{\mathsf{id}\in\mathcal{P}} (\mathrm{Y}^{(1)}_{\mathsf{id}})^{\hat{\mu}_{\mathsf{id}}} \wedge \sum_{\mathsf{id}\in\mathcal{P}} \hat{\mu}_{\mathsf{id}} = \hat{\mu} \right\}$$

$$\mathsf{R}_\Omega := \left\{ \{\Omega_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, \Omega, \hat{\mu}; \{\hat{\mu}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}} \;:\; \Omega = \prod_{\mathsf{id}\in\mathcal{P}} \Omega_{\mathsf{id}}^{\hat{\mu}_{\mathsf{id}}} \wedge \sum_{\mathsf{id}\in\mathcal{P}} \hat{\mu}_{\mathsf{id}} = \hat{\mu} \right\}$$

Verification is split depending on the number of signers involved and simply entails verifying the inner product arguments (if present), and the signature (following *the Simplest*'s blueprint).

5 Security proof of our signature

In this section we prove that our succinct multi key linearly homomorphic signature scheme (smklhs, presented in Figure 4) satisfies the definition of authentication correctness, evaluation correctness, succinctness and security given in Section 2.5. This is formalized in Theorem 5.1.

Theorem 5.1. *Let $\mathscr{A}$ be an algebraic PPT adversary that breaks the weak-HUFCMA security with static corruptions of the* mklhs *scheme in Fig. 4 with advantage $\mathsf{Adv}_{\mathsf{mkhs},\mathscr{A}}$ in the Random Oracle Model (ROM). Then there exists a PPT adversary $\mathscr{B}^{\mathsf{coCDH}^*}$ that breaks the coCDH^* problem over $\mathbb{G}_1, \mathbb{G}_2$ with advantage $\mathsf{Adv}^{\mathsf{coCDH}^*}_{\mathbb{G}_1,\mathbb{G}_2}$ or a PPT prover $\widetilde{\mathcal{P}_{\mathsf{alg}}}$ that breaks the security of $\widetilde{\mathsf{InPrd}}$ for any extractor $\mathcal{E}$ such that for any security parameter λ it holds*

$$\mathsf{Adv}^{\mathsf{wHUFCMA}}_{\mathsf{mkhs}}(\mathscr{A},\lambda) \leq 2 \cdot \frac{\mathsf{Adv}^{\mathsf{coCDH}^*}_{\mathbb{G}_1,\mathbb{G}_2}(\mathscr{B}^{\mathsf{coCDH}^*},\lambda)}{1 - \mathsf{Adv}^{\mathsf{ip-ext}}_{\widetilde{\mathsf{InPrd}},\mathsf{R}^{\widetilde{\mathsf{InPrd}}}}(\widetilde{\mathcal{P}_{\mathsf{alg}}},\mathcal{E},\lambda)}$$

We prove Theorem 5.1 in subsection 5.1. Observe that one can easily prove the security of $\widetilde{\mathsf{InPrd}}$ in the AGM and ROM under dl-rel (cf. Appendix B).

5.1 Proof of Theorem 5.1

Fiore *et al.* proved a fundamental result for mkhs schemes: security against adversaries that do not make corruption queries is equivalent to security against adversaries that make non-adaptive corruption queries (cf. [20, Prop. 1]). This means that the proof of Theorem 5.1 does not need to deal with corruption queries; in particular, the only adaptive queries that the adversary can make are to create new honest parties, signing queries, and to the random oracles. Let $\mathscr{A}$ be the algebraic adversary against the unforgeability of smklhs that returns a forgery $(\mathcal{P}^*, \Delta^*, \{\mathsf{pk}^*_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, \hat{\mu}^*, \hat{\sigma}^*)$, and let $\mathscr{B}^{\mathsf{coCDH}^*}$ be the algorithm that reduces the forgery to a solution of an instance of the coCDH^* problem.

The reduction $\mathscr{B}^{\mathsf{coCDH}^*}$ takes as input a bilinear group $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_{2,}, e)$, and a coCDH^* challenge $(g_1, h_1 = g_1^{\mathsf{x}}, g_2, h_2 = g_2^{\mathsf{x}}, v)$ for $g_1, v \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, and its goal is to output v^{x}. To achieve this $\mathscr{B}^{\mathsf{coCDH}^*}$ simulates the algorithms SetUp,KeyGen,Sign and the hash functions $\mathcal{H}_1$, $\mathcal{H}_2$ of smklhs as defined below.

Simulation. The simulation receives and stores the collection of dataset, label and message triplets $L_{(\Delta,\ell,\mu)} = \{(\Delta,\ell,\mu)_i\}_{i\in[Q]}$.

Setup $\mathscr{B}^{\mathsf{coCDH}^*}$ initializes empty lists: $L_{ID}, L^{\mathsf{InPrd}}_{\mathcal{H}}$ and $L^{\mathsf{Sign}}_{\mathcal{H}}$. Then, it samples $\theta, \psi \xleftarrow{\$} \mathbb{Z}^*_p$ and sets $\Theta_1 = g_1^\theta, \Theta_2 = g_2^\theta$ and $\Psi_1 = h_1^\psi, \Psi_2 = h_2^\psi$. Finally, the reduction samples $u \xleftarrow{\$} \mathbb{G}_1$, and sends the public parameters to $\mathscr{A}$. Note that $\mathcal{G}$ is generated using the groups and generators from the coCDH^* challenge.

Identity Queries Given a query on the form $\mathsf{id} \in \mathsf{ID}$, if $\mathsf{id} \in L_{ID}$, *i.e.,* $\mathscr{A}$ already queried the identity id, then $\mathscr{B}^{\mathsf{coCDH}^*}$ ignores the query. Otherwise, the reduction samples the secret values $s_{\mathsf{id}}, \omega_{\mathsf{id}} \xleftarrow{\$} \mathbb{Z}^*_p$ and set the secret key to $\mathsf{sk}_{\mathsf{id}} = (s_{\mathsf{id}}, \omega_{\mathsf{id}})$, before computing the public key, $\mathsf{pk}_{\mathsf{id}} = (h_1^{s_{\mathsf{id}}}, h_2^{s_{\mathsf{id}}}, (v^{s_{\mathsf{id}}} g_1^{\omega_{\mathsf{id}}}))$. Then $\mathscr{B}^{\mathsf{coCDH}^*}$ updates the list of identities by $L_{ID} \leftarrow L_{ID} \cup \{(\mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}})\}$ before returning $\mathsf{pk}_{\mathsf{id}}$ to $\mathscr{A}$.

Sign Queries Given a query on the form $(\Delta, \mathsf{id}, \mu)$. If $(\Delta, \ell, \cdot, \cdot) \in L_\sigma$, *i.e.,* $\mathscr{A}$ already queried the same dataset and label pair, then $\mathscr{B}^{\mathsf{coCDH}^*}$ ignores the query[1]. If $(\mathsf{id}, \cdot, \cdot) \notin L_{ID}$, i.e. the identity specified by the label has not been initialized, $\mathscr{B}^{\mathsf{coCDH}^*}$ queries the identity oracle with input id before continuing. If $(\Delta, \ell, \mu, \cdot) \notin L^{\mathsf{Sign}}_{\mathcal{H}}$, i.e. the hash oracle has not been queried with input $(\Delta, \ell, \mathsf{pk}_{\mathsf{id}})$, $\mathscr{B}^{\mathsf{coCDH}^*}$ queries the hash oracle with that input before continuing. Then the reduction computes a signature as: $\sigma = h_1^{s_{\mathsf{id}}(t+\theta\mu)+\psi\omega_{\mathsf{id}}\mu}$, where t is the randomness stored in $L^{\mathsf{Sign}}_{\mathcal{H}}$ together with the triple (id, ℓ, μ). Then $\mathscr{B}^{\mathsf{coCDH}^*}$ updates the list $L_\sigma \leftarrow L_\sigma \cup \{(\Delta, \ell, \mu, \sigma)\}$ and returns the signature σ.

Hash Queries The reduction answer hash queries for $\mathcal{H}_1$ and $\mathcal{H}_2$ as follows.

$\mathcal{H}_1$: On input $(\Delta, \ell = (\mathsf{id}, \tau), \mathsf{pk}_{\mathsf{id}})$. If $(\Delta, \ell, \cdot, \cdot) \in L^{\mathsf{Sign}}_{\mathcal{H}}$, $\mathscr{B}^{\mathsf{coCDH}^*}$ retrieves the values (μ, t) stored in $L^{\mathsf{Sign}}_{\mathcal{H}}$ together with the queried dataset and label, and returns $v^{-\psi\mu} g_1^t$. If $(\Delta, \ell, \mu) \notin L_{(\Delta,\ell,\mu)}$, the reduction samples $t, \mu \xleftarrow{\$} \mathbb{Z}^*_p$. Else, it retrieves the message μ paired with Δ, ℓ in $L_{(\Delta,\ell,\mu)}$, samples $t \xleftarrow{\$} \mathbb{Z}^*_p$. Then $\mathscr{B}^{\mathsf{coCDH}^*}$ stores the tuple (Δ, ℓ, μ, t) in $L^{\mathsf{Sign}}_{\mathcal{H}}$ and returns $v^{-\psi\mu} g_1^t$.

$\mathcal{H}_2$: On query $(\mathsf{pp}, \tilde{P}, \{\tilde{L}_k, \tilde{R}_k\}_{k=1}^i, \{x_j\}_{j=1}^{i-1})$, if the query is already in $L^{\mathsf{InPrd}}_{\mathcal{H}}$, $\mathscr{B}^{\mathsf{coCDH}^*}$ retrieves and returns the hash value stored in $L^{\mathsf{InPrd}}_{\mathcal{H}}$.Otherwise, it samples $x \xleftarrow{\$} \mathbb{Z}^*_p$ and stores $((\mathsf{pp}, \tilde{P}, \{\tilde{L}_k, \tilde{R}_k\}_{k=1}^i, \{x_j\}_{j=1}^{i-1}), x)$ in $L^{\mathsf{InPrd}}_{\mathcal{H}}$ and returns x.

Correctness of the simulation. Below we argue that our reduction simulates identity, hash and sign queries in a perfect way. Let r denote the unknown discrete logarithm of v with respect to g_1. Simulated public keys are of the form, $\mathsf{pk}_{\mathsf{id}} = (\mathrm{Y}^{(1)}_{\mathsf{id}}, \mathrm{Y}^{(2)}_{\mathsf{id}}, \Omega_{\mathsf{id}}) = (h_1^{s_{\mathsf{id}}}, h_2^{s_{\mathsf{id}}}, v^{s_{\mathsf{id}}} g_1^{\omega_{\mathsf{id}}})$, with the corresponding secret values $\mathsf{sk}_{\mathsf{id}} = (s_{\mathsf{id}}, \omega_{\mathsf{id}})$. We can rewrite $\mathrm{Y}^{(1)}_{\mathsf{id}}$ as $g_1^{\hat{s}_{\mathsf{id}}}$ where $\hat{s}_{\mathsf{id}} = \mathsf{x}s_{\mathsf{id}}$ and similarly $\mathrm{Y}^{(2)} = g_2^{\hat{s}_{\mathsf{id}}}$. Moreover, we have that $\Omega_{\mathsf{id}} = v^{s_{\mathsf{id}}} g_1^{\omega_{\mathsf{id}}} = g_1^{rs_{\mathsf{id}}+\omega_{\mathsf{id}}} = g_1^{\hat{\omega}_{\mathsf{id}}}$, where $\hat{\omega}_{\mathsf{id}} = rs_{\mathsf{id}} + \omega_{\mathsf{id}}$. Since $s_{\mathsf{id}}, \omega_{\mathsf{id}}$ are sampled at random, $\hat{s_{\mathsf{id}}}, \hat{\omega}_{\mathsf{id}}$ are also uniformly random. Thus, we have showed that that simulated keys are is distributed identically to the keys output by the real KeyGen algorithm.

Second, the responses to all hash queries are uniformly random and thereby indistinguishable from the output of any real cryptographic hash function (in the random oracle model). We show this

[1] Observe that $\mathscr{A}$ cannot query a signature on (Δ, ℓ, μ_1) and (Δ, ℓ, μ_2) where $\mu_1 \neq \mu_2$, as by definition tags are unique per each message.

for the two simulated hash functions separately. Given a query on the form (Δ, ℓ) we can write the response as, $\mathcal{H}_1(\Delta, \ell, \mathsf{pk}_{\mathsf{id}}) = v^{-\psi\mu} g_1^t = g_1^{-\psi\mu r + t}$, from which it is clear that, since $t \xleftarrow{\$} \mathbb{Z}_p^*$ the hash values are uniformly distributed in $\mathbb{G}_1$. For $\mathcal{H}_2$ it follows direct by construction that its output is uniformly random.

The last step is to show that given the public parameters, public keys and answers to hash queries output by the reduction, for any identity, id, the simulated signatures are identical to the signatures output by real Sign algorithm (Figure 4). It suffices to show that for any query on the form $(\mathsf{id}, \mu, \ell, \Delta)$ the simulated signature is identical to signature output by the real Sign algorithm on the same input. Recall that above we showed that the simulated secret key is $\mathsf{sk}_{\mathsf{id}} = (\mathsf{x}s_{\mathsf{id}}, rs_{\mathsf{id}} + \omega_{\mathsf{id}})$, where $(s_{\mathsf{id}}, \omega_{\mathsf{id}})$ is known to the reduction. Let $\sigma \leftarrow \mathsf{Sign}(\hat{\mathsf{sk}}_{\mathsf{id}}, \Delta, \ell, \mu)$, we have that:

$$\begin{aligned}\sigma &= (\mathcal{H}_1(\Delta, \ell, \mathsf{pk}_{\mathsf{id}}) \cdot \Theta_1^{\mu})^{\hat{s}_{\mathsf{id}}} \cdot \Psi_1^{\hat{\omega}_{\mathsf{id}} \cdot \mu} \\ &= \left((v^{-\psi\mu} g_1^t) \cdot (g_1^{\theta})^{\mu}\right)^{\hat{s}_{\mathsf{id}}} \cdot (g_1^{\psi \mathsf{x}})^{\hat{\omega}_{\mathsf{id}}\mu} \\ &= \cdots = h_1^{s_{\mathsf{id}}(t+\theta\mu)+\omega_{\mathsf{id}}\psi\mu},\end{aligned}$$

where the final right-hand-side is equal to the simulated signature.

Finally, since a signature is statistically indistinguishable from a random element of $\mathbb{G}_1$, $\mathscr{A}$ cannot distinguish whether the output of $\mathcal{H}_1$ has been generated at random and the signature computed deterministically as in a real execution of the signing algorithm, or if the signature has been generated first as in the simulation.

The Single Signer Case. If the labeled program $\mathcal{P}^*$ only contains a single identity, *i.e.,* $t = 1$, then at the end of the game $\mathscr{A}$ outputs a forgery $(\mathcal{P}^*, \Delta^*, \hat{\mu}^*, \sigma^*)$, where $\sigma^* := (\sigma^*, \perp, \perp)$, along with an algebraic representation of σ^* respect to the group elements sent by the simulator. Since there is no inner product argument in this case, $\mathscr{B}^{\mathsf{coCDH}^*}$ skips the steps described in the next paragraph (**Abort Probability**) and proceeds directly to the extraction phase. Moreover, since Γ^* satisfies the verification check in Line 3 of Verify, by the bilinear property of the pairing, we can write Γ^* as

$$\Gamma^* == \left(\prod_{i\in[n]} \mathcal{H}_1(\Delta, \ell_i, \mathsf{pk}_{\mathsf{id}})^{f_i^*}\right)^{\mathsf{x}s_{\mathsf{id}}} \cdot (\Upsilon_{\mathsf{id}}^{(1)})^{\hat{\mu}^*\theta} \cdot \Omega_{\mathsf{id}}^{\hat{\mu}^*\mathsf{x}\psi} =$$
$$v^{\psi \mathsf{x} s_{\mathsf{id}}(\hat{\mu}^* - \sum_{i\in[n]} f_i\mu_i)} \cdot g_1^{\mathsf{x}s_{\mathsf{id}}(\sum_{i\in[n]} t_i f_i + \theta\hat{\mu}^*) + \psi\omega_{\mathsf{id}}\hat{\mu}^*} = v^{\square} g_1^{\spadesuit}.$$

Given this, extraction follows from the procedure outlined in the coming sections but with $\clubsuit = \diamondsuit := 0$ and $\square, \spadesuit$ as above.

Abort Probability. At the end of the game $\mathscr{A}$ returns a forgery $(\mathcal{P}^*, \Delta^*, \hat{\mu}^*, \sigma^*)$, where $\sigma^* := (\Gamma^*, M^*, \Omega^*, \tilde{\pi}_M^*, \tilde{\pi}_\Omega^*)$[2], alongside an algebraic representation of all the group elements. Let $\{K_i\}_{i\in[k]}$ denote the set of all group elements sent to the adversary during the simulation and $[M^*] := (M^*, a_1, \ldots, a_k)$, $[\Omega^*] := (\Omega^*, b_1, \ldots, b_k)$ be the representation of the two commitments M^* and Ω^* in the inner product proofs. The reduction then checks whether $a_i == 0$ for all $K_i \notin \{\Upsilon_{\mathsf{id}}^{(1)}\}_{\mathsf{id}\in\mathcal{P}^*}$, $b_i == 0$ for all $K_i \notin \{\Omega_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}$ and $\sum_{i\in[n]} a_i == \sum_{i\in[n]} b_i == \hat{\mu}^*$. If this is not the case, $\mathscr{B}^{\mathsf{coCDH}^*}$ aborts. The probability of this happening is bounded by the probability of breaking the security of the inner product argument. Indeed, if the adversary returns a representation of the commitment (M^* and Ω^* in the forgery) that assigns nonzero exponents to elements outside the instance, then there exists a reduction that solves the discrete logarithm problem (dl). This follows from the proof of Bulletproofs in the AGM [27] adapted to our setting as discussed in section 3, detailed in Appendix A, and Lemma B.1 in Appendix B.

Combining the two results, we have that the probability that $\mathscr{B}^{\mathsf{coCDH}^*}$ aborts is upper bounded by

$$\Pr[\mathscr{B}^{\mathsf{coCDH}^*} \textit{ aborts}] \leq \mathsf{Adv}^{\mathsf{ip-ext}}_{\widetilde{\mathsf{InPrd}}, \mathsf{R}^{\widetilde{\mathsf{InPrd}}}}(\widetilde{\mathcal{P}_{\mathsf{alg}}}, \mathcal{E}, \lambda) \tag{4}$$

that is, if $\mathscr{B}^{\mathsf{coCDH}^*}$ aborts with non-negligible probability, we can construct a PPT cheating prover $\widetilde{\mathcal{P}_{\mathsf{alg}}}$ that breaks the security of $\widetilde{\mathsf{InPrd}}$ for any extractor $\mathcal{E}$ exploiting $\mathscr{A}$.

Extraction. Without loss of generality, let the identities involved in $\mathcal{P}^*$ be $(\mathsf{id}_1, \mathsf{id}_2, \ldots, \mathsf{id}_t)$. Remember that all the identities in $\mathcal{P}^*$ are honest, as corruption queries are dismissed due to [20, Prop. 1]. If the reduction does not abort we have that,

$$M^* := \prod_{\mathsf{id}\in\mathcal{P}^*} (\Upsilon_{\mathsf{id}}^{(1)})^{a_{\mathsf{id}}} \quad \text{and} \quad \Omega^* := \prod_{\mathsf{id}\in\mathcal{P}^*} \Omega_{\mathsf{id}}^{b_{\mathsf{id}}}, \tag{5}$$

and by excluding all zeros from the representations we can write, $[M^*] := (M^*, a_{\mathsf{id}_1}, \ldots, a_{\mathsf{id}_t})$ and $[\Omega^*] := (\Omega^*, b_{\mathsf{id}_1}, \ldots, b_{\mathsf{id}_t})$.

By the correctness of the smklhs scheme the Γ^* component in σ^* must satisfy the verification check in line 9 in Verify:

$$\Gamma^* == \left(\prod_{\mathsf{id}\in\mathcal{P}^*} h_{\mathsf{id}}^{\mathsf{x}s_{\mathsf{id}}}\right) \cdot (M^*)^{\theta} \cdot (\Omega^*)^{\mathsf{x}\psi}, \text{ where,}$$
$$h_{\mathsf{id}} := \prod_{i\in\mathsf{I}_{\mathsf{id}}} \mathcal{H}_1(\Delta, \ell_i, \mathsf{pk}_{\mathsf{id}})^{f_i^*} = \prod_{i\in\mathsf{I}_{\mathsf{id}}} \left(v^{-\psi\mu_i} g_1^{t_i}\right)^{f_i^*},$$
$$M^* := \prod_{\mathsf{id}\in\mathcal{P}^*} g_1^{\mathsf{x}s_{\mathsf{id}}a_{\mathsf{id}}}, \Omega^* := \prod_{\mathsf{id}\in\mathcal{P}^*} ((v \cdot g_1^{\omega_{\mathsf{id}}})^{s_{\mathsf{id}}})^{b_{\mathsf{id}}}.$$

Abstracting away from the variable names Γ^* has the following structure: $\Gamma^* = v^{\diamondsuit+\square} g_1^{\clubsuit+\spadesuit}$, where $\diamondsuit$ and $\clubsuit$ collect the exponents linked to all non-target datasets/identities/labels and $\square$ and $\spadesuit$ correspond to exponents linked to the target datasets/identities/labels, where target and non target datasets/identities/labels will be defined momentarily for the different types of forgeries. Our aim is to extract from Γ^* its v component *i.e.,* $v^{\mathsf{x}\triangle}$ where $\triangle$ is a known non-zero value.

Type-1 Forgeries The reduction can detect if the adversary outputs a type-1 forgery by checking if the dataset identifier Δ^* defined in the labeled program $\mathcal{P}^*$ was queried to the sign oracle, *i.e.,* if $(\Delta^*, \cdot, \cdot) \in L_\sigma$. If that is the case, then $\mathscr{B}^{\mathsf{coCDH}^*}$ extracts from the forgery a witness to the co-CDH* instance as follows.

Since Δ^* is the same for all identities, we consider all identities to be target identities and thus, $\diamondsuit = \clubsuit = 0$. By inspection, we see that $\spadesuit \cdot \mathsf{x}^{-1} = \sum_{\mathsf{id}\in\mathcal{P}^*} s_{\mathsf{id}}\left(\sum_{i\in\mathsf{I}_{\mathsf{id}}} t_i f_i^* + \theta a_{\mathsf{id}}\right) + \psi\omega_{\mathsf{id}} b_{\mathsf{id}}$. Recall that $\spadesuit$ is a value known to the reduction and thereby $\mathscr{B}^{\mathsf{coCDH}^*}$ can remove the "g_1 component" contributions from Γ^* by computing

$$\Gamma_0 := \Gamma^* \cdot \left(g_1^{\spadesuit}\right)^{-1}.$$

[2] For the case $t \geq 2$.

At this point we are left with $\Gamma_0 := v^{\square}$ and $\square := \mathsf{x} \cdot \triangle$, where $\triangle := \psi\left(\sum_{\mathsf{id}\in\mathcal{P}^*} s_{\mathsf{id}}\big(b_{\mathsf{id}} - \sum_{i\in \mathsf{I}_{\mathsf{id}}} \mu_i f_i^*\big)\right)$. If $\mathscr{A}$ output a type-1 forgery it means that for all $\mathsf{id} \in \mathcal{P}^*$ and all $i \in \mathsf{I}_{\mathsf{id}}$ $\mathscr{A}$ does not know the value μ_i and cannot do better than randomly guessing its value. Indeed, assume w.l.o.g. that $\mathscr{A}$ queried $\mathcal{H}_1$ on (Δ^*, ℓ_i) for all the labels in the forgery, and got the oracle outputs $\{z_i \leftarrow v^{-\psi\mu_i} g_1^{t_i}\}_i$. Because the values of μ_i, t_i are sampled at random and not used anywhere else, from the point of view of $\mathscr{A}$ each element z_i is such that $z_i := g_1^u$, where u uniformly distributed over $\mathbb{Z}_p$. Therefore, the probability that $\triangle$ is zero is bounded by the probability of $\mathscr{A}$ guessing the value of $\sum_{i\in\mathsf{I}} \mu_i f_i^*$ for all $\mathsf{id} \in \mathcal{P}^*$ such that all terms cancel, which in turn is bounded by the size of the message space, *i.e.,* $|\mathbb{Z}_p|$ in our construction. We conclude that with probability $(1 - \frac{1}{p})$ the reduction extracts the solution to the co-CDH* challenge as:

$$\Gamma_1 := \Gamma_0^{(\triangle^{-1})} := v^{\mathsf{x}}. \tag{6}$$

Type-2 Forgeries Let $\hat{\mu}_{\mathsf{id}} := \sum_{i\in\mathsf{I}_{\mathsf{id}}} \mu_i f_i^*$, then the reduction can detect which identities the adversary targeted by checking if either a_{id} or b_{id} equals $\hat{\mu}_{\mathsf{id}}$. Next we define the set of targeted identities to be, $\mathsf{ID}_{\mathsf{target}} := \{\mathsf{id} \in \mathcal{P}^* : a_{\mathsf{id}} \vee b_{\mathsf{id}} \neq \hat{\mu}_{\mathsf{id}}\}$. The reduction $\mathscr{B}^{\text{coCDH}^*}$ looks for type-2 forgeries against $\mathsf{id} \in \mathsf{ID}_{\mathsf{target}}$, i.e, satisfying

$$\begin{cases} \mathsf{id} \in \mathsf{ID}_{\mathsf{target}} \subset \mathcal{P}^* \\ \forall \mathsf{id} \in \mathsf{ID}_{\mathsf{target}} : a_{\mathsf{id}} \neq \hat{\mu}_{\mathsf{id}} \vee b_{\mathsf{id}} \neq \hat{\mu}_{\mathsf{id}} \end{cases} \tag{7}$$

We remark that since $\mathscr{A}$ outputs a type-2 forgery it follows that $\hat{\mu}^* := \sum_{\mathsf{id}\in\mathcal{P}^*} b_{\mathsf{id}} \neq \sum_{\mathsf{id}\in\mathcal{P}^*} \hat{\mu}_{\mathsf{id}}$, thus there must exist at least one $\mathsf{id} \in \mathsf{ID}_{\mathsf{target}}$ such that $b_{\mathsf{id}} \neq \hat{\mu}_{\mathsf{id}}$, and analogously for a_{id}.

If the adversary output a type-2 forgery, $\mathscr{B}^{\text{coCDH}^*}$ successfully extracts an answer to the co-CDH* challenger from the type-2 forgery against the target identities, $\mathsf{ID}_{\mathsf{target}}$.

By definition $a_{\mathsf{id}} = b_{\mathsf{id}} = \hat{\mu}_{\mathsf{id}}$ for all non-target identities, $\mathsf{id} \in \mathcal{P}^* \backslash \mathsf{ID}_{\mathsf{target}}$. Thus, $\diamond = 0$ and $\clubsuit \cdot \mathsf{x}^{-1} = \sum_{\mathsf{id}\in\mathcal{P}^*\backslash\mathsf{ID}_{\mathsf{target}}} s_{\mathsf{id}}\big(\sum_{i\in\mathsf{I}_{\mathsf{id}}} t_i f_i^* + \theta\hat{\mu}_{\mathsf{id}}\big) + \psi\omega_{\mathsf{id}}\hat{\mu}_{\mathsf{id}}$. On the other hand, for the target identities it holds that, $\spadesuit \cdot \mathsf{x}^{-1} = \sum_{\mathsf{id}\in\mathsf{ID}_{\mathsf{target}}} s_{\mathsf{id}}\big(\sum_{i\in\mathsf{I}_{\mathsf{id}}} t_i f_i^* + \theta a_{\mathsf{id}}\big) + \psi\omega_{\mathsf{id}} b_{\mathsf{id}}$. The value of $\clubsuit$ and $\spadesuit$ are known to the reduction, thus $\mathscr{B}^{\text{coCDH}^*}$ can remove the "g_1 component" from Γ^* by computing

$$\Gamma_0 := \Gamma^* \cdot \big(g_1^{\clubsuit+\spadesuit}\big)^{-1} \tag{8}$$

At this point we are left with $\Gamma_0 := v^{\square}$, where $\square := \mathsf{x} \cdot \triangle$ and $\triangle := \psi\left(\sum_{\mathsf{id}\in\mathsf{ID}_{\mathsf{target}}} s_{\mathsf{id}}(b_{\mathsf{id}} - \hat{\mu}_{\mathsf{id}})\right)$, and $\triangle \neq 0$ by condition (14). Finally, $\mathscr{B}^{\text{coCDH}^*}$ to extracts the solution for the co-CDH* challenge using the by same computation as in (6).

Type-3 Forgeries The reduction can detect which labels the adversary targeted by checking which labels $\ell_i^* \in \mathcal{P}^*$ were not queried to the sign oracle, *i.e.,* $(\Delta^*, \ell_i^*, \cdot) \notin L_\sigma$. Thus, we define the set of targeted labels to be, $\ell_{\mathsf{target}} := \{\ell^* \in \mathcal{P}^* : (\Delta^*, \ell_i^*, \cdot) \notin L_\sigma\}$. Furthermore, we define the target identities to be as follows, $\mathsf{ID}_{\mathsf{target}} := \{\mathsf{id} \in \mathcal{P}^* : \exists\, \ell^* = (\cdot, \mathsf{id}) \in \ell_{\mathsf{target}}\}$ $\mathscr{B}^{\text{coCDH}^*}$ extracts a solution to the co-CDH* problem from the forgery as follows. For all labels we have that $\mathcal{H}_1(\Delta, \ell_i, \mathsf{pk}_{\mathsf{id}}) \leftarrow v^{\psi\mu_i} g_1^{t_i}$. Specifically for all non target labels, *i.e.,* $\forall \ell_i^* \in \mathcal{P}^* \backslash \ell_{\mathsf{target}}$, the value of μ_i is the message identified by Δ^*, ℓ^* and the value of t_i is included in the signature Γ_i. However, for all target labels, *i.e.,* $\forall \ell_i^* \in \ell_{\mathsf{target}}$, the values of μ_i, t_i are sampled at random by $\mathscr{B}^{\text{coCDH}^*}$ upon generating the hash and not shared with $\mathscr{A}$ (this is analogous to what happens in the case of type-1 forgeries). Hence, the value of μ_i is information-theoretically hidden, and $\mathscr{A}$ cannot to better than randomly guessing its value.

By inspection, we see that $\clubsuit \cdot \mathsf{x}^{-1} = \sum_{\mathsf{id}\in\mathcal{P}^*\backslash\mathsf{ID}_{\mathsf{target}}} s_{\mathsf{id}}\big(\sum_{i\in\mathsf{I}_{\mathsf{id}}} t_i f_i^* + \theta a_{\mathsf{id}}\big) + \psi\omega_{\mathsf{id}} b_{\mathsf{id}}$ and $\spadesuit \cdot \mathsf{x}^{-1} = \sum_{\mathsf{id}\in\mathsf{ID}_{\mathsf{target}}} s_{\mathsf{id}}\big(\sum_{i\in\mathsf{I}_{\mathsf{id}}} t_i f_i^* + \theta a_{\mathsf{id}}\big) + \psi\omega_{\mathsf{id}} b_{\mathsf{id}}$. We remark that both $\clubsuit$ and $\spadesuit$ are known to the reduction and thereby $\mathscr{B}^{\text{coCDH}^*}$ can remove the "g_1 component" for all identities–target or not– from Γ^* by computing Γ_0 as in (8).

At this point we are left with $\Gamma_0 := v^{\square+\diamond}$. First, consider $\diamond := x \cdot \triangle$ and $\triangle = \psi\big(\sum_{\mathsf{id}\in\mathcal{P}^*\backslash\mathsf{ID}_{\mathsf{target}}} s_{\mathsf{id}}\big(b_{\mathsf{id}} - \sum_{i\in\mathsf{I}_{\mathsf{id}}} \mu_i f_i^*\big)\big)$. Since $\mathscr{A}$ behaves honestly for $\mathsf{id} \in \mathcal{P}^*\backslash\mathsf{ID}_{\mathsf{target}}$, we have that $b_{\mathsf{id}} = \sum_{i\in\mathsf{I}_{\mathsf{id}}} \mu_i f_i^*$, and consequently $\triangle = 0$.

Next consider $\square = \mathsf{x}\cdot\triangle$ and $\triangle = \psi\big(\sum_{\mathsf{id}\in\mathsf{ID}_{\mathsf{target}}} s_{\mathsf{id}}\big(b_{\mathsf{id}} - \sum_{i\in\mathsf{I}_{\mathsf{id}}} \mu_i f_i^*\big)\big)$. If $\mathscr{A}$ output a type-3 forgery it means that for all $\mathsf{id} \in \mathsf{ID}_{\mathsf{target}}$ there exists at least one $i \in \mathsf{I}_{\mathsf{id}}$ such that $\ell_i^* \in \ell_{\mathsf{target}}$ and consequently for which $\mathscr{A}$ don't know the value μ_i and can't do better than randomly guessing its value. Therefore, the probability that $\triangle$ is zero is bounded by the probability of $\mathscr{A}$ guessing the value of $\sum_{i\in\mathsf{I}} \mu_i f_i^*$ for all $\mathsf{id} \in \mathsf{ID}_{\mathsf{target}}$ such that all terms cancel. This is bounded by the size of the message space, *i.e.,* $|\mathbb{Z}_p|$ in our construction.

With probability $(1 - \frac{1}{p})$ $\mathscr{B}^{\text{coCDH}^*}$ extracts the solution to the co-CDH* from Γ_0 by the computation in (6).

Success Probability. The success probability of $\mathscr{B}^{\text{coCDH}^*}$ in winning the coCDH* game can be bounded by the minimum success probability depending on the type of forgery. That is,

$$\begin{aligned} &\mathsf{Adv}^{\text{coCDH}^*}_{\mathbb{G}_1,\mathbb{G}_2}(\mathscr{B}^{\text{coCDH}^*}, \lambda) \geq \\ &\geq \left(1 - \frac{1}{p}\right) \cdot \mathsf{Adv}^{\text{wHUFCMA}}_{\text{mkhs}}(\mathscr{A}, \lambda) \cdot \left(1 - \Pr[\mathscr{B}^{\text{coCDH}^*} \textit{ aborts}]\right) \\ &\geq \frac{1}{2}\mathsf{Adv}^{\text{wHUFCMA}}_{\text{mkhs}}\left(1 - \mathsf{Adv}^{\text{ip-ext}}_{\widetilde{\text{InPrd}},\mathsf{R}^{\widetilde{\text{InPrd}}}}(\widetilde{\mathcal{P}_{\text{alg}}}, \mathcal{E}, \lambda)\right) \end{aligned}$$

where we used the bound $\left(1 - \frac{1}{p}\right) \geq \frac{1}{2}$ and Equation (4).

6 Our adaptive secure construction

We extend our construction from Figure 4 to fully adaptive security in Figure 11 in Appendix E. We sketch here only the intuition of the adjustments and defer all details to Appendix E.

In somewhat oversimplified terms, our adaptive secure construction consists of two instances of the weak construction: the first instance runs on random messages, the second uses real messages, carefully masked by the random ones to preserve unforgeability but allow for simulation in the adaptive setting.

In more detail, we set up public parameters of the weak construction for the random-messages instance, and for the masked-message instance. The setups are independent except for using the same bilinear group, generators, set of identities/labels. Similarly, key generation corresponds to running the weakly-adaptive key generation twice (once for each instance). Instead of signing a message μ directly, the new construction proceeds as follows: First, we sample a random message r, then we sign it (using the weak adaptive secure construction) to obtain σ_r, and finally we sign μ and r

as

$$\sigma_\mu \leftarrow \left(\mathcal{H}_1(\Delta, \ell, \mathsf{pk}_{\mathsf{id}}) \cdot \Theta_{\mu,1}^{\mu}\right)^{s_{\mathsf{id},\mu}} \cdot \Psi_{\mu,1}^{\omega_{\mathsf{id},\mu}\cdot(\mu+r)}$$

where $\Theta_{\mu,1}, \Psi_{\mu,1}$ are public parameters (for the masked-message instance) and $\omega_{\mathsf{id},\mu}$ is part of the secret key for (for the masked-message instance). The final signature is, $\sigma = (r, \sigma_r, \sigma_\mu)$.

The structure of σ_μ is crucial for adaptive security. Indeed, for the reduction to work correctly, the simulator needs produce σ_μ without knowing the secret key. This is easy in the weakly-adaptive secure case, where the adversary declares all signing queries at the beginning of the game, so $\mathcal{H}_1(\Delta, \ell, \mathsf{pk}_{\mathsf{id}})$ can depend on the message μ. But this is not the case in full adaptive security. To circumvent this issue, we introduce a new element r in σ_μ, which allows the simulator to adjust the signature even if the random oracle was already programmed on $\mathcal{H}_1(\Delta, \ell, \mathsf{pk})$. The trick is to let r depend on the message μ, and rely on the fact that the weak construction is actually unforgeable against adversaries making adaptive queries, as long as the messages are random (cf. Appendix D). Finally, given that r appears in both σ_r and σ_μ we need to modify the inner product protocol to prove that the respective relations hold in parallel (that is, w.r.t. the same witness r).

Theorem 6.1. *Let $\mathscr{A}$ be an algebraic PPT adversary that breaks the HUFCMA security with static corruptions of the* mklhs *scheme in Fig. 11 with advantage $\mathsf{Adv}_{\mathsf{mkhs},\mathscr{A}}$ in the Random Oracle Model (ROM). Then there exists a PPT adversary $\mathscr{B}^{\mathsf{coCDH}^*}$ that breaks the coCDH^* problem over $\mathbb{G}_1, \mathbb{G}_2$ with advantage $\mathsf{Adv}^{\mathsf{coCDH}^*}_{\mathbb{G}_1,\mathbb{G}_2}$ or a PPT prover $\widetilde{\mathcal{P}_{\mathsf{alg}}}$ that breaks the security of $\widetilde{\mathsf{InPrd}}$ for any extractor $\mathcal{E}$ such that for any security parameter λ it holds*

$$\mathsf{Adv}^{\mathsf{HUFCMA}}_{\mathsf{mkhs}}(\mathscr{A}, \lambda) \leq \quad 2 \cdot \frac{\mathsf{Adv}^{\mathsf{coCDH}^*}_{\mathbb{G}_1,\mathbb{G}_2}(\mathscr{B}^{\mathsf{coCDH}^*}, \lambda)}{1 - \Pr[\mathscr{B}^{\mathsf{coCDH}^*} \textit{ aborts}]},$$

where $\Pr[\mathscr{B}^{\mathsf{coCDH}^*} \textit{ aborts}] = \Big(\mathsf{Adv}^{\mathsf{wHUFCMA}}_{\mathsf{mkhs}}(\mathscr{A}, \lambda) + \mathsf{Adv}^{\mathsf{ip-ext}}_{\widetilde{\mathsf{InPrd}},R^{\widetilde{\mathsf{InPrd}}}}(\widetilde{\mathcal{P}_{\mathsf{alg}}}, \mathcal{E}, \lambda)\Big)$.

Like in Section 5, we prove security of $\widetilde{\mathsf{InPrd}}$ from dl-rel in the ROM and AGM (cf. Section A). The proof of Theorem 6.1, in Appendix E, is very similar to the proof of Theorem 5.1.

7 Performance evaluation

We implemented smklhs in the latest version of RELIC [4] using curve BLS12-381, a *de facto* standard for pairing friendly curves at the 128-bit security level. The main performance features of this parameter choice that benefit our scheme are efficient exponentiations in all groups accelerated by fast endomorphisms [24], an optimal pairing with fast extension field arithmetic, fast hashing through the SwiftEC algorithm [5, 17], and Assembly acceleration for Intel 64-bit platforms.

Most operations in our scheme happen in $\mathbb{G}_1$, so it is well-tailored to asymmetric pairings. The only important operation involving arithmetic in the more expensive $\mathbb{G}_2$ is key generation, but it is assumed to be performed offline and once per user. Computing a signature involves hashing the $(\Delta, \ell\ \mathsf{pk})$ to $\mathbb{G}_1$ using $\mathcal{H}_1$, and performing three variable-base exponentiations by μ, s and $(\omega_{\mathsf{id}} \cdot \mu)$, hence both operations dominate the execution time for generating signatures. The performance-critical operations in the homomorphic evaluation of smklhs are multi-exponentiation in $\mathbb{G}_1$, corresponding to the computation of $\Gamma \leftarrow \prod_{i=1}^{n} \sigma_i^{f_i}$, and the two inner product arguments $\tilde{\pi}_M, \tilde{\pi}_\Omega$ proving that aggregations are correct. Verifying a multi key homomorphic signature for computations with t signers involves hashing to $\mathbb{G}_1$, variable-base exponentiations in $\mathbb{G}_1$ and a product of $(t + 3)$ pairings. When there are multiple signers ($t > 1$), the inner product arguments must be verified as well. In our implementation, we choose Δ to be a global database identifier and ℓ to be an entry-specific label, but the interface is flexible for other context-specific configurations.

From a side-channel analysis point of view, the only critical procedure is the signature generation, for which we implemented operations involving the private keys in constant time. All other operations were implemented in variable-time for better performance, since they rely only on public or ephemeral data. The inner product arguments were implemented for power of 2 sizes, and dummy values were used as padding otherwise. The resulting code is available on GitHub[3], with changes to the library highlighted in the `README`.

7.1 Experimental results and discussion

We benchmarked our implementation on a modest Intel Core i7-7700 Kaby Lake processor running at 3.6GHz, with HyperThreading and TurboBoost turned off to reduce noise. For comparison with related work, we also benchmarked the schemes by Schabhüser *et al.* [37] (with BLS [11] as the underlying signature algorithm) and the Simplest [6] (denoted mklhs). The benchmarking dataset was created by generating 10 different users, who sign 16 individual messages with unique labels. These signatures are collected by a third party, which homomorphically evaluates a linear function f composed of 16 random non-zero 32-bit integer coefficients across all signatures, and the resulting multi key signature is then verified. Each procedure is executed 10^4 times and the timings are presented in Table 2. The figures are amortized per user, so similar performance are expected for datasets of different size. Fresh signatures appear slower to verify in [37] because the sublinear portions are amortized during verification after homomorphic evaluation. The reverse is observed in smklhs because signatures include the inner product argument after evaluation that also needs to be verified.

In comparison to related work, our scheme achieves a series of trade-offs. It is more efficient than Schabhüser *et al.* for key generation (24.9x), signing (2.2x) and verification (1.45–3.4x), while trading-off context-hiding and faster evaluation for much more compact signatures. The main reason behind the slower homomorphic evaluation (8x) is the computation of inner product arguments, which allows our signature size reduction from linear to logarithmic. In comparison to mklhs, our scheme trade-offs performance for succinctness, being 1.84x slower for key generation and up to 1.25x slower for signing or verifying evaluated signatures. We start obtaining concrete savings in terms of signature size for $t \geq 38$. The adaptive secure version (adpt.smklhs) is a little more than 2x slower than the weaker construction, since it essentially consists of running two of its instances, impacting all operations similarly.

[3] https://github.com/dfaranha/smklhs/blob/main/README.md

Table 2: Efficiency comparison of our schemes with related work. We display execution times in clock cycles (cc) for each of the main algorithms. The dataset includes homomorphic evaluation and verification of 16 signatures computed by $t = 10$ users each. Figures represent the average of 10^4 executions. Timings for signature computation and fresh signature verification refer to individual signatures, and timings for homomorphic evaluation and verification are amortized per user. Fresh and combined signature sizes are also displayed in bytes, assuming point compression of $\mathbb{G}_1$ and $\mathbb{G}_2$ elements. $|sgn|$ denotes the size of an arbitrary signature scheme needed in Schabhüser *et al.*'s compiler [37].

Operation / Protocol	Schabhüser *et al.* [37]	*The Simplest* [6]	smklhs	adpt.smklhs
KeyGen	16,593,650	362,273	667,737	1,332,424
Sign	2,832,021	1,026,648	1,290,311	2,365,773
Verify (fresh)	17,677,199	5,290,656	5,151,515	10,039,193
Hom. evaluation	2,482,164	502,902	4,000,607	7,517,623
Verify (eval.)	10,320,599	5,699,010	7,046,625	10,355,423
Size of fresh sig	384	48	48	128
Size of eval. sigs	$144 + 240t$	$48 + 32t$	$256 + 192\log_2 t$	$576 + 480\log_2 t$
eval. sigs structure	$t\|sgn\| + (2t+1)\|\mathbb{G}_1\| + (t+1)\|\mathbb{G}_2\|$	$t \cdot \|\mathbb{Z}_p\| + 1 \cdot \|\mathbb{G}_1\|$	$2\|\mathbb{Z}_p\| + (4 + 4\log_2 t) \cdot \|\mathbb{G}_1\|$	$6 \cdot \|\mathbb{Z}_p\| + (8 + 10\log_2 t)\|\mathbb{G}_1\|$

7.2 Case study

We now illustrate the application of our technique to computing COVID-19 mortality metrics using data provided by the government of Spain in the EuroMOMO platform [34]. EuroMOMO monitors mortality across Europe to measure the impact of heat waves and other seasonal events. Our case study is inspired by and uses data from The Economist's COVID-19 excess deaths tracker [19], where an independent organization aggregates government-published data to derive metrics. Our approach allows the metrics to be computed in a publicly verifiable and authenticated way, assuming that government-published data is authentic and authenticated in the first place. We assume demographic data to be public, allowing the linear functions $f(\cdot)$ to be defined using public data only.

Relevant mortality metrics for monitoring pandemics are the Excess Mortality (EM) and Standardized Mortality Rate (SMR)[4]. The EM metric is the difference $(D_O - D_E)$ between observed deaths (D_O) and expected deaths (D_E), and is used to compare currently observed all-cause mortality rates with a baseline. It is a useful metric to estimate the overall impact of a disease, especially when suboptimal notification is in place or when healthcare systems are overwhelmed, leading to higher mortality for other causes. The SMR is defined as the ratio D_O/D_E between observed deaths and expected deaths and can be used to compare the morality risk of a certain study population to that of a standard population. It is especially applicable when the two populations have different age distributions, as it is typically the case for populations living in different states of large countries. In our particular case, we consider the whole population of Spain as the standard population, and a specific state population as the study one, so that variations in age distributions across different states are considered.

We assume that a distributed mortality reporting system is maintained by each state (or *autonomous community* in the case of Spain). We denote the set of states by S and the age groups by G. Each state health authority is equipped with its own independent signing key to authenticate and publish daily mortality records. Each individual record summarizes the number $m_{i,j,k}$ of deaths in day i for state j in a certain age group k.

[4]https://ibis.doh.nm.gov/resource/SMR_ISR.html

We now focus on computing observed deaths (D_O) and expected deaths (D_E), from which the SMR and EM metrics can be obtained. Using this notation, the total number of observed deaths D_O in an evaluation period P can be simply defined as:

$$D_O = \sum_{i \in P} \sum_{j \in S} \sum_{k \in G} m_{i,j,k},$$

where $m_{i,j,k}$ is the number of deaths in day i for state j in a certain age group k.

Computing D_O homomorphically using smklhs over authenticated records $m_{i,j,k}$ is trivial and amounts to addition over all records. The number of expected deaths D_E can be computed by first calculating the age-specific yearly mortality rate per million $d_{R,k}$ in the standard population in a certain age group k over a baseline period B. We denote by p_k the total population in age group k during B, and we select B to be the same period P but in the y previous years. To avoid computation involving real numbers that would not be supported in smklhs, we adjust the formula by a fixed-point scaling factor $s = 10^4$ and compute the *scaled* metric instead:

$$d_{R,k} = \frac{s}{y} \cdot \sum_{i \in B} \sum_{j \in S} \left(\frac{m_{i,j,k}}{p_k \cdot 10^{-6}} \right).$$

Now we can compute the *scaled* number of expected deaths for state j by multiplying the yearly mortality rate per million by the current state population $q_{j,k}$ in age group k, across all groups:

$$d_{E,j} = 10^{-6} \cdot \sum_{k \in G} (q_{j,k} \cdot d_{R,k}).$$

The total number of *scaled* expected deaths D'_E can then be computed by simply summing across states as $D'_E = \sum_{j \in S} d_{E,j}$. Computing D'_E homomorphically using smklhs requires rewriting the expression as an integer linear combination of authenticated records:

$$D'_E = \sum_{j \in S} 10^{-6} \cdot \sum_{k \in G} (q_{j,k} \cdot d_{R,k}) = \sum_{j \in S} \sum_{k \in G} \sum_{i \in B} \left(\sum_{j' \in S} \frac{s}{y} \cdot \frac{q_{j',k}}{p_k} \right) \cdot m_{i,j,k}$$

The computation can then be performed with $f_k = \left(\sum_{j' \in S} \frac{s}{y} \cdot \frac{q_{j',k}}{p_k} \right)$ as $D'_E = \sum_{j \in S} \sum_{k \in G} \sum_{i \in B} f_k \cdot m_{i,j,k}$. The *real* number of expected deaths is then given by $D_E = D'_E/s$ after removing the scaling

factor. By computing both D_O and D_E in an authenticated manner, the remaining SMR and EM metrics can be easily computed. Data published by Spain includes daily mortality reports across three different age groups ($|G| = 3$: younger than 65, between 65 and 74, older than 74) over the previous 2-year period ending on the publishing date. In our implementation, we selected a 180-day evaluation period ($|P| = 180$) starting in March 27, 2020, which corresponds to a time window where COVID-19 particularly affected Spain. The baseline period is set to the same period ($|B| = 180$) in years 2018 and 2019 ($y = 2$), collected from The Economist's historical dataset[5]. The population for each of the autonomous communities ($|S| = 19$) was collected from the same source, and population pyramids to split in age groups were taken from a public source[6]. The code is available on GitHub[7].

For the experiment, we obtained EM = 43, 587 and SMR = 1.24. The efficiency of our scheme allows to verify the whole computation involving $(y+1)\cdot|S|\cdot|G|\cdot|B| = 30,780$ signatures in 2.20 seconds. The cost of verification in our experiment is essentially linear, so we replicated the dataset 10 times to scale the experiment up to 190 signers, which is close to the number of member countries in the World Health Organization (WHO). Verifying 307,800 records for the same time period then takes 22 seconds, with a 70.8% reduction in signature size in comparison to [6] for this number of signers.

8 Conclusion and further directions

We presented two concrete realizations of MKHS that enjoy the asymptotically shortest evaluated signatures, are efficient, and rely solely on well studied falsifiable assumptions.

Our open-source implementation builds on the renown RELIC library, and offers a ready-to-use tool for publicly accountable data reporting and analytics, with potential applications in critical domains such as healthcare, journalism, transparency-oriented government services, federated learning and monitoring systems.

Future work includes extending our adaptively secure construction to support more expressive evaluations, analyzing whether unforgeability can be proven in just ROM (without AGM), and exploring context-hiding and multi-hop evaluation while maintaining succinctness and efficiency.

[5]https://github.com/TheEconomist/covid-19-excess-deaths-tracker
[6]https://countryeconomy.com/countries/spain
[7]https://github.com/dfaranha/smklhs/blob/main/README.md

References

[1] Shweta Agrawal, Dan Boneh, Xavier Boyen, and David Mandell Freeman. 2010. Preventing pollution attacks in multi-source network coding. In International Workshop on Public Key Cryptography. Springer, 161–176.

[2] Thomas Agrikola, Dennis Hofheinz, and Julia Kastner. 2020. On Instantiating the Algebraic Group Model from Falsifiable Assumptions. In EUROCRYPT 2020, Part II (LNCS, Vol. 12106), Anne Canteaut and Yuval Ishai (Eds.). Springer, Cham, 96–126. doi:10.1007/978-3-030-45724-2_4

[3] Gaspard Anthoine, David Balbás, and Dario Fiore. 2024. Fully-Succinct Multi-key Homomorphic Signatures from Standard Assumptions. In CRYPTO 2024, Part III (LNCS, Vol. 14922), Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 317–351. doi:10.1007/978-3-031-68382-4_10

[4] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. [n. d.]. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[5] Diego F. Aranha, Benjamin Salling Hvass, Bas Spitters, and Mehdi Tibouchi. 2023. Faster Constant-time Evaluation of the Kronecker Symbol with Application to Elliptic Curve Hashing. In ACM CCS 2023, Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda (Eds.). ACM Press, 3228–3238. doi:10.1145/3576915.3616597

[6] Diego F. Aranha and Elena Pagnin. 2019. The Simplest Multi-key Linearly Homomorphic Signature Scheme. In LATINCRYPT 2019 (LNCS, Vol. 11774), Peter Schwabe and Nicolas Thériault (Eds.). Springer, Cham, 280–300. doi:10.1007/978-3-030-30530-7_14

[7] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In ACM CCS 93, Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby (Eds.). ACM Press, 62–73. doi:10.1145/168588.168596

[8] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. 2009. Signing a Linear Subspace: Signature Schemes for Network Coding. In PKC 2009 (LNCS, Vol. 5443), Stanislaw Jarecki and Gene Tsudik (Eds.). Springer, Berlin, Heidelberg, 68–87. doi:10.1007/978-3-642-00468-1_5

[9] Dan Boneh and David Mandell Freeman. 2011. Homomorphic Signatures for Polynomial Functions. In EUROCRYPT 2011 (LNCS, Vol. 6632), Kenneth G. Paterson (Ed.). Springer, Berlin, Heidelberg, 149–168. doi:10.1007/978-3-642-20465-4_10

[10] Dan Boneh and David Mandell Freeman. 2011. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In International Workshop on Public Key Cryptography. Springer, 1–16.

[11] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In ASIACRYPT 2001 (LNCS, Vol. 2248), Colin Boyd (Ed.). Springer, Berlin, Heidelberg, 514–532. doi:10.1007/3-540-45682-1_30

[12] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. 2016. Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting. In EUROCRYPT 2016, Part II (LNCS, Vol. 9666), Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer, Berlin, Heidelberg, 327–357. doi:10.1007/978-3-662-49896-5_12

[13] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. In 2018 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, 315–334. doi:10.1109/SP.2018.00020

[14] Dario Catalano, Dario Fiore, and Bogdan Warinschi. 2012. Efficient network coding signatures in the standard model. In International Workshop on Public Key Cryptography. Springer, 680–696.

[15] Dario Catalano, Dario Fiore, and Bogdan Warinschi. 2014. Homomorphic signatures with efficient verification for polynomial functions. In International Cryptology Conference. Springer, 371–389.

[16] Sanjit Chatterjee and Alfred Menezes. 2011. On cryptographic protocols employing asymmetric pairings — The role of Φ revisited. Discrete Applied Mathematics 159, 13 (2011), 1311–1322. doi:10.1016/j.dam.2011.04.021

[17] Jorge Chávez-Saab, Francisco Rodríguez-Henríquez, and Mehdi Tibouchi. 2022. SwiftEC: Shallue-van de Woestijne Indifferentiable Function to Elliptic Curves - Faster Indifferentiable Hashing to Elliptic Curves. In ASIACRYPT 2022, Part I (LNCS, Vol. 13791), Shweta Agrawal and Dongdai Lin (Eds.). Springer, Cham, 63–92. doi:10.1007/978-3-031-22963-3_3

[18] Somayeh Dolatnezhad Samarin, Dario Fiore, Daniele Venturi, and Morteza Amini. 2021. A compiler for multi-key homomorphic signatures for Turing machines. Theoretical Computer Science 889 (2021), 145–170. doi:10.1016/j.tcs.2021.08.002

[19] The Economist. 2021. Tracking covid-19 excess deaths across countries. https://www.economist.com/graphic-detail/coronavirus-excess-deaths-tracker.

[20] Dario Fiore, Aikaterini Mitrokotsa, Luca Nizzardo, and Elena Pagnin. 2016. Multi-key Homomorphic Authenticators. In ASIACRYPT 2016, Part II (LNCS, Vol. 10032), Jung Hee Cheon and Tsuyoshi Takagi (Eds.). Springer, Berlin, Heidelberg, 499–530. doi:10.1007/978-3-662-53890-6_17

[21] Dario Fiore and Anca Nitulescu. 2016. On the (In)Security of SNARKs in the Presence of Oracles. In TCC 2016-B, Part I (LNCS, Vol. 9985), Martin Hirt and

Adam D. Smith (Eds.). Springer, Berlin, Heidelberg, 108–138. doi:10.1007/978-3-662-53641-4_5

[22] Dario Fiore and Elena Pagnin. 2018. Matrioska: A Compiler for Multi-key Homomorphic Signatures. In SCN 18 (LNCS, Vol. 11035), Dario Catalano and Roberto De Prisco (Eds.). Springer, Cham, 43–62. doi:10.1007/978-3-319-98113-0_3

[23] Dario Fiore and Ida Tucker. 2022. Efficient Zero-Knowledge Proofs on Signed Data with Applications to Verifiable Computation on Data Streams. In ACM CCS 2022, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, 1067–1080. doi:10.1145/3548606.3560630

[24] Robert P Gallant, Robert J Lambert, and Scott A Vanstone. 2001. Faster point multiplication on elliptic curves with efficient endomorphisms. In Annual International Cryptology Conference. Springer, 190–200.

[25] Romain Gay and Bogdan Ursu. 2024. On Instantiating Unleveled Fully-Homomorphic Signatures from Falsifiable Assumptions. In PKC 2024, Part I (LNCS, Vol. 14601), Qiang Tang and Vanessa Teague (Eds.). Springer, Cham, 74–104. doi:10.1007/978-3-031-57718-5_3

[26] Rosario Gennaro and Daniel Wichs. 2013. Fully Homomorphic Message Authenticators. In ASIACRYPT 2013, Part II (LNCS, Vol. 8270), Kazue Sako and Palash Sarkar (Eds.). Springer, Berlin, Heidelberg, 301–320. doi:10.1007/978-3-642-42045-0_16

[27] Ashrujit Ghoshal and Stefano Tessaro. 2021. Tight State-Restoration Soundness in the Algebraic Group Model. In CRYPTO 2021, Part III (LNCS, Vol. 12827), Tal Malkin and Chris Peikert (Eds.). Springer, Cham, Virtual Event, 64–93. doi:10.1007/978-3-030-84252-9_3

[28] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. 2015. Leveled Fully Homomorphic Signatures from Standard Lattices. In 47th ACM STOC, Rocco A. Servedio and Ronitt Rubinfeld (Eds.). ACM Press, 469–477. doi:10.1145/2746539.2746576

[29] Chloé Hébant, Duong Hieu Phan, and David Pointcheval. 2020. Linearly-homomorphic signatures and scalable mix-nets. In IACR International Conference on Public-Key Cryptography. Springer, 597–627.

[30] Joseph Jaeger and Stefano Tessaro. 2020. Expected-Time Cryptography: Generic Techniques and Applications to Concrete Soundness. In TCC 2020, Part III (LNCS, Vol. 12552), Rafael Pass and Krzysztof Pietrzak (Eds.). Springer, Cham, 414–443. doi:10.1007/978-3-030-64381-2_15

[31] Robert Johnson, David Molnar, Dawn Song, and David Wagner. 2002. Homomorphic signature schemes. In Cryptographers' Track at the RSA Conference. Springer, 244–262.

[32] Russell W. F. Lai, Raymond K. H. Tai, Harry W. H. Wong, and Sherman S. M. Chow. 2018. Multi-key Homomorphic Signatures Unforgeable Under Insider Corruption. In ASIACRYPT 2018, Part II (LNCS, Vol. 11273), Thomas Peyrin and Steven Galbraith (Eds.). Springer, Cham, 465–492. doi:10.1007/978-3-030-03329-3_16

[33] Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. 2015. Linearly homomorphic structure-preserving signatures and their applications. Designs, Codes and Cryptography 77, 2 (2015), 441–477.

[34] Kåre Mølbak and A Mazick. 2013. European monitoring of excess mortality for public health action (EuroMOMO) Kåre Mølbak. European Journal of Public Health 23, suppl_1 (2013).

[35] Somayeh Dolatnezhad Samarin, Dario Fiore, Daniele Venturi, and Morteza Amini. 2021. A compiler for multi-key homomorphic signatures for Turing machines. Theoretical Computer Science 889 (2021), 145–170.

[36] Lucas Schabhüser, Johannes A. Buchmann, and Patrick Struck. 2017. A Linearly Homomorphic Signature Scheme from Weaker Assumptions. In 16th IMA International Conference on Cryptography and Coding (LNCS, Vol. 10655), Máire O'Neill (Ed.). Springer, Cham, 261–279. doi:10.1007/978-3-319-71045-7_14

[37] Lucas Schabhüser, Denis Butin, and Johannes Buchmann. 2019. Context Hiding Multi-key Linearly Homomorphic Authenticators. In CT-RSA 2019 (LNCS, Vol. 11405), Mitsuru Matsui (Ed.). Springer, Cham, 493–513. doi:10.1007/978-3-030-12612-4_25

A Non interactive inner product argument, InPrd

In Figure 5 we present the non interactive inner product argument (InPrd)obtained by the Fiat-Shamir transform applied to the interactive inner product argument given in Protocol 1&2 in [13], where we ignore all steps highlighted by underlining and purple text as they are related to the parallel version of the inner product described later.

ParInPrd.P$(\mathsf{pp} = (n, \mathbf{g}, \underline{\tilde{\mathbf{g}}}, \mathbf{h}, u), P, \underline{\tilde{P}}; \mathbf{a}, \mathbf{b}) \to \pi$

$\mathbf{g}^{(0)} \leftarrow \mathbf{g},\ \mathbf{h}^{(0)} \leftarrow \mathbf{h}\ \mathbf{a}^{(0)} \leftarrow \mathbf{a},\ \mathbf{b}^{(0)} \leftarrow \mathbf{b},\ m \leftarrow n$

$\underline{\tilde{\mathbf{g}}^{(0)} \leftarrow \tilde{\mathbf{g}}}$

for $i = 1, \dots, \log(n)$

  $m \leftarrow m/2$

  $c_L \leftarrow \langle \mathbf{a}^{(i-1)}_{[:m]}, \mathbf{b}^{(i-1)}_{[m:]} \rangle$

  $c_R \leftarrow \langle \mathbf{a}^{(i-1)}_{[m:]}, \mathbf{b}^{(i-1)}_{[:m]} \rangle$

  $L_i \leftarrow (\mathbf{g}^{(i-1)}_{[m:]})^{\mathbf{a}^{(i-1)}_{[:m]}} (\mathbf{h}^{(i-1)}_{[:m]})^{\mathbf{b}^{(i-1)}_{[m:]}} u^{c_L}$

  $\underline{\tilde{L}_i \leftarrow (\tilde{\mathbf{g}}^{(i-1)}_{[m:]})^{\mathbf{a}^{(i-1)}_{[:m]}} (\mathbf{h}^{(i-1)}_{[:m]})^{\mathbf{b}^{(i-1)}_{[m:]}} u^{c_L}}$

  $R_i \leftarrow (\mathbf{g}^{(i-1)}_{[:m]})^{\mathbf{a}^{(i-1)}_{[m:]}} (\mathbf{h}^{(i-1)}_{[m:]})^{\mathbf{h}^{(i-1)}_{[:m]}} u^{c_R}$

  $\underline{\tilde{R}_i \leftarrow (\tilde{\mathbf{g}}^{(i-1)}_{[:m]})^{\mathbf{a}^{(i-1)}_{[m:]}} (\mathbf{h}^{(i-1)}_{[m:]})^{\mathbf{h}^{(i-1)}_{[:m]}} u^{c_R}}$

  $x_i \leftarrow \mathcal{H}(\mathsf{pp}, P, \underline{\tilde{P}}, (L_j, R_j, \underline{\tilde{L}_j, \tilde{R}_j})_{j=1}^{i}, (x_j)_{j=1}^{i-1})$

  $\mathbf{g}^{(i)} \leftarrow (\mathbf{g}^{(i-1)}_{[:m]})^{x_i^{-1}} \circ (\mathbf{g}^{(i-1)}_{[m:]})^{x_i}$

  $\underline{\tilde{\mathbf{g}}^{(i)} \leftarrow (\tilde{\mathbf{g}}^{(i-1)}_{[:m]})^{x_i^{-1}} \circ (\tilde{\mathbf{g}}^{(i-1)}_{[m:]})^{x_i}}$

  $\mathbf{h}^{(i)} \leftarrow (\mathbf{h}^{(i-1)}_{[:m]})^{x_i} \circ (\mathbf{h}^{(i-1)}_{[m:]})^{x_i^{-1}}$

  $\mathbf{a}^{(i)} \leftarrow \mathbf{a}^{(i-1)}_{[:m]} \cdot x_i + \mathbf{a}^{(i-1)}_{[m:]} \cdot x_i^{-1}$

  $\mathbf{b}^{(i)} \leftarrow \mathbf{b}^{(i-1)}_{[:m]} \cdot x_i^{-1} + \mathbf{b}^{(i-1)}_{[m:]} \cdot x_i$

$a \leftarrow a^{\log(n)}, b \leftarrow b^{\log(n)}$

$\pi \leftarrow \left(\{(L_i, \underline{\tilde{L}_i}, R_i, \underline{\tilde{R}_i})\}_{i=1,\dots,\log(n)}, (a, b) \right)$

ParInPrd.V$(\mathsf{pp} = (n, \mathbf{g}, \underline{\tilde{\mathbf{g}}}, \mathbf{h}, u), P, \underline{\tilde{P}}, \pi) \to \mathsf{b}'$

parse π

$\mathbf{g}^{(0)} \leftarrow \mathbf{g},\ \underline{\tilde{\mathbf{g}}^{(0)} \leftarrow \tilde{\mathbf{g}}}, \mathbf{h}^{(0)} \leftarrow \mathbf{h},$

$P^{(0)} \leftarrow P,\ \underline{\tilde{P}^{(0)} \leftarrow \tilde{P}}, m \leftarrow n$

for $i = 1, \dots, \log(n)$

  $m \leftarrow m/2$

  $x_i \leftarrow \mathcal{H}(\mathsf{pp}, P, \underline{\tilde{P}}, (L_j, R_j, \underline{\tilde{L}_j, \tilde{R}_j})_{j=1}^{i}, (x_j)_{j=1}^{i-1})$

  $P^{(i)} \leftarrow L_i^{x_i^2} \cdot P^{(i-1)} \cdot R_i^{x_i^{-2}}$

  $\underline{\tilde{P}^{(i)} \leftarrow \tilde{L}_i^{x_i^2} \cdot \tilde{P}^{(i-1)} \cdot \tilde{R}_i^{x_i^{-2}}}$

  $\mathbf{g}^{(i)} \leftarrow (\mathbf{g}^{(i-1)}_{[:m]})^{x_i^{-1}} \circ (\mathbf{g}^{(i-1)}_{[m:]})^{x_i}$

  $\mathbf{h}^{(i)} \leftarrow (\mathbf{h}^{(i-1)}_{[m:]})^{x_i} \circ (\mathbf{h}^{(i-1)}_{[:m]})^{x_i^{-1}}$

  $\underline{\tilde{\mathbf{g}}^{(i)} \leftarrow (\tilde{\mathbf{g}}^{(i-1)}_{[:m]})^{x_i^{-1}} \circ (\tilde{\mathbf{g}}^{(i-1)}_{[m:]})^{x_i}}$

$g \leftarrow \mathbf{g}^{(\log(n))}, \underline{\tilde{g} \leftarrow \tilde{\mathbf{g}}^{(\log(n))}}, h \leftarrow \mathbf{h}^{(\log(n))}$

$\mathsf{b}' \leftarrow (P^{(\log(n))} == g^a h^b u^{ab})$

$\underline{\mathsf{b}' \leftarrow \mathsf{b}' \wedge (\tilde{P}^{(\log(n))} == \tilde{g}^a h^b u^{ab})}$

Figure 5: The InPrd protocol, a non interactive inner product argument obtained by applying Fiat-Shamir transform to the inner product argument in Protocol 2 in [13] using a random oracle. $\mathcal{H}$ is a domain separating hash function that takes two group elements in $\mathbb{G}_1$ as input and outputs a value in $\mathbb{Z}_p$. In purple we highlighted the components needed for the parallel version defined in Appendix C.

$\underline{\text{Game } \mathsf{IP}-\mathsf{EXT}^{\mathcal{P}_{\mathsf{alg}},\mathcal{E}}_{\mathsf{IP},\mathsf{R}^{\mathsf{IP}}}(\lambda) \to \mathsf{b}'}$

1 : $\mathsf{pp} := (n, \mathbf{g}, \mathbf{h}, u) \xleftarrow{\$} \mathsf{IP.setup}(1^\lambda); H \xleftarrow{\$} \Omega_{\mathsf{hLen}(\lambda)}$
2 : $\mathbf{f} \xleftarrow{\$} \mathbb{G}^m$;
3 : $([\pi], [P], \mathbf{x}) \xleftarrow{\$} \mathcal{P}^H_{\mathsf{alg},\lambda}(\mathsf{pp}, \mathbf{f})$
4 : **parse** $\pi := ((L_i, R_i,)_{i=1,\dots,\log(n)}, a, b)$
5 : **parse** $\mathbf{x} := (x_1, \dots, x_{\log(n)})$
6 : **for** $i = 1, \dots, \log n$
7 : $\quad \mathsf{b'}_i \leftarrow \left(x_i == H(\mathsf{pp}, P, (L_j, R_j, x_j)_{j\in[i-1]}, L_i, R_i)\right)$
8 : $\mathsf{b}' \leftarrow (\mathsf{IP.V}(\mathsf{pp}, P, \pi) = 1) \wedge_{i=1}^{\log n} \mathsf{b'}_i$
9 : $w \xleftarrow{\$} \mathcal{E}(1^\lambda, \mathsf{pp}, [P], [\pi])$;
10 : $\mathsf{b}' \leftarrow (\mathsf{b}' \wedge (\pi, w) \notin \mathsf{R}^{\mathsf{IP}})$

Figure 6: Definition of the ip − ext **security game in AGM. The game** IP − EXT **defines** ip − ext **secuirty in the AGM for a non-uniform algebraic adversary** $\mathcal{P}_{\mathsf{alg}}$**, an extractor** $\mathcal{E}$ **and a non interactive inner product argument** InPrd**. The length of the challenges are** $\mathsf{hLen}(\lambda)$ **and the set** $\Omega_{\mathsf{hLen}(\lambda)}$ **contain all the functions mapping from** $\{0,1\}^*$ **to** $\{0,1\}^{\mathsf{hLen}}$.

A.1 Security of InPrd in the AGM

Ghoshal and Tessaro [27], prove security of the interactive Bulletproof protocol in the AGM and outline the proof of the non interactive version. Our proof builds on their result but we prove security in a slightly different setting:

(1) We focus only on the inner product argument, not the full Bulletproof protocol. This allows us to simplify the proof, due to the reduced complexity of the protocol.
(2) We adapt the proof to a setting where the adversary has access to additional uniformly sampled group elements besides the public parameters of the inner product argument. This adaptation is highlighted in red throughout this section.
(3) We consider a non interactive version of the protocol, obtained by the Fiat-Shamir transform.

Analogously to their proof, we define soundness of the non interactive protocol by a witness extraction game, IP − EXT (Figure 6), which is slightly different from the original due to the reasoning above.

In Figure 6 we define the game $\mathsf{IP}-\mathsf{EXT}^{\mathcal{P}_{\mathsf{alg}},\mathcal{E}}_{\mathsf{R}^{\mathsf{InPrd}}}(\lambda)$. An adversary $\mathcal{P}_{\mathsf{alg}}$ wins it if they manage to output a proof whose transcript $[\tau]$ is accepted by the verifier, but from which no extractor $\mathcal{E}$ can extract a witness w, that is, $[\tau] \in \mathcal{T}_{\mathsf{Acc}} \backslash \mathcal{T}^{\mathsf{e},\mathsf{R}^{\mathsf{InPrd}}}_{\mathsf{Correct}}$ where $\mathcal{T}_{\mathsf{Acc}} := \{[\tau] \in \mathcal{T} \ : \ \mathsf{InPrd.V}(\mathsf{pp}, [P], [\pi]) = 1\}$, and

$$\mathcal{T}^{\mathsf{e},\mathsf{R}^{\mathsf{InPrd}}}_{\mathsf{Correct}} := \{[\tau] \in \mathcal{T}_{\mathsf{Acc}} \ : \ w \leftarrow \mathcal{E}([\pi], [P]), (\pi, w) \in \mathsf{R}^{\mathsf{InPrd}}\}.$$

Observe that a transcript $\tau := (\mathsf{pp}, \pi, P, \mathbf{x})$ includes the following: pp is the public parameters, π is an inner product proof, P is the commitment, and $\mathbf{x}$ is the challenges used in the verification of the proof. In this section we will make intermediate use of τ and π, where in such case π specifically denoted the InPrd proof included in τ.

For the inner product argument InPrd, the relation $\mathsf{R}^{\mathsf{InPrd}}$, an algebraic prover $\mathcal{P}_{\mathsf{alg}}$ and an extractor $\mathcal{E}$ we define the advantage of $\mathcal{P}_{\mathsf{alg}}$ to output a valid proof for which $\mathcal{E}$ cannot extract a valid witness as,

$$\mathsf{Adv}^{\mathsf{ip-ext}}_{\mathsf{InPrd},\mathsf{R}^{\mathsf{InPrd}}}(\mathcal{P}_{\mathsf{alg}}, \mathcal{E}, \lambda) = \Pr[\mathsf{IP}-\mathsf{EXT}^{\mathcal{P}_{\mathsf{alg}},\mathcal{E}}_{\mathsf{InPrd},\mathsf{R}^{\mathsf{InPrd}}}(\lambda)],$$

This is advantage is bounded in Theorem A.1.

Theorem A.1. *Let* $\mathbb{G} = \{\mathbb{G}_\lambda\}_{\lambda\in\mathbb{N}^+}$ *be a family of groups of order* $p = p(\lambda)$*. Let* $\mathsf{InPrd} = \mathsf{InPrd}[\mathbb{G}]$ *be the non interactive inner product argument in Figure 5. We can construct an extractor* $\mathcal{E}$*, such that for any non-uniform algebraic adversary* $\mathcal{P}_{\mathsf{alg}}$ *making at most* $q = q(\lambda)$ *queries to its random oracle, there exists a non-uniform adversary* $\mathcal{F}$ *with the property that for all* $\lambda \in \mathbb{N}^+$,

$$\mathsf{Adv}^{\mathsf{ip-ext}}_{\mathsf{InPrd},\mathsf{R}^{\mathsf{InPrd}}}(\mathcal{P}_{\mathsf{alg}}, \mathcal{E}, \lambda) \leq (14n + 8{+}4m)q/(p-1) + \mathscr{A}^{\mathsf{dl}}_{\mathbb{G}_1}(\mathcal{F}, \lambda) + 1/p,$$

where $n = |\mathbf{g}| = |\mathbf{h}|$, $m = |\mathbf{f}|$ *and* $2^{\mathsf{hLen}} = p - 1$.

Moreover, the time complexity of the extractor $\mathcal{E}$ *and adversary* $\mathcal{F}$ *is* $O(q \cdot (n + m))$.

A.2 Proof of Theorem A.1

The following notation will be used throughout the proof. A partial proof, $\pi^{(i)}$, is the proof up until just before the point where the ith challenge is computed, that is $\pi^{(i)} = ((L_1, R_1,), \dots, (L_i, R_i,))$ and the corresponding transcript is, $\tau^{(i)} = (\mathsf{pp}, P, \pi^{(i)}, x_1, \cdots, x_{i-1})$. For the algebraic representation of the group elements in the proof, $[\pi]$, and corresponding transcript, $[\tau]$, we use the following notation. For $[P], [L_k], [R_k]$ for $k = 1, \dots, \log n$, we denote the exponents of the elements g_j, h_j, u in their expression as $p^{g_j}, p^{h_j}, p^u, l_k^{g_j}, \dots, r_k^{g_j}, \dots$, that is,

$$P = \prod_{j-1}^{n} g_j^{p^{g_j}} h_j^{p^{h_j}} u^{p^u}, L_k = \prod_{j-1}^{n} g_j^{l_k^{g_j}} h_j^{l_k^{h_j}} u^{l_k^u} \text{ and}$$

$$R_k = \prod_{j-1}^{n} g_j^{r_k^{g_j}} h_j^{r_k^{h_j}} u^{r_k^u}.$$

The same notation is used also w.r.t. $\mathbf{f}$ when necessary.

Lemma A.2 states that the hardness of the non-trivial discrete relation problem in the group $\mathbb{G}$ is tightly implied by the hardness of the discrete logarithm problem on the same group.

Lemma A.2 ([30]). *Let* n *be a positive integer and* $\mathbb{G} \in \{\mathbb{G}_\lambda\}_{\lambda\in\mathbb{N}^+}$ *be a family of cyclic groups of prime order* p*. For every non-uniform adversary* $\mathscr{A} = \{\mathscr{A}_\lambda\}_{\lambda\in\mathbb{N}^+}$ *there exist a non-uniform adversary* $\mathscr{B} = \{\mathscr{B}_\lambda\}_{\lambda\in N^+}$ *such that for all* $\lambda \in \mathbb{N}^+$,

$$\mathsf{Adv}^{\mathsf{dl-rel}}_{\mathbb{G},n}(\mathscr{A}, \lambda) \leq \mathsf{Adv}^{\mathsf{dl}}_{\mathbb{G}}(\mathscr{B}, \lambda) + 1/p.$$

The runtime of $\mathscr{B}$ *is that of* $\mathscr{A}$ *plus the time to perform* $n + 1$ *multi-exponentiations of size* 2 *and some computations in the field* $\mathbb{Z}_p$.

Finally, we recall the Schwartz-Zippel lemma.

Lemma A.3 (Schwartz-Zippel). *Let* $\mathbb{F}$ *be a finite field and* $f \in \mathbb{F}[X_1, \dots, X_n]$ *a non-zero n-variate polynomial with maximum degree* d*. Let* S *be a subset of* $\mathbb{F}$*. Then* $\Pr[f(x_1, \dots, x_n) = 0] \leq d/|S|$,

where the probability is over the choice of $x_1, \ldots, x_n$ *according to* $x_i \xleftarrow{\$} S$.

Let $\mathbf{p}(\mathbf{X})$ be a vector of polynomials. We define $\mathsf{sz}(\mathbf{p}(\mathbf{X}), \mathbf{x})$ to be a function that returns true if and only if, $\mathbf{p}(\mathbf{X})$ is non-zero but $\mathbf{p}(\mathbf{x})$ evaluates to zero.

Given this notation we bound the advantage in two steps.

First, we introduce the concept of *bad challenges*, which essentially are challenges that would make extraction impossible, and show that it is unlikely that a transcript contains any such challenges. Let $\mathsf{BadCh}([\tau^{(i)}]) \subseteq \mathsf{Ch}$ to be the set of bad challenges and $\mathcal{T}_{\mathsf{BadCh}}$ be the set of all extended transcripts that contain at least one bad challenge. We will see that probability of generating a bad challenge for the extended partial transcripts $[\tau^{(i)}]$ is bounded by $\varepsilon \in [0, 1]$.

Second, we bound the probability that $\mathcal{E}$ fails to extract a valid witness given an accepting but incorrect transcript that does not contain any bad challenges, *i.e.*, for a transcript $[\tau]$ in the set

$$\mathcal{T}^{\mathsf{e},\mathsf{R}^{\mathsf{InPrd}}}_{\mathsf{fail}} := \mathcal{T}_{\mathsf{Acc}} \setminus \left(\mathcal{T}^{\mathsf{e},\mathsf{R}^{\mathsf{InPrd}}}_{\mathsf{Correct}} \cup \mathcal{T}_{\mathsf{BadCh}} \right).$$

We denote the probability that the extractor fails by p_{fail} and bound it by the advantage of solving the dl problem in $\mathbb{G}_1$.

A.2.1 Bounding ε. We define ε as an upper bound of $(|\mathsf{BadCh}([\tau^{(i)}])|/|\mathsf{Ch}|)$ for $i \in \{1, \ldots, \log n\}$. Where $[\tau^{(i)}]$ is a partial extended transcript, Ch is the space from challenges are sampled, by assumption $|\mathsf{Ch}| = 2^{\mathsf{hLen}}$. A challenge, x_i, is *bad* if and only if $\mathsf{CheckBad}([\tau^{(i)}], x_i) = 1$, where CheckBad[8] is as defined in Figure 7. One can ignore the details of CheckBad for now, as the rationale behind them will become clear later on. Observe though that whether a challenge is bad or not only depends on the extended transcript so far. Next, we define the set of all bad challenges,

$$\mathsf{BadCh}(\tau^{(i)}) := \{x_i \in \mathsf{Ch} \ : \ \mathsf{CheckBad}([\tau^{(i)}], x_i) = 1\}.$$

By the Schwartz-Zipple Lemma and a rewriting of Lemma 5 in [27] to our setting, with additionally m group elements, it follows that

$$|\mathsf{BadCh}(\tau^{(i)})|/|\mathsf{Ch}| \leq (14n + 8{+}4m)/(2^{\mathsf{hLen}}).$$

Finally, for a hash function that maps to the entire set $\mathbb{Z}_p$ it follows that $\varepsilon \leq (14n + 8{+}4m)/(p-1)$.

A.2.2 Bounding p_{fail}. To upper bound p_{fail} we construct an adversary $\mathcal{A}_{\mathsf{dl\text{-}rel}}$ against the non-trivial discrete relation problem that takes as input the independent group generators $\mathbf{g}, \mathbf{h}, u$ of the group $\mathbb{G}_1$ and acts as follows. The adversary $\mathcal{A}_{\mathsf{dl\text{-}rel}}$ simulates the game $\mathsf{IP} - \mathsf{EXT}$ to $\mathcal{P}_{\mathsf{alg}}$ using the public parameters $\mathsf{pp} = (\mathbf{g}, \mathbf{h}, u)$. If $\mathcal{P}_{\mathsf{alg}}$ manages to produce an accepting transcript $[\tau] \in \mathcal{T}^{\mathsf{e},\mathsf{R}^{\mathsf{InPrd}}}_{\mathsf{fail}}$, $\mathcal{A}_{\mathsf{dl\text{-}rel}}$ calls the function h on the input $[\tau]$ and outputs whatever h outputs as the answer to the non-trivial discrete relation challenge.

Claim: *For any accepting transcript* $[\tau] \notin \mathcal{T}^{\mathsf{InPrd}}_{\mathsf{BadCh}}$, *there exist two functions* e *and* h *such that : if* $\mathsf{e}([\tau])$ *does not return a valid witness then* $\mathsf{h}([\tau])$ *returns a non-trivial discrete relation between the group generators* $\mathbf{g}, \mathbf{h}, u$.

[8] CheckBad is equivalent to the last algorithm of Figure 8 in [27] with m additional degree 4 polynomials related to the group elements $\mathbf{f}$.

$\mathsf{CheckBad}([\tau^{(k)}]) \to \mathsf{b}'$

$\mathsf{parse}\ \tau^{(k)} := (\mathsf{pp}, [P], [\pi^{(k)}], \{x_i\}_{i\in[k-1]})$

$\mathsf{parse}\ \pi^{(k)} := \left(([L_i], [R_i],)_{i=1,\ldots,k}\right)$

$\mathbf{for}\ j = 0, \ldots, n-1$

$$f^{\mathbf{g}}_{k,j}(X) \leftarrow l_k^{g_{1+j}} \cdot X^2 + r_k^{g_{1+j}} \cdot X^{-2} + p^{g_{1+j}} + \sum_{i=1}^{k-1} (l_i^{g_{1+j}} \cdot x_i^2 + r_i^{g_{1+j}} \cdot x_i^{-2})$$

$$f^{\mathbf{h}}_{k,j}(X) \leftarrow l_k^{h_{1+j}} \cdot X^2 + r_k^{h_{1+j}} \cdot X^{-2} + p^{h_{1+j}} + \sum_{i=1}^{k-1} (l_i^{h_{1+j}} \cdot x_i^2 + r_i^{h_{1+j}} \cdot x_i^{-2})$$

$\mathbf{for}\ j = 0, \ldots, m-1$

$$f^{\mathbf{f}}_{k,j}(X) \leftarrow l_k^{f_{1+j}} \cdot X^2 + r_k^{f_{1+j}} \cdot X^{-2} + p^{f_{1+j}} + \sum_{i=1}^{k-1} (l_i^{f_{1+j}} \cdot x_i^2 + r_i^{f_{1+j}} \cdot x_i^{-2})$$

$$f^{u}_{k}(X) \leftarrow l_k^{u} \cdot X^2 + r_k^{u} \cdot X^{-2} + p'_u + \sum_{i=1}^{k-1} (l_i^{u} x_i^2 + r_i^{u} x_i^{-2})$$

$\mathsf{b}' \leftarrow \mathit{false}$

$\mathbf{for}\ t = 1, \ldots, k-1$

$\quad \mathbf{for}\ j = 0, \ldots, n/2^t - 1$

$$\mathsf{b}' \leftarrow \mathsf{b}' \vee \mathsf{sz}\left(f^{\mathbf{g}}_{k,j}(X) \cdot x_t^2 - f^{\mathbf{g}}_{k,j+n/2^t}(X), x_k\right) \vee \mathsf{sz}\left(f^{\mathbf{h}}_{k,j}(X) - f^{\mathbf{h}}_{k,j+n/2^t}(X) \cdot x_t^2, x_k\right)$$

$\mathbf{for}\ j = 0, \ldots, n/2^k - 1$

$$\mathsf{b}' \leftarrow \mathsf{b}' \vee \mathsf{sz}\left(f^{\mathbf{g}}_{k,j}(X) \cdot X^2 - f^{\mathbf{g}}_{k,j+n/2^m}(X), x_k\right) \vee \mathsf{sz}\left(f^{\mathbf{h}}_{k,j}(X) - f^{\mathbf{h}}_{k,j+n/2^m}(X) \cdot X^2, x_k\right)$$

$$\mathsf{b}' \leftarrow \mathsf{b}' \vee \mathsf{sz}\left(f^{u}_{k}(X) - \sum_{j=0}^{n/2^k-1} f^{\mathbf{g}}_{k,j}(X) \cdot f^{\mathbf{h}}_{k,j}(X), x_k\right)$$

$\mathbf{for}\ j = 1, \ldots, m-1$

$$\mathsf{b}' \leftarrow \mathsf{b}' \vee \mathsf{sz}\left(f^{\mathbf{f}}_{k,j}(X), x_k\right)$$

Figure 7: The function CheckBad **for the** InPrd**. The function returns true if the challenge x_k is a *bad challenge* otherwise false.**

We prove the claim in subsubsection A.2.3.

By the claim, $\mathcal{A}_{\mathsf{dl\text{-}rel}}$ succeeds whenever $[\tau] \in \mathcal{T}^{\mathsf{e},\mathsf{R}^{\mathsf{InPrd}}}_{\mathsf{fail}}$ and consequently, $p_{\mathsf{fail}} \leq \mathsf{Adv}^{\mathsf{dl\text{-}rel}}_{\mathbb{G}_1, 2n+1}(\mathcal{A}_{\mathsf{dl\text{-}rel}})$. By Lemma Theorem A.2 we have that $p_{\mathsf{fail}} \leq \mathsf{Adv}^{\mathsf{dl}}_{\mathbb{G}_1}(\mathcal{A}_{\mathsf{dl}}) + 1/p$ where $\mathcal{A}_{\mathsf{dl}}$ is almost as efficient as $\mathcal{A}_{\mathsf{dl\text{-}rel}}$.

A.2.3 Proof of claim. To prove the above claim, we instantiate the two functions e and h and show that they satisfy the above claim. Let $\mathsf{e}([\tau])$ and $h([\tau])$ be as in Figure 8.Let $(e_{\mathbf{g}}, e_{\mathbf{h}}, e_{\mathbf{f}}, e_u) \leftarrow \mathsf{h}([\tau])$, if the returned values are not all zeros then $P^{(\log n)}, g^{(\log n)}, h^{(\log n)}, a, b, u$ can be expressed in terms of $\mathbf{g}, \mathbf{h}, \mathbf{f}, u$ such that $\mathbf{g}^{e_{\mathbf{g}}} \mathbf{h}^{e_{\mathbf{h}}}, \mathbf{f}^{e_{\mathbf{f}}} u^{e_u} = 1$.

$\mathsf{e}([\tau]) \to (\mathbf{a}', \mathbf{b}')$

/ $\tau = \big(\mathsf{pp}, [P], \pi = (\dots, (x_1, \dots, x_{\log(n)}))\big), [P] = (P, p^{\mathbf{g}}, p^{\mathbf{h}} p^{\mathbf{f}}, p^u)$

1 : $(\mathbf{a}' \leftarrow p^{\mathbf{g}}, \mathbf{b}' \leftarrow p^{\mathbf{h}})$

$\mathsf{h}([\tau]) \to (e_{\mathbf{g}}, e_{\mathbf{h}}, e_{\mathbf{f}}, e_u)$

/ $\tau = \big(\mathsf{pp}, [P], \pi, (x_1, \dots, x_{\log(n)})\big)$

/ $\pi = (([L_1], [R_1]) \dots, ([L_{\log(n)}], [R_{\log(n)}]), (a, b))$

1 : **for** $j = 0, \dots, n-1$

2 : $e^{g_{1+j}} \leftarrow p^{g_{1+j}} + \sum_{i=1}^{\log(n)} l_i^{g_{1+j}} x_i^2 + r_i^{g_{1+j}} x_i^{-2} - a \cdot \prod_{i=1}^{\log(n)} x_i^{(-1)^{1-\mathsf{bit}(j,i,\log(n))}}$

3 : $e^{h_{1+j}} \leftarrow p^{h_{1+j}} + \sum_{i=1}^{\log(n)} l_i^{h_{1+j}} x_i^2 + r_i^{h_{1+j}} x_i^{-2} - b \cdot \prod_{i=1}^{\log(n)} x_i^{(-1)^{\mathsf{bit}(j,i,\log(n))}}$

4 : **for** $j = 0, \dots, m-1$

5 : $e^{f_{1+j}} \leftarrow p^{f_{1+j}} + \sum_{i=1}^{\log(n)} l_i^{f_{1+j}} x_i^2 + r_i^{f_{1+j}} x_i^{-2}$

6 : $e_u \leftarrow p'_u + \sum_{i=1}^{\log(n)} l_i^u x_i^2 + r_i^u x_i^{-2} - ab$

7 : $e_{\mathbf{g}} \leftarrow (e^{g_1}, \dots, e^{g_n})$

8 : $e_{\mathbf{h}} \leftarrow (e^{h_1}, \dots, e^{h_n})$

9 : $e_{\mathbf{f}} \leftarrow (e^{f_1}, \dots, e^{f_m})$

Figure 8: The function e **and** h **for** InPrd.

Given e and h as defined above, the claim follow from the below lemma.

Lemma A.4. *Let $[\tau]$ be an extended accepting transcript of* InPrd *such that $[\tau] \notin \mathcal{T}^{\mathsf{InPrd}}_{\mathsf{BadCh}}$. If $\mathsf{e}([\tau])$ returns $(\mathbf{a}', \mathbf{b}')$ such that one of the following does not hold: $P = \mathbf{g}^{\mathbf{a}'} \mathbf{h}^{\mathbf{b}'} u^{c'}$ or $\langle \mathbf{a}', \mathbf{b}' \rangle = c'$.' then $h([\tau])$ returns a non-trivial discrete logarithm relation.*

Proof. We will prove the contrapositive statement: assume $h([\tau])$ returns the trivial discrete logarithm relation, then $g^{\mathbf{a}'} h^{\mathbf{b}'} = P$ and $\langle \mathbf{a}, \mathbf{b} \rangle = c$.

Since $h[\tau]$ returns the trivial discrete logarithm relation we have that $(e_{\mathbf{g}}, e_{\mathbf{h}}, e_{\mathbf{f}}, e_u) = (\mathbf{0}^n, \mathbf{0}^n, \mathbf{0}^n, 0)$. Furthermore, since $\tau \notin \mathcal{T}^{\mathsf{InPrd}}_{\mathsf{BadCh}}$ it follows that $x_i \notin \mathsf{BadCh}(\tau|_i)$ for all challenges x_i in the protocol. From $e_{\mathbf{g}} = \mathbf{0}^n$ it follows that for all $j \in \{0, \dots, n-1\}$

$$p^{g_{1+j}} + \sum_{i=1}^{\log n} l_i^{g_{1+j}} x_i^2 + r_i^{g_{1+j}} x_i^{-2} - a \cdot \prod_{i=1}^{\log n} x_i^{(-1)^{1-\mathsf{bit}(j,i,\log n)}} = 0$$

From $e_{\mathbf{h}} = \mathbf{0}^n$ it follows that for all $j \in \{0, \dots, n-1\}$

$$p^{h_{1+j}} + \sum_{i=1}^{\log n} l_i^{h_{1+j}} x_i^2 + r_i^{h_{1+j}} x_i^{-2} - b \cdot \prod_{i=1}^{\log n} x_i^{(-1)^{\mathsf{bit}(j,i,\log n)}} = 0$$

From $e_u = 0$ it follows that

$$p'_u + \sum_{i=1}^{\log n} l_i^u x_i^2 + r_i^u x_i^{-2} - ab = 0$$

With regard to some notational differences the proof follows from Lemma 7 & 8 in [27] □

B Reduction from $\widetilde{\mathsf{InPrd}}$ to InPrd

In what follows we show that our simplified inner product argument $\widetilde{\mathsf{InPrd}}$ reduces so the original inner product argument InPrd from Bulletproofs [13] by showing that any successful forgery against $\widetilde{\mathsf{InPrd}}$ can be transformed into a successful forgery against InPrd.

Lemma B.1. *The simplified inner produce argument $\widetilde{\mathsf{InPrd}}$ in Figure 3 reduces to* InPrd *in Figure 5. Formally, for any security parameter λ, and for any algebraic adversary $\widetilde{\mathcal{P}}_{\mathsf{alg}}$, there exist an algebraic prover $\mathcal{P}_{\mathsf{alg}}$ such that:*

$$\mathsf{Adv}^{\mathsf{ip-ext}}_{\widetilde{\mathsf{InPrd}}, \mathsf{R}^{\widetilde{\mathsf{InPrd}}}}(\widetilde{\mathcal{P}}_{\mathsf{alg}}, \widetilde{\mathcal{E}}, \lambda) \leq \mathsf{Adv}^{\mathsf{ip-ext}}_{\mathsf{InPrd}, \mathsf{R}^{\mathsf{InPrd}}}(\mathcal{P}_{\mathsf{alg}}, \mathcal{E}, \lambda)$$

where $\mathsf{R}^{\widetilde{\mathsf{InPrd}}}, \mathsf{R}^{\mathsf{InPrd}}, \widetilde{\mathcal{E}}, \mathcal{E}$ are specified by $\widetilde{\mathsf{InPrd}}$, and InPrd.

Proof. The proof is in the random oracle model, and builds a reduction $\mathscr{B}$ between $\mathsf{IP} - \mathsf{EXT}^{\mathcal{P}_{\mathsf{alg}}, \mathcal{E}}_{\mathsf{InPrd}, \mathsf{R}^{\mathsf{InPrd}}}(\lambda)$, i.e., an InPrd challenger, and an $\widetilde{\mathsf{InPrd}}$ forger $\widetilde{\mathcal{P}}_{\mathsf{alg}}$. The reduction takes in input the public parameters of InPrd, that is $\mathsf{pp} = (n, \mathbf{g}, \mathbf{h}, u)$; has access to the random oracle $\mathcal{RO}$; and is expected to output a proof π^* that is accepted by the security game $\mathsf{IP} - \mathsf{EXT}$ in Figure 6.

The reduction acts as follows. First, $\mathscr{B}$ simulates the public parameters of $\widetilde{\mathsf{InPrd}}$ by sending $\widetilde{\mathsf{pp}} = (n, \mathbf{g}, u)$; that is pp without the $\mathbf{h}$ component. Second, $\mathscr{B}$ simulates the interaction with the random oracle using the $Sim - Oracle^{\mathcal{RO}}$ algorithm described in Figure 9. Specifically, the reduction bakes in the input query the $\mathbf{h}$ components, so that it perfectly simulates a query to $\mathcal{RO}$ for InPrd, and returns $\mathcal{RO}$'s answer. So far the simulation is indistinguishable apart from $\widetilde{\mathcal{P}}_{\mathsf{alg}}$ guessing the vector $\mathbf{h}$, which happens with negligible probability $(Q/p)^n$ (where Q denotes the number of queries to $Sim - Oracle^{\mathcal{RO}}$). Finally, let $\tilde{\pi}$ denote $\widetilde{\mathcal{P}}_{\mathsf{alg}}$'s output at the end of its interaction with $\mathscr{B}$. Since we assume $\widetilde{\mathcal{P}}_{\mathsf{alg}}$ is a forger, $\tilde{\pi}$ should be such that $\widetilde{\mathsf{InPrd}}.\mathsf{V}(\widetilde{\mathsf{pp}}, \widetilde{P}, \tilde{\pi}) = 1$, extraction is successful $\widetilde{w} \leftarrow \widetilde{\mathcal{E}}(1^\lambda, \widetilde{\mathsf{pp}}, \widetilde{P}, \tilde{\pi})$, but $(\widetilde{\pi}, \widetilde{w})$ is not in the relation. Then, $\mathscr{B}$ runs the $\pi^* \leftarrow \mathsf{ReduceIP}(\mathsf{pp}, \tilde{\pi})$ algorithm Figure 9, which essentially bakes in the $\mathbf{h}$ contribution into $\widetilde{\pi}$, and returns π^* to its challenger. Note that on input $\widetilde{P} \neq \mathbf{g}^{\mathbf{a}} u^{\langle \mathbf{a}, \mathbf{1} \rangle}$, ReduceIP returns $P = \widetilde{P} \mathbf{h}^{\mathbf{1}} \neq \mathbf{g}^{\mathbf{a}} \mathbf{h}^{\mathbf{1}} u^{\langle \mathbf{a}, \mathbf{1} \rangle}$

We are left to show that π^* is accepted by $\mathsf{IP} - \mathsf{EXT}$. This boils down to showing that $\mathsf{InPrd}.\mathsf{V}(\mathsf{pp}, P^*, \pi^*) = 1$ and that $(\pi^*, w^*) \notin \mathsf{R}^{\mathsf{InPrd}}$, with $w^* \leftarrow \mathcal{E}(1^\lambda, \mathsf{pp}, P^*, \pi^*)$. By construction, it is clear that if $\widetilde{\mathsf{InPrd}}.\mathsf{V}(\widetilde{\mathsf{pp}}, \widetilde{P}, \tilde{\pi}) = 1$, then $\mathsf{InPrd}.\mathsf{V}(\mathsf{pp}, P, \pi^*) = 1$ (compare

$Sim - Oracle^{\mathcal{RO}}_{\mathsf{pp}}(\widetilde{P}, \{\widetilde{L}_i, \widetilde{R}_i\}_{i=1}^k)$

if $((\mathsf{pp}, \widetilde{P}, \{\widetilde{L}_i, \widetilde{R}_i\}_{i=1}^k), x_k) \in L_{\mathcal{H}}$:
  return x
$\mathbf{h}^{(0)} \leftarrow h_{[:n]}$, $\mathbf{b}^{(0)} \leftarrow \mathbf{1}^n$, $P \leftarrow \tilde{P} \cdot \mathbf{h}^{(0)}$
for $i \in [1, k]$:
  if $((\mathsf{pp}, \widetilde{P}, \{\widetilde{L}_j, \widetilde{R}_j\}_{j=1}^i), \cdot) \in L_{\mathcal{H}}$:
    Retrieve x_i
  else :
    $n \leftarrow n/2$
    $\mathbf{h}^{(i)} = (\mathbf{h}_{[:n]}^{(i-1)})^{\mathbf{b}_{[n:]}^{(i-1)}} \circ (\mathbf{h}_{[n:]}^{(i-1)})^{\mathbf{b}_{[:n]}^{(i-1)}}$
    $L_i = \widetilde{L}_i \cdot \left(\mathbf{h}_{[n:]}^{(i-1)}\right)^{\mathbf{b}_{[:n]}^{(i-1)}}$
    $R_i = \widetilde{R}_i \cdot \left(\mathbf{h}_{[:n]}^{(i-1)}\right)^{\mathbf{b}_{[n:]}^{(i-1)}}$
    $x_i \leftarrow \mathcal{RO}(\mathsf{pp}, P, \{L_j, R_j, x_j\}_{j\in[i-1]}, L_i, R_i)$
    $L_{\mathcal{H}} \leftarrow L_{\mathcal{H}} \cup \left((\mathsf{pp}, \widetilde{P}, \{\widetilde{L}_j, \widetilde{R}_j\}_{j=1}^i), x_i\right)$
  $\mathbf{b}^{(i)} = \mathbf{b}_{[:n]}^{(i-1)} \cdot x_i^{-1} + \mathbf{b}_{[n:]}^{(i-1)} \cdot x_i$
return x_k

$\mathsf{ReduceIP}(\mathsf{pp}, \widetilde{P}, \tilde{\pi}) \to P, \pi$

parse $\mathsf{pp} := (n, \mathbf{g}, \mathbf{h}, u)$
parse $\tilde{\pi} := ((\widetilde{L}_1, \widetilde{R}_1), \ldots, (\widetilde{L_{\log n}}, \widetilde{R_{\log n}}), a)$
$\mathbf{h}^{(0)} := \mathbf{h}_{[:n]}$, $\mathbf{b}^{(0)} \leftarrow \mathbf{1}^n, m \leftarrow n, P \leftarrow \widetilde{P} \cdot \mathbf{h}^{(0)}$
for $i \in [1, \log n]$:
  $m \leftarrow m/2$
  $\mathbf{h}^{(i)} \leftarrow (\mathbf{h}_{[:m]}^{(i-1)})^{\mathbf{b}_{[m:]}^{(i-1)}} \circ (\mathbf{h}_{[m:]}^{(i-1)})^{\mathbf{b}_{[:m]}^{(i-1)}}$
  $L_i \leftarrow \widetilde{L}_i \cdot \left(\mathbf{h}_{[m:]}^{(i-1)}\right)^{\mathbf{b}_{[:m]}^{(i-1)}}$
  $R_i \leftarrow \widetilde{R}_i \cdot \left(\mathbf{h}_{[:m]}^{(i-1)}\right)^{\mathbf{b}_{[m:]}^{(i-1)}}$
  $x_i \leftarrow \mathcal{RO}(\mathsf{pp}, P, \{L_j, R_j, x_j\}_{j\in[i-1]}, L_i, R_i)$
  $\mathbf{b}^{(i)} = \mathbf{b}_{[:m]}^{(i-1)} \cdot x_i + \mathbf{b}_{[m:]}^{(i-1)} \cdot x_i^{-1}$
$b \leftarrow \mathbf{b}^{(\log n)}$
$\pi \leftarrow ((L_1, R_1), \ldots, (L_{\log n}, R_{\log n}), (a, b))$

Figure 9: First procedure, description of the programmable random oracle used in Appendix B. Second procedure, reduction from a proof, $\tilde{\pi}$, of the protocol $\widetilde{\mathsf{InPrd}}$ to a proof, π, of the protocol InPrd.

Figure 5, and ReduceIP in Figure 9). It also holds that if $(\tilde{\pi}, \tilde{w})$ is not in $\mathsf{R}^{\widetilde{\mathsf{InPrd}}}$, then (π^*, w^*) is also not in $\mathsf{R}^{\mathsf{InPrd}}$. □

C Parallel inner product, ParInPrd

In this section we define ParInPrd, an parallel version of InPrd and prove that the security of ParInPrd follows from the security of InPrd. This stronger version of our inner product argument that proves that the same witness $\mathbf{a}$ satisfies two different equations at the same time:

$$\mathsf{R}^{\mathsf{ParInPrd}} = \\ = \left\{ (\mathbf{g}, \underline{\tilde{\mathbf{g}}}, \mathbf{h} \in \mathbb{G}_1^n,\ P, \underline{\tilde{P}} \in \mathbb{G}_1,\ c \in \mathbb{Z}_p ; \mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n) : \right. \\ \left. P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}} \wedge \underline{\tilde{P} = \tilde{\mathbf{g}}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}}} \wedge \langle \mathbf{a}, \mathbf{b} \rangle = c \right\}$$

This is ensured through a parallel repetition of InPrd.

The ParInPrd scheme is defined in Figure 5.

Let $[\tau'] := (\mathsf{pp}', [P], [\underline{\tilde{P}}], [\pi'], \mathbf{x})$ be the transcript of ParInPrd, where $\mathsf{pp}' := (n, \mathbf{g}, \underline{\tilde{\mathbf{g}}}, \mathbf{h}, u)$, $P := \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{h}}u^{\langle \mathbf{a},\mathbf{b}\rangle}$, $\underline{\tilde{P}} := \underline{\tilde{\mathbf{g}}}^{\mathbf{a}}\mathbf{h}^{\mathbf{h}}u^{\langle \mathbf{a},\mathbf{b}\rangle}$, $[\pi'] := (([L_i], \underline{[\tilde{L}_i]}, [R_i], \underline{[\tilde{R}_i]})_{i\in[\log n]}, (a, b))$ and $\mathbf{x} := (x_1, \ldots, x_{\log(n)})$ for $x_i \leftarrow \mathcal{H}_2(\mathsf{pp}', P, \underline{\tilde{P}}, (L_j, R_j, \underline{\tilde{L}_j, \tilde{R}_j})_{j=1}^i, (x_j)_{j=1}^{i-1})$.

Given such a transcript we can define two new transcripts τ and $\tilde{\tau}$ as follows;

$[\tau] := (\mathsf{pp} := (n, \mathbf{g}, \mathbf{h}, u), [P], [\pi] := (([L_1], [R_1])_{i\in\log n}, , (a, b)), \mathbf{x})$,

$[\tilde{\tau}] := (\underline{\tilde{\mathsf{pp}}} := (n, \underline{\tilde{\mathbf{g}}}, \mathbf{h}, u), [\underline{\tilde{P}}], [\tilde{\pi}] := (([\underline{\tilde{L}_i}], [\underline{\tilde{R}_i}])_{i\in\log n}, , (a, b)), \mathbf{x})$

The transcripts τ and $\tilde{\tau}$ both corresponds to transcript of InPrd for the relation in Equation (1) with $P := \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}}$ and $\underline{\tilde{P}} := \underline{\tilde{\mathbf{g}}}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}}$ respectively, with the difference that the challenges x_i is computed with some additional arguments given as input to the hash function. However, extending the argument of hash function do not change the unpredictability of the hash function, thus $\mathbf{x}$ is still valid challenges for the two InPrd proof. Thus, the security follows from the security of InPrd. Observe that in doing so the security bound degrades by at most a factor 2 (as the algebraic representation of the extra elements might depend on $\mathbf{g}$ or $\tilde{\mathbf{g}}$ respectively).

C.1 Simplified parallel inner product, $\widetilde{\mathsf{ParInPrd}}$

ParInPrd can be simplified to $\widetilde{\mathsf{ParInPrd}}$, where $\widetilde{\mathsf{ParInPrd}}$ is an inner product argument of the relation,

$$\mathsf{R}^{\widetilde{\mathsf{ParInPrd}}} = \left\{ (\mathbf{g}, \tilde{\mathbf{g}} \in \mathbb{G}_1^n,\ P, \tilde{P} \in \mathbb{G}_1,\ c \in \mathbb{Z}_p ; \mathbf{a} \in \mathbb{Z}_p^n) : \right. \\ \left. P = \mathbf{g}^{\mathbf{a}} \wedge \tilde{P} = \tilde{\mathbf{g}}^{\mathbf{a}} \wedge \langle \mathbf{a}, \mathbf{1} \rangle = c \right\}.$$

We obmit the expicit definition of $\widetilde{\mathsf{ParInPrd}}$ and simply note that it is obtained by applying the same simplification to ParInPrd as was applied to InPrd in deriving $\widetilde{\mathsf{InPrd}}$.

Moreover, the security of $\widetilde{\mathsf{ParInPrd}}$ follows from the security of ParInPrd by a similar reasoning as in Appendix B

D Homomorphic unforgeability under random message attack (HUFRMA)

As a stepping stone to an adaptive secure construction, we introduce the concept of Homomorphic Unforgeability under Random Message Attack (HUFRMA) and prove, a slightly modified version of, the smklhs construction in Figure 4 secure against adaptive queries in the HUFRMA game, Definition D.1. We give our HUFRMA secure construction in Figure 10 and refer to it as rnd.smklhs. Our construction is equivalent to the smklhs construction given in Figure 4 with the modification that the messages are sampled at random by the signing algorithm and returned together with the signature, instead of given as input.

Definition D.1 (HUFRMA). The HUFRMA with adaptive queries and non-adaptive corruptions is defined as the following security game between a PPT adversary $\mathscr{A}$ and a challenger $\mathscr{C}$.

Setup: Identical to **Setup** in Definition 2.3

Identity Queries: Identical to **Identity Queries** in Definition 2.3

Signing Queries: Identical to **Signing Queries** in Definition 2.3, except: (1) $\mathscr{A}$ adaptively submits queries on the form (Δ, ℓ)(2) $\mathscr{C}$ randomly samples a message $r \xleftarrow{\$} \mathcal{M}$, then computes the signature, $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{id}}, \Delta, \ell = (\tau, \mathsf{id}), r)$ of this message and returns the pair (r, σ). adversary

Definition D.2 (Forgeries). Let $(\mathcal{P}^*, \Delta^*, \{\mathsf{pk}^*_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, \hat{r}^*, \hat{\sigma}^*)$ be the tuple returned by $\mathscr{A}$ at the end of the HUFRMA security game (Definition D.1). Let $\mathcal{P}^* := (f, \{\ell_i\}_{i\in[n]})$. We say that $\mathscr{A}$ outputs a valid forgery if

$$\mathsf{Verify}(\mathcal{P}^*, \Delta^*, \{\mathsf{pk}^*_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, \hat{r}^*, \hat{\sigma}^*) == 1$$

and (at least) one of the following conditions holds:

Type-1 forgery Identical to **Type-1** in Definition 2.4

Type-2 forgery Identical to **Type-2** in Definition 2.4, except notational differences.

Type-3 forgery Identical to **Type-3** in Definition 2.4

Definition D.3 (Unforgeability (HUFRMA)). A mkhs scheme satisfies HUFRMA with non-adaptive corruptions if the advantage of any $\mathscr{A}$ in winning the HUFRMA security game, i.e. returning a successful forgery (Definition D.2), is negligible in the security parameter. That is,

$$\mathsf{Adv}^{HUFRMA}_{\mathsf{mkhs}}(\mathscr{A}, \lambda) := \Pr[\mathsf{HUFRMA}_{\mathscr{A},\mathsf{mkhs}}(\lambda) == 1] \leq \varepsilon(\lambda).$$

D.1 Our HUFRMA Construction

The construction in Figure 10 is unforgeable against the HUFRMA game given in Definition D.1. We denote this construction rnd.smklhs.

rnd.SetUp(λ) → pp

1 : pp ← SetUp(λ)

rnd.KeyGen(pp) → (id, sk, pk)

1 : (id, sk, pk) ← KeyGen(pp)

rnd.Sign(sk, Δ, ℓ) → σ

1 : $r \xleftarrow{\$} \mathcal{M}$

2 : $\sigma_r \leftarrow \mathsf{Sign}(\mathsf{sk}, \Delta, \ell, r)$

3 : $\sigma \leftarrow (r, \sigma_r)$

rnd.Eval($\mathcal{P}, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, \{\sigma_i\}_{i\in[n]}$) → $\hat{\sigma}$

1 : **parse** $\mathcal{P}$ as $(\{f_i\}_{i=1}^n, \cdot) \in \mathbb{Z}_p^n$

2 : **parse** $\sigma_i := (r_i, \sigma_{r,i})$, $i \in [n]$

3 : $\hat{r} \leftarrow \sum_{i\in[n]} r_i \cdot f_i \mod p$

4 : $\hat{\sigma}_r \leftarrow \mathsf{Eval}(\mathcal{P}, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, \{r_i\}_{i\in[n]}, \{\sigma_{r,i}\}_{i\in[n]})$

5 : $\hat{\sigma} \leftarrow (\hat{r}, \hat{\sigma}_r)$

rnd.Verify($\mathcal{P}, \Delta, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, \hat{\sigma}$) → *ver*

1 : **parse** $\hat{\sigma}$ as $(\hat{r}, \hat{\sigma}_r)$

2 : *ver* ← $\mathsf{Verify}(\mathcal{P}, \Delta, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, \hat{r}, \hat{\sigma}_r)$

Figure 10: Our adaptively secure rnd.smklhs **scheme for random messages. Lines in green color display black-box use of** smklhs**.**

Theorem D.4. *Let $\mathscr{A}$ be an algebraic PPT adversary that breaks the HUFRMA security with static corruptions of the* rnd.smklhs *scheme in Fig. 10 with advantage* $\mathsf{Adv}_{\mathsf{rnd.smklhs},\mathscr{A}}$ *in the Random Oracle Model (ROM). Then there exists a PPT adversary $\mathscr{B}^{\mathsf{coCDH}^*}$ that breaks the* coCDH* *problem over $\mathbb{G}_1, \mathbb{G}_2$ with advantage* $\mathsf{Adv}^{\mathsf{coCDH}^*}_{\mathbb{G}_1,\mathbb{G}_2}$ *or a PPT prover $\widetilde{\mathcal{P}_{\mathsf{alg}}}$ that breaks the security of $\widetilde{\mathsf{InPrd}}$ for any extractor $\mathcal{E}$ such that for any security parameter λ it holds,*

$$\mathsf{Adv}^{\mathsf{HUFRMA}}_{\mathsf{rnd.smklhs}}(\mathscr{A}, \lambda) \leq 2 \cdot \frac{\mathsf{Adv}^{\mathsf{coCDH}^*}_{\mathbb{G}_1,\mathbb{G}_2}(\mathscr{B}^{\mathsf{coCDH}^*}, \lambda)}{1 - \mathsf{Adv}^{\mathsf{ip-ext}}_{\widetilde{\mathsf{InPrd}},\mathsf{R}^{\widetilde{\mathsf{InPrd}}}}(\widetilde{\mathcal{P}_{\mathsf{alg}}}, \mathcal{E}, \lambda)}.$$

The proof of Theorem D.4 resembles that of Theorem 5.1 and is given below.

D.2 Proof of Theorem D.4

Fiore *et al.* made a fundamental result for mkhs schemes: security against adversaries that do not make corruption queries is equivalent to security against adversaries that make non-adaptive corruption queries, see [20, Prop. 1]. This means that the proof of Theorem D.4 does not need to deal with corruption queries; in particular, the only adaptive queries that the adversary can make are to create new honest parties, signing queries, and to the random oracles. Let $\mathscr{A}$ be the algebraic adversary against the unforgeability of rnd.smklhs that returns a forgery $(\mathcal{P}^*, \Delta^*, \{\mathsf{pk}^*_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, \hat{\mu}^*, \hat{\sigma}^*)$, and let $\mathscr{B}^{\mathsf{coCDH}^*}$ be the algorithm that reduces the forgery to a solution of an instance of the coCDH* problem.

The reduction $\mathscr{B}^{\mathsf{coCDH}^*}$ takes as input a bilinear group $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_{2,}, e)$, and a coCDH* challenge $(g_1, h_1 = g_1^{\mathsf{x}}, g_2, h_2 = g_2^{\mathsf{x}}, v)$ for $g_1, v \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, and its goal is to output v^{x}. To achieve this $\mathscr{B}^{\mathsf{coCDH}^*}$ simulates rnd.SetUp, rnd.KeyGen, rnd.Sign and the hash functions $\mathcal{H}_1, \mathcal{H}_2$ of rnd.smklhs as defined below.

Simulation.

Setup $\mathscr{B}^{\mathsf{coCDH}^*}$ initializes empty lists: L_{ID}, $L^{\mathsf{InPrd}}_{\mathcal{H}}$ and $L^{\mathsf{Sign}}_{\mathcal{H}}$. Then, it samples $\theta, \psi \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $\Theta_1 = g_1^{\theta}, \Theta_2 = g_2^{\theta}$ and $\Psi_1 = h_1^{\psi}, \Psi_2 = h_2^{\psi}$. Finally, the reduction samples $u \xleftarrow{\$} \mathbb{G}_1$, and sends the public parameters to $\mathscr{A}$. Note that $\mathcal{G}$ is generated using the groups and generators $(p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2)$ from the coCDH* challenge.

Identity Queries Given a query on the form id $\in$ ID, if id $\in L_{ID}$, *i.e.,* $\mathscr{A}$ already queried the identity id, then $\mathscr{B}^{\mathsf{coCDH}^*}$ ignores the query. Otherwise, the reduction samples the secret values $s_{\mathsf{id}}, \omega_{\mathsf{id}} \xleftarrow{\$} \mathbb{Z}_p^*$ and set the secret key to $\mathsf{sk}_{\mathsf{id}} = (s_{\mathsf{id}}, \omega_{\mathsf{id}})$, before computing the public key as $\mathsf{pk}_{\mathsf{id}} = (h_1^{s_{\mathsf{id}}}, h_2^{s_{\mathsf{id}}}, (v^{s_{\mathsf{id}}} g_1^{\omega_{\mathsf{id}}}))$. Then $\mathscr{B}^{\mathsf{coCDH}^*}$ updates the list of identities by $L_{ID} \leftarrow L_{ID} \cup \{(\mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}})\}$ before returning $\mathsf{pk}_{\mathsf{id}}$ to $\mathscr{A}$.

Sign Queries Given a query on the form (Δ, id). If $(\Delta, \ell, \cdot, \cdot) \in L_\sigma$, *i.e.,* $\mathscr{A}$ already queried the same dataset and label pair, then $\mathscr{B}^{\mathsf{coCDH}^*}$ ignores the query[9]. If $(\mathsf{id}, \cdot, \cdot) \notin L_{ID}$, i.e. the identity specified by the label has not been initialized, $\mathscr{B}^{\mathsf{coCDH}^*}$ queries the identity oracle with input id before continuing. Then, the reduction queries $(\Delta, \mathsf{id}, \mathsf{pk}_{\mathsf{id}})$ to $\mathcal{H}_1$. Next, the reduction computes a signature as: $\sigma = h_1^{s_{\mathsf{id}}(t+\theta r)+\psi\omega_{\mathsf{id}} r}$, where t, r is the randomness stores in $L_{\mathcal{H}}$ together with the pair (id, ℓ). Finally, $\mathscr{B}^{\mathsf{coCDH}^*}$ updates the list $L_\sigma \leftarrow L_\sigma \cup \{(\Delta, \ell, r, \sigma)\}$ and returns the random message and signature (r, σ).

Hash Queries The reduction answer hash queries for the hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$ as follows.

$\mathcal{H}_1$: On input $(\Delta, \ell = (\mathsf{id}, \tau), \mathsf{pk}_{\mathsf{id}})$. If $(\Delta, \ell, \cdot, \cdot) \in L^{\mathsf{Sign}}_{\mathcal{H}}$, the reduction retrieves the values (r, t) stored in $L^{\mathsf{Sign}}_{\mathcal{H}}$ Otherwise, the reduction samples $t \xleftarrow{\$} \mathbb{Z}_q^*, r \xleftarrow{\$} \mathcal{M}$ and stores the tuple (Δ, ℓ, r, t) in $L^{\mathsf{Sign}}_{\mathcal{H}}$. Finally, $\mathscr{B}^{\mathsf{coCDH}^*}$ returns $v^{-\psi r} g_1^t$.

$\mathcal{H}_2$: On input $(\mathsf{pp}, \tilde{P}, \{\tilde{L}_k, \tilde{R}_k\}_{k=1}^{i}, \{x_j\}_{j=1}^{i-1})$, if the input has already been queried, and stored in $L^{\mathsf{InPrd}}_{\mathcal{H}}$, $\mathscr{B}^{\mathsf{coCDH}^*}$ retrieves and returns the value stored in $L^{\mathsf{InPrd}}_{\mathcal{H}}$ together with the queried group elements. Otherwise, the reduction samples $x \xleftarrow{\$} \mathbb{Z}_p^*$ and stores the tuple $((\mathsf{pp}, \tilde{P}, \{\tilde{L}_k, \tilde{R}_k\}_{k=1}^{i}, \{x_j\}_{j=1}^{i-1}), x)$ in $L^{\mathsf{InPrd}}_{\mathcal{H}}$ before returning x.

Correctness of the simulation. Below we argue that our reduction simulates identity, hash and sign queries in a perfect way. Let q denote the unknown discrete logarithm of v with respect to g_1, *i.e.,* $v = g_1^q$.

Simulated public keys are of the form, $\mathsf{pk}_{\mathsf{id}} = (\mathrm{Y}^{(1)}_{\mathsf{id}}, \mathrm{Y}^{(2)}_{\mathsf{id}}, \Omega_{\mathsf{id}}) = (h_1^{s_{\mathsf{id}}}, h_2^{s_{\mathsf{id}}}, v^{s_{\mathsf{id}}} g_1^{\omega_{\mathsf{id}}})$, with the corresponding secret values $\mathsf{sk}_{\mathsf{id}} = (s_{\mathsf{id}}, \omega_{\mathsf{id}})$. We can rewrite $\mathrm{Y}^{(1)}_{\mathsf{id}}$ as $g_1^{\hat{s}_{\mathsf{id}}}$ where $\hat{s}_{\mathsf{id}} = \mathrm{x} s_{\mathsf{id}}$ and similarly $\mathrm{Y}^{(2)} = g_2^{\hat{s}_{\mathsf{id}}}$. Moreover, we have that $\Omega_{\mathsf{id}} = v^{s_{\mathsf{id}}} g_1^{\omega_{\mathsf{id}}} = g_1^{q s_{\mathsf{id}} + \omega_{\mathsf{id}}} = g_1^{\hat{\omega}_{\mathsf{id}}}$, where $\hat{\omega}_{\mathsf{id}} = q s_{\mathsf{id}} + \omega_{\mathsf{id}}$. Since $s_{\mathsf{id}}, \omega_{\mathsf{id}}$ are sampled at random, $\hat{s_{\mathsf{id}}}, \hat{\omega}_{\mathsf{id}}$ are also uniformly random. Thus, we have showed that that simulated keys are is distributed identically to the keys output by the real rnd.KeyGen algorithm.

Second, the responses to all hash queries are uniformly random and thereby indistinguishable from the output of any real cryptographic hash function (in the random oracle model). We show this for the two simulated hash functions separately. Given a query on the form (Δ, ℓ) we can write the response as, $\mathcal{H}_1(\Delta, \ell, \mathsf{pk}_{\mathsf{id}}) = v^{-\psi q} g_1^t = g_1^{-\psi q + t}$, from which is clear that the since $t \xleftarrow{\$} \mathbb{Z}_p^*$ the hash values are uniformly distributed in $\mathbb{G}_1$. For $\mathcal{H}_2$ it follows direct by construction that its output is uniformly random.

Third, while r is sampled uniformly random, it might be determined prior to $\mathscr{A}$ requesting a signature of (Δ, ℓ), since $\mathscr{A}$ might query the hash oracle on input $(\Delta, \ell, \mathsf{pk}_{\mathsf{id}})$ prior to requesting a signature of (Δ, ℓ). Since r is information-theoretically hidden by the output of the hash query, $\mathscr{A}$ can not detect this. Finally, we show that, given the public parameters, public keys and answers to hash queries output by the reduction, for any identity, id, the simulated signatures are identical to the signatures output by real rnd.Sign algorithm (Figure 10). It suffices to show that for any query on the form $(\mathsf{id}, r, \ell, \Delta)$ the simulated signature is identical to signature output by the smklhs Sign algorithm on the same input. By the discussion above we have that the simulated secret key is $\mathsf{sk}_{\mathsf{id}} = (s_{\mathsf{id}}, \omega_{\mathsf{id}})$ and the corresponding "real" secret key is $\hat{\mathsf{sk}}_{\mathsf{id}} = (\hat{s}_{\mathsf{id}}, \hat{\omega}_{\mathsf{id}}) = (\mathrm{x} s_{\mathsf{id}}, r s_{\mathsf{id}} + \omega_{\mathsf{id}})$, where $(s_{\mathsf{id}}, \omega_{\mathsf{id}})$ is known to the reduction. Let $\sigma \leftarrow \mathsf{Sign}(\hat{\mathsf{sk}}_{\mathsf{id}}, \Delta, \ell, r)$, we have that:

$$
\begin{aligned}
\sigma &= (\mathcal{H}_1(\Delta, \ell, \mathsf{pk}_{\mathsf{id}}) \cdot \Theta_1^r)^{\hat{s}_{\mathsf{id}}} \cdot \Psi_1^{\hat{\omega}_{\mathsf{id}} \cdot r} \\
&= \left((v^{-\psi r} g_1^t) \cdot (g_1^{\theta})^r\right)^{\hat{s}_{\mathsf{id}}} \cdot (g_1^{\psi \mathrm{x}})^{\hat{\omega}_{\mathsf{id}} r} \\
&= (v^{-\psi r} \cdot g_1^t)^{\hat{s}_{\mathsf{id}}} \cdot (g_1^{\hat{s}_{\mathsf{id}}})^{\theta r} \cdot (g_1^{\hat{\omega}_{\mathsf{id}}})^{\psi r \mathrm{x}} \\
&= (v^{-\psi r \mathrm{x} s_{\mathsf{id}}} \cdot h_1^{s_{\mathsf{id}} t}) \cdot h_1^{s_{\mathsf{id}} \theta r} \cdot (v^{s_{\mathsf{id}}} \cdot g_1^{\omega_{\mathsf{id}}})^{\psi r \mathrm{x}} \\
&= v^{-\psi r \mathrm{x} s_{\mathsf{id}}} \cdot h_1^{s_{\mathsf{id}} t} \cdot h_1^{s_{\mathsf{id}} \theta r} \cdot v^{\psi r \mathrm{x} s_{\mathsf{id}}} \cdot h_1^{\omega_{\mathsf{id}} \psi r} \\
&= h_1^{s_{\mathsf{id}} t + s_{\mathsf{id}} \theta r + \omega_{\mathsf{id}} \psi r} \\
&= h_1^{s_{\mathsf{id}}(t+\theta r) + \omega_{\mathsf{id}} \psi r},
\end{aligned}
$$

where the final right-hand-side term is equal to the simulated signature.

Finally, because a signature is statistically indistinguishable from a random element of $\mathbb{G}_1$, $\mathscr{A}$ cannot distinguish whether the output of $\mathcal{H}_1$ has been generated at random and the signature has been computed deterministically like in a real execution of the signing algorithm, or if the signature has been generated first like in the simulation.

The Single Signer Case. If the labeled program $\mathcal{P}^*$ only contains a single identity, *i.e.,* $t = 1$, then at the end of the game $\mathscr{A}$ outputs a forgery $(\mathcal{P}^*, \Delta^*, \hat{\mu}^*, \sigma^*)$, where $\sigma^* := (\sigma^*, \perp, \perp)$, along with an algebraic representation of σ^* respect to the group elements sent by the simulator. Since there is no inner product argument in this case, $\mathscr{B}^{\mathsf{coCDH}^*}$ skips the steps described in the next paragraph (Abort Probability) and proceeds directly to the extraction phase. Moreover, since Γ^* satisfies the verification check in smklhs for a single signer, by the bilinear property of the pairing, we can write Γ^* as

$$
\Gamma^* == \left(\prod_{i \in [n]} \mathcal{H}_1(\Delta, \ell_i, \mathsf{pk}_{\mathsf{id}})^{f_i^*}\right)^{\mathrm{x} s_{\mathsf{id}}} \cdot (\mathrm{Y}^{(1)}_{\mathsf{id}})^{\hat{r}^* \theta} \cdot \Omega_{\mathsf{id}}^{\hat{r}^* \mathrm{x} \psi} =
$$

$$
v^{\psi \mathrm{x} s_{\mathsf{id}} (\hat{r}^* - \sum_{i \in [n]} f_i r_i)} \cdot g_1^{\mathrm{x} s_{\mathsf{id}} (\sum_{i \in [n]} t_i f_i + \theta \hat{r}^*) + \psi \omega_{\mathsf{id}} \hat{r}^*} = v^{\square} g_1^{\spadesuit}.
$$

Given this, extraction follows from the procedure outlined in the coming sections but with $\clubsuit = \diamond := 0$ and $\square, \spadesuit$ as above.

Abort Probability. At the end of the game $\mathscr{A}$ returns a forgery $(\mathcal{P}^*, \Delta^*, \hat{r}^*, \sigma^*)$, where $\sigma^* := (\Gamma^*, R^*, \Omega^*, \tilde{\pi}_R^*, \tilde{\pi}_\Omega^*)$[10], alongside an algebraic representation of all the group elements. Let $\{K_i\}_{i \in [k]}$ denote the set of all group elements sent to the adversary during the simulation and $[R^*] := (R^*, a_1, \ldots, a_k)$, $[\Omega^*] := (\Omega^*, b_1, \ldots, b_k)$ be the representation of the two commitments R^* and Ω^* in the inner product proofs. The reduction then checks whether $a_i == 0$ for all $K_i \notin \{\mathrm{Y}^{(1)}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}^*}$, $b_i == 0$ for all $K_i \notin \{\Omega_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}^*}$ and

[9] Observe that $\mathscr{A}$ cannot query a signature on (Δ, ℓ, r_1) and (Δ, ℓ, r_2) where $r_1 \neq r_2$, as by definition tags are unique per each message.

[10] For the case $t \geq 2$.

$\sum_{i\in[n]} a_i == \sum_{i\in[n]} b_i == \hat{r}^*$. If this is not the case, $\mathscr{B}^{\text{coCDH}^*}$ aborts. The probability of this happening is bounded by the probability of breaking the security of the inner product argument. Indeed, in Lemma A.1 we show that if the adversary returns a representation of the commitment (R^* and Ω^* in the forgery) that assigns nonzero exponents to elements outside the instance (represented in the lemma by the vector $\mathbf{f}$), then there exists a reduction that solves the discrete logarithm problem (dl). Lemma A.1 analyzes the generic inner product argument, which we then proved secure also for simplified inner product argument. Combining the two results, we have that the probability that $\mathscr{B}^{\text{coCDH}^*}$ aborts is upper bounded by

$$\Pr[\mathscr{B}^{\text{coCDH}^*} \textit{ aborts}] \leq \mathsf{Adv}^{\text{ip-ext}}_{\widetilde{\mathsf{InPrd}},\mathsf{R}^{\widetilde{\mathsf{InPrd}}}}(\widetilde{\mathcal{P}_{\mathsf{alg}}}, \mathcal{E}, \lambda) \tag{9}$$

that is, if $\mathscr{B}^{\text{coCDH}^*}$ aborts with non-negligible probability, we can construct a PPT cheating prover $\widetilde{\mathcal{P}_{\mathsf{alg}}}$ that breaks the security of $\widetilde{\mathsf{InPrd}}$ for any extractor $\mathcal{E}$ exploiting $\mathscr{A}$.

Extraction. Without loss of generality, let the identities involved in $\mathcal{P}^*$ be $(\mathsf{id}_1, \mathsf{id}_2, \dots, \mathsf{id}_t)$. Remember that these all the identities in $\mathcal{P}^*$ are honest, as corruption queries are dismissed due to [20, Prop. 1]. If the reduction does not abort we have that,

$$R^* := \prod_{\mathsf{id}\in\mathcal{P}^*} (\mathrm{Y}^{(1)}_{\mathsf{id}})^{a_{\mathsf{id}}} \quad \text{and} \quad \Omega^* := \prod_{\mathsf{id}\in\mathcal{P}^*} \Omega_{\mathsf{id}}^{b_{\mathsf{id}}}, \tag{10}$$

and by excluding all zeros from the representations we can write, $[R^*] := (R^*, a_{\mathsf{id}_1}, \dots, a_{\mathsf{id}_t})$ and $[\Omega^*] := (\Omega^*, b_{\mathsf{id}_1}, \dots, b_{\mathsf{id}_t})$.

By the correctness of the rnd.smklhs scheme the Γ^* component in σ^* must satisfy all the verification checks in Verify of smklhs, specifically:

$$\Gamma^* == \left(\prod_{\mathsf{id}\in\mathcal{P}^*} h_{\mathsf{id}}^{\mathsf{x}s_{\mathsf{id}}}\right) \cdot (R^*)^{\theta} \cdot (\Omega^*)^{\mathsf{x}\psi},$$

where

$$\begin{aligned} h_{\mathsf{id}} &:= \prod_{i\in\mathsf{I}_{\mathsf{id}}} \mathcal{H}_1(\Delta, \ell_i, \mathsf{pk}_{\mathsf{id}})^{f_i^*} = \prod_{i\in\mathsf{I}_{\mathsf{id}}} \left(v^{-\psi r_i} g_1^{t_i}\right)^{f_i^*}, \\ R^* &:= \prod_{\mathsf{id}\in\mathcal{P}^*} (\mathrm{Y}^{(1)}_{\mathsf{id}})^{a_{\mathsf{id}}} = \prod_{\mathsf{id}\in\mathcal{P}^*} g_1^{\mathsf{x}s_{\mathsf{id}}a_{\mathsf{id}}}, \\ \Omega^* &:= \prod_{\mathsf{id}\in\mathcal{P}^*} (\Omega_{\mathsf{id}})^{b_{\mathsf{id}}} = \prod_{\mathsf{id}\in\mathcal{P}^*} ((v \cdot g_1^{\omega_{\mathsf{id}}})^{s_{\mathsf{id}}})^{b_{\mathsf{id}}}. \end{aligned}$$

Abstracting away from the variable names Γ^* has the following structure: $\Gamma^* = v^{\diamondsuit+\square} g_1^{\clubsuit+\spadesuit}$, where $\diamondsuit$ and $\clubsuit$ collect the exponents linked to all non-target datasets/identities/labels and $\square$ and $\spadesuit$ correspond to exponents linked to the target datasets/identities/labels, where target and non target datasets/identities/labels will be defined momentarily for the different types of forgeries. Our aim is to extract from Γ^* its v component *i.e.,* $v^{\mathsf{x}\triangle}$ where $\triangle$ is a known non-zero value.

Type-1 Forgeries The reduction can detect if the adversary outputs a type-1 forgery by checking if the dataset identifier Δ^* defined in the labelled program $\mathcal{P}^*$ was queried to the sign oracle, *i.e.,* if $(\Delta^*, \cdot, \cdot) \in L_\sigma$. If that is the case, then $\mathscr{B}^{\text{coCDH}^*}$ extracts from the forgery a witness to the co-CDH* instance as follows.

Since Δ^* is the same for all identities, we consider all identities to be target identities and thus, $\diamondsuit = \clubsuit = 0$. By inspection, we see that $\spadesuit \cdot \mathsf{x}^{-1} = \sum_{\mathsf{id}\in\mathcal{P}^*} s_{\mathsf{id}}\left(\sum_{i\in\mathsf{I}_{\mathsf{id}}} t_i f_i^* + \theta a_{\mathsf{id}}\right) + \psi\omega_{\mathsf{id}} b_{\mathsf{id}}$. Recall that $\spadesuit$ is a value known to the reduction and thereby $\mathscr{B}^{\text{coCDH}^*}$ can remove the "g_1 component" contributions from Γ^* by computing

$$\Gamma_0 := \frac{\Gamma^*}{g_1^{\spadesuit}}.$$

At this point we are left with $\Gamma_0 := v^{\square}$ and $\square := \mathsf{x} \cdot \triangle$, where $\triangle := \psi\left(\sum_{\mathsf{id}\in\mathcal{P}^*} s_{\mathsf{id}}\left(b_{\mathsf{id}} - \sum_{i\in\mathsf{I}_{\mathsf{id}}} r_i f_i^*\right)\right)$. If $\mathscr{A}$ output a type-1 forgery it means that for all $\mathsf{id} \in \mathcal{P}^*$ and all $i \in \mathsf{I}_{\mathsf{id}}$ $\mathscr{A}$ does not know the value r_i and cannot do better than randomly guessing its value. Indeed, assume w.l.o.g. that $\mathscr{A}$ queried $\mathcal{H}_1$ on (Δ^*, ℓ_i) for all the labels in the forgery, and got the oracle outputs $\{z_i \leftarrow v^{-\psi r_i} g_1^{t_i}\}_i$. Because the values of r_i, t_i are sampled at random and not used anywhere else, from the point of view of $\mathscr{A}$ each element z_i is such that $z_i := g_1^u$, where u uniformly distributed over $\mathbb{Z}_p$. Therefore, the probability that $\triangle$ is zero is bounded by the probability of $\mathscr{A}$ guessing the value of $\sum_{i\in\mathsf{I}} r_i f_i^*$ for all $\mathsf{id} \in \mathcal{P}^*$ such that all terms cancel, which in turn is bounded by the size of the message space, *i.e.,* $|\mathbb{Z}_p|$ in our construction. We conclude that with probability $(1 - \frac{1}{p})$ the reduction extracts the solution to the co-CDH* challenge by computing $\Gamma_1 := \Gamma_0^{(\triangle^{-1})} := v^{\mathsf{x}}$.

Type-2 Forgeries Let $\hat{r}_{\mathsf{id}} := \sum_{i\in\mathsf{I}_{\mathsf{id}}} r_i f_i^*$, then the reduction can detect which identities the adversary targeted by checking if either a_{id} or b_{id} equals $\hat{r}_{\mathsf{id}}$. Next we define the set of targeted identities to be, $\mathsf{ID}_{\mathsf{target}} := \{\mathsf{id} \in \mathcal{P}^* : a_{\mathsf{id}} \vee b_{\mathsf{id}} \neq \hat{r}_{\mathsf{id}}\}$. The reduction $\mathscr{B}^{\text{coCDH}^*}$ looks for type-2 forgeries against the identities $\mathsf{id} \in \mathsf{ID}_{\mathsf{target}}$, i.e, forgeries that satisfy

$$\begin{cases} \mathsf{id} \in \mathsf{ID}_{\mathsf{target}} \subset \mathcal{P}^* \\ \forall \mathsf{id} \in \mathsf{ID}_{\mathsf{target}} : a_{\mathsf{id}} \neq \hat{r}_{\mathsf{id}} \vee b_{\mathsf{id}} \neq \hat{r}_{\mathsf{id}} \end{cases} \tag{11}$$

Remark that since $\mathscr{A}$ outputs a type-2 forgery it follows that $\hat{r}^* := \sum_{\mathsf{id}\in\mathcal{P}^*} b_{\mathsf{id}} \neq \sum_{\mathsf{id}\in\mathcal{P}^*} \hat{r}_{\mathsf{id}}$, thus there must exist at least one $\mathsf{id} \in \mathsf{ID}_{\mathsf{target}}$ such that $b_{\mathsf{id}} \neq \hat{r}_{\mathsf{id}}$, and analogously for a_{id}.

If the adversary output a type-2 forgery, $\mathscr{B}^{\text{coCDH}^*}$ successfully extracts an answer to the co-CDH* challenger from the type-2 forgery against the target identities, $\mathsf{ID}_{\mathsf{target}}$.

By definition $a_{\mathsf{id}} = b_{\mathsf{id}} = \hat{r}_{\mathsf{id}}$ for all non-target identities, $\mathsf{id} \in \mathcal{P}^* \setminus \mathsf{ID}_{\mathsf{target}}$. Thus, $\diamondsuit = 0$ and $\clubsuit \cdot \mathsf{x}^{-1} = \sum_{\mathsf{id}\in\mathcal{P}^*\setminus\mathsf{ID}_{\mathsf{target}}} s_{\mathsf{id}}\left(\sum_{i\in\mathsf{I}_{\mathsf{id}}} t_i f_i^* + \theta\hat{r}_{\mathsf{id}}\right) + \psi\omega_{\mathsf{id}}\hat{r}_{\mathsf{id}}$. On the other hand, for the target identities it holds that, $\spadesuit \cdot \mathsf{x}^{-1} = \sum_{\mathsf{id}\in\mathsf{ID}_{\mathsf{target}}} s_{\mathsf{id}}\left(\sum_{i\in\mathsf{I}_{\mathsf{id}}} t_i f_i^* + \theta a_{\mathsf{id}}\right) + \psi\omega_{\mathsf{id}} b_{\mathsf{id}}$. The value of $\clubsuit$ and $\spadesuit$ are known to the reduction, thus $\mathscr{B}^{\text{coCDH}^*}$ can remove the "g_1 component" from Γ^* by computing

$$\Gamma_0 := \frac{\Gamma^*}{g_1^{\clubsuit+\spadesuit}}.$$

At this point we are left with $\Gamma_0 := v^{\square}$, where $\square := \mathsf{x} \cdot \triangle$ and $\triangle := \psi\left(\sum_{\mathsf{id}\in\mathsf{ID}_{\mathsf{target}}} s_{\mathsf{id}}(b_{\mathsf{id}} - \hat{r}_{\mathsf{id}})\right)$, and $\triangle \neq 0$ by condition (14). Finally, to extract the solution to the co-CDH* challenger, $\mathscr{B}$ simply computes

$$\Gamma_1 := \Gamma_0^{(\triangle^{-1})} := v^{\mathsf{x}}.$$

Type-3 Forgeries The reduction can detect which labels the adversary targeted by checking which labels $\ell_i^* \in \mathcal{P}^*$ was not queried to the sign oracle, *i.e.,* $(\Delta^*, \ell_i^*, \cdot) \notin L_\sigma$. Thus, we define the set of targeted labels to be, $\ell_{\text{target}} := \{\ell^* \in \mathcal{P}^* : (\Delta^*, \ell_i^*, \cdot) \notin L_\sigma\}$. Furthermore, we define the target identities to be as follows, $\text{ID}_{\text{target}} := \{\text{id} \in \mathcal{P}^* : \exists\, \ell^* = (\cdot, \text{id}) \in \ell_{\text{target}}\}$ $\mathscr{B}^{\text{coCDH}^*}$ extracts a solution to the co-CDH* problem from the forgery as follows. For all labels we have that $\mathcal{H}_1(\Delta, \ell_i, \text{pk}_{\text{id}}) \leftarrow v^{\psi r_i} g_1^{t_i}$. Specifically for all non target labels, *i.e.,* $\forall \ell_i^* \in \mathcal{P}^* \backslash \ell_{\text{target}}$, the value of r_i is the message identified by Δ^*, ℓ^* and the value of t_i is included in the signature Γ_i. However, for all target labels, *i.e.,* $\forall \ell_i^* \in \ell_{\text{target}}$, the values of r_i, t_i are sampled at random by $\mathscr{B}^{\text{coCDH}^*}$ upon generating the hash and not shared with $\mathscr{A}$ (this is analogous to what happens in the case of type-1 forgeries). Hence, the value of r_i is information-theoretically hidden, and $\mathscr{A}$ cannot to better than randomly guessing its value.

By inspection, we see that $\clubsuit \cdot \mathsf{x}^{-1} = \sum_{\text{id} \in \mathcal{P}^* \backslash \text{ID}_{\text{target}}} s_{\text{id}} \big(\sum_{i \in \mathsf{I}_{\text{id}}} t_i f_i^* + \theta a_{\text{id}} \big) + \psi \omega_{\text{id}} b_{\text{id}}$ and $\spadesuit \cdot \mathsf{x}^{-1} = \sum_{\text{id} \in \text{ID}_{\text{target}}} s_{\text{id}} \big(\sum_{i \in \mathsf{I}_{\text{id}}} t_i f_i^* + \theta a_{\text{id}} \big) + \psi \omega_{\text{id}} b_{\text{id}}$. Remark that the value of both $\clubsuit$ and $\spadesuit$ are known to the reduction and thereby $\mathscr{B}^{\text{coCDH}^*}$ can remove the "g_1 component" for all identities–target or non target– from Γ^* by computing

$$\Gamma_0 := \frac{\Gamma^*}{g_1^{\clubsuit + \spadesuit}}.$$

At this point we are left with $\Gamma_0 := v^{\square + \diamond}$. First, consider $\diamond := \mathsf{x} \cdot \triangle$ and $\triangle = \psi \big(\sum_{\text{id} \in \mathcal{P}^* \backslash \text{ID}_{\text{target}}} s_{\text{id}} \big(b_{\text{id}} - \sum_{i \in \mathsf{I}_{\text{id}}} r_i f_i^* \big) \big)$. Since $\mathscr{A}$ behaves honestly for $\text{id} \in \mathcal{P}^* \backslash \text{ID}_{\text{target}}$, we have that $b_{\text{id}} = \sum_{i \in \mathsf{I}_{\text{id}}} r_i f_i^*$, and consequently $\triangle = 0$.

Next consider $\square = \mathsf{x} \cdot \triangle$ and $\triangle = \psi \big(\sum_{\text{id} \in \text{ID}_{\text{target}}} s_{\text{id}} \big(b_{\text{id}} - \sum_{i \in \mathsf{I}_{\text{id}}} r_i f_i^* \big) \big)$. If $\mathscr{A}$ output a type-3 forgery it means that for all $\text{id} \in \text{ID}_{\text{target}}$ there exists, at lest one, $i \in \mathsf{I}_{\text{id}}$ such that $\ell_i^* \in \ell_{\text{target}}$ and consequently for which $\mathscr{A}$ don't know the value r_i and can't do better than randomly guessing its value. Therefore, the probability that $\triangle$ is zero is bounded by the probability of $\mathscr{A}$ guessing the value of $\sum_{i \in \mathsf{I}} r_i f_i^*$ for all $\text{id} \in \text{ID}_{\text{target}}$ such that all terms cancel. This is bounded by the size of the message space, *i.e.,* $|\mathbb{Z}_p|$ in our construction.

To conclude, with probability $(1 - \frac{1}{p})$ the reduction extracts the solution to the co-CDH* challenge by computing $\Gamma_1 := \Gamma_0^{(\triangle^{-1})} = v^{\mathsf{x}}$.

Success Probability. The success probability of $\mathscr{B}^{\text{coCDH}^*}$ in winning the coCDH* game can be bounded by the minimum success probability depending on the type of forgery, that is,

$$\begin{aligned}
&\text{Adv}^{\text{coCDH}^*}_{\mathbb{G}_1, \mathbb{G}_2}(\mathscr{B}^{\text{coCDH}^*}, \lambda) \geq \\
&\quad \left(1 - \frac{1}{p}\right) \cdot \text{Adv}^{\text{HUFRMA}}_{\text{rnd.smklhs}}(\mathscr{A}, \lambda) \cdot \left(1 - \Pr[\mathscr{B}^{\text{coCDH}^*} \; \textit{aborts}]\right) \\
&\quad \geq \frac{1}{2} \text{Adv}^{\text{HUFRMA}}_{\text{rnd.smklhs}} \left(1 - \text{Adv}^{\text{ip-ext}}_{\widetilde{\text{InPrd}}, \mathsf{R}^{\widetilde{\text{InPrd}}}}(\widetilde{\mathcal{P}_{\text{alg}}}, \mathcal{E}, \lambda)\right)
\end{aligned}$$

where we used the bound $\left(1 - \frac{1}{p}\right) \geq \frac{1}{2}$ and Equation (9).

adpt.SetUp(λ) → pp

1 : $\text{pp}_{\text{rnd}} := \leftarrow \text{rnd.SetUp}(\lambda)$,
/ $(\mathcal{G}, \text{ID}, \mathcal{M}, \mathcal{T}, (\Theta_{\text{rnd},1}, \Theta_{\text{rnd},2}), (\Psi_{\text{rnd},1}, \Psi_{\text{rnd},2}), u, \mathcal{H}_1, \mathcal{H}_2)$
2 : $\theta_\mu \xleftarrow{\$} \mathbb{Z}_p, \Theta_{\mu,1} := g_1^{\theta_\mu}, \Theta_{\mu,2} := g_2^{\theta_\mu}$
3 : $\psi_\mu \xleftarrow{\$} \mathbb{Z}_p, \Psi_{\mu,1} := g_1^{\psi_\mu}, \Psi_{\mu,2} := g_2^{\psi_\mu}$
4 : $\text{pp} \leftarrow (\text{pp}_{\text{rnd}}, \text{pp}_\mu)$, for $\text{pp}_\mu := (\Theta_{\mu,1}, \Theta_{\mu,2}, \Psi_{\mu,1}, \Psi_{\mu,2})$

adpt.KeyGen(pp) → (id, sk, pk)

1 : $(\text{id}, \text{sk}_{\text{rnd}}, \text{pk}_{\text{rnd}}) \leftarrow \text{rnd.KeyGen}(\text{pp}_{\text{rnd}})$
2 : $\text{sk}_\mu := (s_\mu, \omega_\mu) \xleftarrow{\$} \mathbb{Z}_p^* \times \mathbb{Z}_p^*$
3 : $\text{pk}_\mu := \big(\Upsilon_\mu^{(1)} \leftarrow g_1^{s_\mu}, \Upsilon_\mu^{(2)} \leftarrow g_2^{s_\mu}, \Omega_\mu \leftarrow g_1^{\omega_\mu}\big)$
4 : $\text{sk} \leftarrow (\text{sk}_{\text{rnd}}, \text{sk}_\mu)$
5 : $\text{pk} \leftarrow (\text{pk}_{\text{rnd}}, \text{pk}_\mu)$

adpt.Sign(sk, Δ, ℓ, μ) → σ

1 : $(r, \sigma_r) \leftarrow \text{rnd.Sign}(\text{sk}_{\text{rnd}}, \Delta, \ell, r)$
2 : $\sigma_\mu \leftarrow \big(\mathcal{H}_1(\Delta, \ell, \text{pk}) \cdot \Theta_{\mu,1}^{\mu}\big)^{s_\mu} \cdot \Psi_{\mu,1}^{\omega_\mu \cdot (\mu + r)}$
3 : $\sigma \leftarrow (r, \sigma_r, \sigma_\mu)$

adpt.Eval($\mathcal{P}$, $\{\text{pk}_{\text{id}}\}_{\text{id} \in \mathcal{P}}$, $\{\mu_i\}_{i \in [n]}$, $\{\sigma_i\}_{i \in [n]}$) → $\hat{\sigma}$

1 : **parse** $\mathcal{P} := (\{f_i\}_{i=1}^n, \cdot) \in \mathbb{Z}_p^n, \sigma_i := (r_i, \sigma_{r,i}, \sigma_{\mu,i})$, $i \in [n]$
2 : $(\hat{r}, \hat{\sigma}_r) \leftarrow \text{rnd.Eval}(\mathcal{P}, \{\text{pk}_{\text{id,rnd}}\}_{\text{id} \in \mathcal{P}}, \{r_i, \sigma_{r,i}\}_{i \in [n]})$
/ $\hat{\sigma}_r := (\Gamma_r, R, \tilde{\pi}_R, \tilde{\pi}_{\Omega,r})$
3 : $\Gamma_\mu \leftarrow \prod_{i=1}^n \sigma_{\mu,i}^{f_i}$ / evaluated signature of messages
4 : **for** id ∈ $\mathcal{P}$ / consider only distinct id
5 : $\hat{r}_{\text{id}} \leftarrow \sum_{i \in \mathsf{I}_{\text{id}}} r_i \cdot f_i \mod p$ / each signer's randomness
6 : $\hat{\mu}_{\text{id}} \leftarrow \sum_{i \in \mathsf{I}_{\text{id}}} \mu_i \cdot f_i \mod p$ / each signer's contribution
7 : $\hat{\mu} \leftarrow \sum_{\text{id} \in \mathcal{P}} \hat{\mu}_{\text{id}}$
8 : **if** $(t == 1)$: $M \leftarrow \perp, \Omega \leftarrow \perp, \Omega_r \leftarrow \perp, \tilde{\pi}_M \leftarrow \perp, \tilde{\pi}_{\Omega_\mu} \leftarrow \perp, \tilde{\pi}_\Omega \leftarrow \perp$
9 : **else** / run the simplified succinct inner product arguments
Overwrite $\tilde{\pi}_{\Omega,r}$ with a parallel $\widetilde{\text{InPrd}}$
10 : $\text{pp}_\Omega \leftarrow (t, \{\Omega_{\text{id,rnd}}\}_{\text{id} \in \mathcal{P}}, \{\Omega_{\text{id},\mu}\}_{\text{id} \in \mathcal{P}}, u)$
11 : $\Omega_r = \prod_{\text{id} \in \mathcal{P}} (\Omega_{\text{id,rnd}})^{\hat{r}_{\text{id}}}, \Omega = \prod_{\text{id} \in \mathcal{P}} (\Omega_{\text{id},\mu})^{\hat{r}_{\text{id}}}$,
12 : $x \leftarrow \mathcal{H}_2(\text{pp}_\Omega, \Omega), \hat{\Omega}_r \leftarrow \Omega_r u^{\hat{r}x}, \hat{\Omega} \leftarrow \Omega u^{\hat{r}x}$
13 : $\tilde{\pi}_\Omega \leftarrow \widetilde{\text{ParInPrd}}.\text{P}\left(\text{pp}_\Omega, \hat{\Omega}_r, \hat{\Omega}; \{\hat{r}_{\text{id}}\}_{\text{id} \in \mathcal{P}}\right)$
Inner product proofs for verification of Γ_μ
14 : $M \leftarrow \prod_{\text{id} \in \mathcal{P}} (\Upsilon_{\text{id}}^{(1)})^{\hat{\mu}_{\text{id}}}, \Omega_\mu \leftarrow \prod_{\text{id} \in \mathcal{P}} \Omega_{\text{id}}^{\hat{\mu}_{\text{id}}}$
15 : $\text{pp}_M \leftarrow (t, \{\Upsilon_{\text{id},\mu}^{(1)}\}_{\text{id} \in \mathcal{P}}, u), x_M \leftarrow \mathcal{H}_2(M, \text{pp}_M)$
16 : $\text{pp}_{\Omega,\mu} \leftarrow (t, \{\Omega_{\text{id},\mu}\}_{\text{id} \in \mathcal{P}}, u), x_{\Omega,\mu} \leftarrow \mathcal{H}_2(\Omega_\mu, \text{pp}_{\Omega,\mu})$
17 : $\tilde{\pi}_M \leftarrow \widetilde{\text{InPrd}}.\text{P}\left(\text{pp}_M, M \cdot u^{\hat{\mu} x_\mu}; \{\hat{\mu}_{\text{id}}\}_{\text{id} \in \mathcal{P}}\right)$,
18 : $\tilde{\pi}_{\Omega,\mu} \leftarrow \widetilde{\text{InPrd}}.\text{P}\left(\text{pp}_{\Omega,\mu}, \Omega_\mu \cdot u^{\hat{\mu} x_{\Omega,\mu}}; \{\hat{\mu}_{\text{id}}\}_{\text{id} \in \mathcal{P}}\right)$
19 : $\hat{\sigma}_r \leftarrow (\Gamma_r, R, \tilde{\pi}_R, \tilde{\pi}_\Omega)$, / Remark the update $\tilde{\pi}_\Omega$
20 : $\hat{\sigma}_\mu \leftarrow (\Gamma_\mu, M, \Omega_\mu, \Omega, \tilde{\pi}_M, \tilde{\pi}_{\Omega,\mu})$
21 : $\hat{\sigma} \leftarrow \left(\hat{r}, \hat{\sigma}_r, \hat{\sigma}_\mu\right)$

Figure 11: Our adaptively secure adpt.smklhs **scheme. Lines in green highlight the differences from** smklhs**.**

E Making smklhs adaptive secure

Figure 11 and Figure 12 presents adpt.smklhs, a variation of our base scheme that provably achieves security against adaptive signing queries, with non-adaptive corruptions. This means that $\mathscr{A}$ has no restriction on the signing queries, and is allowed to corrupt an identity id only if id had not been already queried to the signing or the verification oracles. This is the strongest attacker model present in the literature for multi key homomorphic signatures. Our construction and proof makes black box use of the rnd.smklhs scheme.

$\mathsf{adpt.Verify}(\mathcal{P}, \Delta, \{\mathsf{pk_{id}}\}_{\mathsf{id}\in\mathcal{P}}, \hat{\mu}, \hat{\sigma}) \to ver$

1 : **parse** $\mathcal{P}$ as $(\{f_i\}_{i=1}^n, \{\ell_i\}_{i=1}^n)$

2 : **parse** $\hat{\sigma}$ as $\left(\hat{r}, \hat{\sigma}_r, \hat{\sigma}_\mu\right)$

3 : **if** $(t == 1)$ / single-signer signature

4 : $ver_r \leftarrow e\left(\Gamma_r, g_2\right) == e\left((\mathrm{Y}^{(1)}_{\mathsf{id,rnd}})^{\hat{r}}, \Theta_{\mathsf{rnd},2}\right) \cdot e\left(\Omega^{\hat{r}}_{\mathsf{id,rnd}}, \Psi_{\mathsf{rnd},2}\right) \cdot e\left(\prod_{i=1}^{n} \mathcal{H}_1(\Delta, \ell_i, \mathsf{pk_{id}})^{f_i}, \mathrm{Y}^{(2)}_{\mathsf{id,rnd}}\right)$

5 : $ver_\mu \leftarrow e\left(\Gamma_\mu, g_2\right) == e\left((\mathrm{Y}^{(1)}_{\mathsf{id},\mu})^{\hat{\mu}}, \Theta_{\mu,2}\right) \cdot e\left(\Omega^{\hat{\mu}+\hat{r}}_{\mathsf{id},\mu}, \Psi_{\mu,2}\right) \cdot e\left(\prod_{i=1}^{n} \mathcal{H}_1(\Delta, \ell_i, \mathsf{pk_{id}})^{f_i}, \mathrm{Y}^{(2)}_{\mathsf{id},\mu}\right)$

6 : **else** / multi-signer sigature

Verify $\hat{\sigma}_r$ / Can't use rnd.Verify black-box due to the parallel $\tilde{\pi}_\Omega$

7 : $\mathsf{pp}_R \leftarrow (t, \{\mathrm{Y}^{(1)}_{\mathsf{id,rnd}}\}_{\mathsf{id}\in\mathcal{P}}, u)$,

8 : $\mathsf{pp}_\Omega \leftarrow (t, \{\Omega_{\mathsf{rnd,id}}\}_{\mathsf{id}\in\mathcal{P}}, \{\Omega_{\mu,\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, u)$,

9 : $x_r \leftarrow \mathcal{H}_2(\mathsf{pp}_R, R), x \leftarrow \mathcal{H}_2(\mathsf{pp}_\Omega, \Omega_r, \Omega)$,

10 : $ver_{r,1} \leftarrow \widetilde{\mathsf{InPrd}}.\mathsf{V}\left(\mathsf{pp}_R, R \cdot u^{x\hat{r}}\tilde{\pi}_R\right)$

11 : $ver_{r,2} \leftarrow \widetilde{\mathsf{ParInPrd}}.\mathsf{V}\left(\mathsf{pp}_\Omega, \Omega_r u^{x\hat{r}}, \Omega u^{x\hat{r}}, \tilde{\pi}_\Omega\right)$

12 : $ver_{r,3} \leftarrow e(\Gamma_r, g_2) == e(R, \Theta_{\mathsf{rnd},2}) \cdot e(\Omega_r, \Psi_{\mathsf{rnd},2}) \cdot \prod_{\mathsf{id}\in\mathcal{P}} e\left(\prod_{i\in I_{\mathsf{id}}} \mathcal{H}_1(\Delta, \ell_i, \mathsf{pk_{id}})^{f_i}, \mathrm{Y}^{(2)}_{\mathsf{rnd,id}}\right)$

13 : $ver_r = (ver_{r,1} \wedge ver_{r,2} \wedge ver_{r,3})$

Verify $\hat{\sigma}_\mu$

14 : $\mathsf{pp}_M \leftarrow (t, \{\mathrm{Y}^{(1)}_{\mathsf{id},\mu}\}_{\mathsf{id}\in\mathcal{P}}, u)$

15 : $\mathsf{pp}_{\Omega,\mu} \leftarrow (t, \{\Omega_{\mu,\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, u)$

16 : $x_M \leftarrow \mathcal{H}_2(\mathsf{pp}_M, M), x_{\Omega,\mu} \leftarrow \mathcal{H}_2(\mathsf{pp}_{\Omega,\mu}, \Omega_\mu)$

17 : $ver_{\mu,1} \leftarrow \widetilde{\mathsf{InPrd}}.\mathsf{V}\left(\mathsf{pp}_M, M \cdot u^{x_\mu\hat{\mu}}, \tilde{\pi}_M\right)$,

18 : $ver_{\mu,2} \leftarrow \widetilde{\mathsf{InPrd}}.\mathsf{V}\left(\mathsf{pp}_{\Omega_\mu}, \Omega_\mu \cdot u^{x_{\Omega,\mu}\hat{\mu}}, \tilde{\pi}_{\Omega,\mu}\right)$

19 : $ver_{\mu,3} \leftarrow e(\Gamma_\mu, g_2) == e(M, \Theta_{\mu,2}) \cdot e(\Omega_\mu, \Psi_{\mu,2}) \cdot e(\Omega, \Psi_{\mu,2}) \cdot \prod_{\mathsf{id}\in\mathcal{P}} e\left(\prod_{i\in I_{\mathsf{id}}} \mathcal{H}_1(\Delta, \ell_i, \mathsf{pk_{id}})^{f_i}, \mathrm{Y}^{(2)}_{\mu,\mathsf{id}}\right)$

20 : $ver_\mu = (ver_{\mu,1} \wedge ver_{\mu,2} \wedge ver_{\mu,3})$

21 : $ver \leftarrow (ver_r \wedge ver_\mu)$

Figure 12: The verification of the adaptively secure adpt.smklhs **scheme. Lines in green highlight the differences from** smklhs**.**

We prove Theorem 6.1 in subsection E.1.

E.1 Proof of Theorem 6.1

As before, we rely on the result by Fiore *et al.* to reduce security against adversaries that do not make corruption queries to security against adversaries that make non-adaptive corruption queries (cf. [20, Prop. 1]). This means that the proof of Theorem 6.1 does not need to deal with corruption queries; in particular, the only adaptive queries that the adversary can make are to create new honest parties, signing queries, and to the random oracles. Let $\mathscr{A}$ be the algebraic adversary against the unforgeability of adpt.smklhs that returns a forgery $(\mathcal{P}^*, \Delta^*, \{\mathsf{pk}^*_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, \hat{\mu}^*, \hat{\sigma}^*)$, and let $\mathscr{B}^{\mathsf{coCDH}^*}$ be the algorithm that reduces the forgery to a solution of an instance of the coCDH* problem. In what follows, green text highlights parts of the construction or reduction uses the rnd.smklhs scheme and its security proof.

The reduction $\mathscr{B}^{\mathsf{coCDH}^*}$ takes as input a bilinear group $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e)$, and a coCDH* challenge $(g_1, h_1 = g_1^{\mathsf{x}}, g_2, h_2 = g_2^{\mathsf{x}}, v)$ for $g_1, v \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, and its goal is to output v^{x}. To achieve this $\mathscr{B}^{\mathsf{coCDH}^*}$ simulates adpt.SetUp, adpt.KeyGen, adpt.Sign and the hash functions $\mathcal{H}_1, \mathcal{H}_2$ of adpt.smklhs as defined below.

Simulation.

Setup $\mathscr{B}^{\mathsf{coCDH}^*}$ initializes empty lists: L_{ID},L_σ, $L^{\mathsf{InPrd}}_{\mathcal{H}}$ and $L^{\mathsf{Sign}}_{\mathcal{H}}$. The reduction computes pp_{rnd}, by running rnd.SetUp(λ) as described in Figure 10, with the exception that $\mathcal{G}$ is generated using the groups and generators $(p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2)$ from the coCDH* challenge. Then $\mathscr{B}^{\mathsf{coCDH}^*}$ computes pp_μ, by sampling $\theta_\mu, \psi_\mu \xleftarrow{\$} \mathbb{Z}^*_p$ and computing $\Theta_{\mu,i} = g_i^{\theta_\mu}, \Psi_{\mu,i} = g_i^{\psi_\mu}$, for $i = 1, 2$. Finally, the reduction sends the public parameters $\mathsf{pp} = (\mathsf{pp}_{\mathsf{rnd}}, \mathsf{pp}_\mu = (\Theta_{\mu,1}, \Theta_{\mu,2}, \Psi_{\mu,1}, \Psi_{\mu,2}))$ to $\mathscr{A}$.

Identity Queries Given a query on the form $\mathsf{id} \in \mathsf{ID}$, if $\mathsf{id} \in L_{ID}$, *i.e.,* $\mathscr{A}$ already queried the identity id, then $\mathscr{B}^{\mathsf{coCDH}^*}$ ignores the query. Otherwise, the reduction computes the keys $(\mathsf{sk_{id,rnd}}, \mathsf{pk_{id,rnd}})$ by running rnd.KeyGen($\mathsf{pp}_{\mathsf{rnd}}$) as described in Figure 10. Then samples the secret values $s_{\mathsf{id},\mu}, \omega_{\mathsf{id},\mu} \xleftarrow{\$} \mathbb{Z}^*_p$, sets $\mathsf{sk_{id},\mu} = (s_{\mathsf{id},\mu}, \omega_{\mathsf{id},\mu})$, and computes $\mathsf{pk_{id},\mu} := (h_1^{s_{\mathsf{id},\mu}}, h_2^{s_{\mathsf{id},\mu}}, (v^{s_{\mathsf{id},\mu}} g_1^{\omega_{\mathsf{id},\mu}}))$. The public and secret keys are $\mathsf{pk_{id}} := (\mathsf{pk_{id,rnd}}, \mathsf{pk_{id},\mu})$ and $\mathsf{sk_{id}} := (\mathsf{sk_{id,rnd}}, \mathsf{sk_{id},\mu})$ respectively. Finally $\mathscr{B}^{\mathsf{coCDH}^*}$ updates the list of identities by $L_{ID} \leftarrow L_{ID} \cup \{(\mathsf{id}, \mathsf{sk_{id}}, \mathsf{pk_{id}})\}$ before returning $\mathsf{pk_{id}}$ to $\mathscr{A}$.

Sign Queries Given a query on the form $(\Delta, \ell = (\mathsf{id}, \cdot), \mu)$.
If $(\Delta, \ell, \cdot, \cdot) \in L_\sigma$, *i.e.,* $\mathscr{A}$ already queried the same dataset and identity pair, then $\mathscr{B}^{\mathsf{coCDH}^*}$ ignores the query. If $(\mathsf{id}, \cdot, \cdot) \notin L_{ID}$, i.e. the identity specified by the label has not been initialized, $\mathscr{B}^{\mathsf{coCDH}^*}$ queries the identity oracle with input id before continuing.
The reduction computes a signature and hash for each entry as follows: First, it runs $\mathcal{H}_1(\Delta, \ell, \mathsf{pk_{id}})$ to obtain t_v, t_g. Second, it computes $\sigma_r \leftarrow \mathsf{Sign}(\mathsf{sk_{id,rnd}}, \Delta, \ell, r)$ where,

$$r := \psi_\mu^{-1}(t_v - \mu\psi_\mu) \tag{12}$$

Third, it computes $\sigma_\mu = h_1^{s_{\mathsf{id},\mu}(t_g + \theta\mu) + \psi\omega_{\mathsf{id},\mu}(\mu + r)}$. Fourth, it updates the list $L_\sigma \leftarrow L_\sigma \cup \{(\Delta, \ell, \mu, \sigma)\}$, where $\sigma :=$

$(r, \sigma_r, \sigma_\mu)$. Finally, the reduction returns the signature σ to $\mathscr{A}$.

Hash Queries The reduction answer hash queries for the hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$ as follows.

$\mathcal{H}_1$: On input $(\Delta, \ell = (\mathsf{id}, \tau), \mathsf{pk}_{\mathsf{id}})$. If $(\Delta, \ell, \cdot, \cdot) \in L^{\mathsf{Sign}}_{\mathcal{H}}$ the reduction retrieves the values (t_v, t_g) stored in $L^{\mathsf{Sign}}_{\mathcal{H}}$ together with the queried dataset and label. Otherwise, the reduction samples $t_v, t_g \xleftarrow{\$} \mathbb{Z}^*_p$ and stores the tuple (Δ, ℓ, t_v, t_g) in $L^{\mathsf{Sign}}_{\mathcal{H}}$. Finally, $\mathscr{B}^{\mathsf{coCDH}^*}$ returns $v^{-t_v} g_1^{t_g}$

$\mathcal{H}_2$: On input $(\mathsf{pp}, \tilde{P}, \{\tilde{L}_k, \tilde{R}_k\}^i_{k=1}, \{x_j\}^{i-1}_{j=1})$, if the query has already been queries, and in stored in $L^{\mathsf{InPrd}}_{\mathcal{H}}$, $\mathscr{B}^{\mathsf{coCDH}^*}$ retrieves and returns the value stored in $L^{\mathsf{InPrd}}_{\mathcal{H}}$ together with the queried group elements. Otherwise, the reduction samples $x \xleftarrow{\$} \mathbb{Z}^*_p$ and stores $((\mathsf{pp}, \tilde{P}, \{\tilde{L}_k, \tilde{R}_k\}^i_{k=1}, \{x_j\}^{i-1}_{j=1}), x)$ in $L^{\mathsf{InPrd}}_{\mathcal{H}}$ before returning x.

Correctness of the simulation. Below we argue that our reduction simulates hash, identity, and sign queries in a perfect way. Let d denote the unknown discrete logarithm of v with respect to g_1, *i.e.,* $v = g_1^d$.

The responses to all hash queries are uniformly random and thereby indistinguishable from the output of any real cryptographic hash function (in the random oracle model). We show this for the two simulated hash functions separately. Given a query on the form (Δ, ℓ) we can write the response as, $\mathcal{H}_1(\Delta, \ell, \mathsf{pk}_{\mathsf{id}}) = v^{-t_v} g_1^{t_g} = g_1^{-dt_v + t_g}$, from which is clear that the since $t_v, t_g \xleftarrow{\$} \mathbb{Z}^*_p$ the hash values are uniformly distributed in $\mathbb{G}_1$. For $\mathcal{H}_2$ it follows direct by construction that its output is uniformly random.

The keys $\mathsf{pk}_{\mathsf{rnd}}$ are honestly simulated, thus it suffices to argue that pk_μ are indistinguishable from the keys output by adpt.KeyGen. We have that the simulated public keys are of the form $\mathsf{pk}_{\mathsf{id},\mu} = (\mathrm{Y}^{(1)}_{\mathsf{id},\mu}, \mathrm{Y}^{(2)}_{\mathsf{id},\mu}, \Omega_{\mathsf{id},\mu})$, and the secret keys $\mathsf{sk}_{\mathsf{id},\mu} = (s_{\mathsf{id},\mu}, \omega_{\mathsf{id},\mu})$. By rewriting it follows that $\mathrm{Y}^{(1)}_{\mathsf{id},\mu} := g_1^{\hat{s}_{\mathsf{id},\mu}}$, $\mathrm{Y}^{(2)}_{\mathsf{id},\mu} := g_2^{\hat{s}_{\mathsf{id},\mu}}$, for $\hat{s}_{\mathsf{id},\mu} := \mathrm{x} s_{\mathsf{id},\mu}$; and $\Omega_{\mathsf{id},\mu} = g_1^{\hat{\omega}_{\mathsf{id},\mu}}$, for $\hat{\omega}_{\mathsf{id}} = d s_{\mathsf{id},\mu} + \omega_{\mathsf{id},\mu}$. Since $s_{\mathsf{id},\mu}, \omega_{\mathsf{id},\mu}$ are sampled at random it holds $\hat{s_{\mathsf{id},\mu}}, \hat{\omega}_{\mathsf{id},\mu}$ are also uniformly random. This concludes that simulated keys are is distributed identically to the keys output by adpt.KeyGen algorithm.

The pair, (r, σ_r) clearly satisfies the verification (by the correctness of rnd.smklhs in Figure 10 and since σ_r is honestly simulated). In addition, r computed in Equation (12) is distributed uniformly at random since t_v is random (moreover ψ_μ is non-zero, and $g_1^{t_g}$ masks t_v in $\mathcal{H}_1$ queries). Next, we argue that σ_μ is perfectly simulated given the public parameters, public keys and answers to hash queries output by the reduction. That is, we show that for any identity, id, the simulated signatures are identical to the signatures output by real adpt.Sign algorithm (Figure 11). It suffices to show that for any query on the form $(\mathsf{id}, \mu, \ell, \Delta)$ the simulated signature is identical to signature output by the real adpt.Sign algorithm on the same input. By the discussion above we have that the simulated secret key is $\mathsf{sk}_{\mathsf{id},\mu} = (s_{\mathsf{id},\mu}, \omega_{\mathsf{id},\mu})$ and the corresponding "real" secret key is $\hat{\mathsf{sk}}_{\mathsf{id},\mu} = (\hat{s}_{\mathsf{id},\mu} = \mathrm{x} s_{\mathsf{id},\mu}, \hat{\omega}_{\mathsf{id},\mu} = d s_{\mathsf{id},\mu} + \omega_{\mathsf{id},\mu})$.

For $\sigma \leftarrow \mathsf{Sign}(\hat{\mathsf{sk}}_{\mathsf{id}}, \Delta, \ell, \mu)$, we have that:

$$
\begin{aligned}
\sigma_\mu &= (\mathcal{H}_1(\Delta, \ell, \mathsf{pk}_{\mathsf{id}}) \cdot \Theta_1^{\mu})^{\hat{s}_{\mathsf{id},\mu}} \cdot \Psi_1^{\hat{\omega}_{\mathsf{id},\mu} \cdot (\mu + r)} \\
&= \left((v^{-t_v} g_1^{t_g}) \cdot (g_1^{\theta_\mu})^{\mu} \right)^{\hat{s}_{\mathsf{id},\mu}} \cdot (g_1^{\psi_\mu \mathrm{x}})^{\hat{\omega}_{\mathsf{id},\mu}(\mu + r)} \\
&= (v^{-t_v} \cdot g_1^{t_g})^{\hat{s}_{\mathsf{id},\mu}} \cdot (g_1^{\hat{s}_{\mathsf{id},\mu}})^{\theta_\mu \mu} \cdot (g_1^{\hat{\omega}_{\mathsf{id},\mu}})^{\psi_\mu \mathrm{x}(\mu + r)} \\
&= (v^{-\mathrm{x} t_v s_{\mathsf{id},\mu}} \cdot h_1^{s_{\mathsf{id},\mu} t_g}) \cdot h_1^{s_{\mathsf{id},\mu} \theta_\mu \mu} \cdot (v^{s_{\mathsf{id},\mu}} \cdot g_1^{\omega_{\mathsf{id},\mu}})^{\psi_\mu \mathrm{x}(\mu + r)} \\
&= v^{-\mathrm{x} t_v s_{\mathsf{id},\mu}} \cdot h_1^{s_{\mathsf{id},\mu} t_g} \cdot h_1^{s_{\mathsf{id},\mu} \theta_\mu \mu} \cdot v^{\psi_\mu \mathrm{x} s_{\mathsf{id},\mu}(\mu + r)} \cdot h_1^{\omega_{\mathsf{id},\mu} \psi_\mu (\mu + r)} \\
&= v^{-\mathrm{x} t_v s_{\mathsf{id},\mu}} \cdot v^{\mathrm{x} s_{\mathsf{id},\mu} \psi_\mu (\mu + \psi_\mu^{-1}(t_v - \mu \psi_\mu))} \cdot h_1^{s_{\mathsf{id},\mu}(t + \theta_\mu \mu) + \omega_{\mathsf{id},\mu} \psi_\mu (\mu + r)} \\
&= v^{-\mathrm{x} s_{\mathsf{id},\mu}(-t_v + \psi_\mu \mu + t_v - \psi_\mu \mu)} \cdot h_1^{s_{\mathsf{id},\mu}(t + \theta_\mu \mu) + \omega_{\mathsf{id},\mu} \psi_\mu (\mu + r)} \\
&= h_1^{s_{\mathsf{id},\mu}(t + \theta_\mu \mu) + \omega_{\mathsf{id},\mu} \psi_\mu (\mu + r)}
\end{aligned}
$$

where the final right-hand-side term is the simulated signature.

The Single Signer Case. If the labeled program $\mathcal{P}^*$ only contains a single identity, *i.e.,* $t = 1$, then at the end of the game $\mathscr{A}$ outputs a forgery $(\mathcal{P}^*, \Delta^*, \hat{\mu}^*, \sigma^*)$, where $\sigma^* := (\hat{\sigma}^*_r = (\Gamma^*_r, \perp, \perp), \hat{\sigma}^*_\mu = (\Gamma^*_\mu, \perp, \perp, \perp))$, along with the algebraic representation of Γ^*_r and Γ^*_μ with respect to the group elements sent by the simulator. Since there is no inner product argument in this case, $\mathscr{B}^{\mathsf{coCDH}^*}$ skips the steps described in the next paragraph (Abort Probability) and proceeds directly to the extraction phase. From the verification of Γ^*_r in Line 4 and the correctness of rnd.smklhs it follows that $\hat{r}$ is as in Equation (12). Moreover, since Γ^*_μ satisfies the verification check in Line 5 of adpt.Verify, by the bilinear property of the pairing, we can write Γ^*_μ as

$$
\begin{aligned}
&\Gamma^*_\mu == \\
&\quad (\mathrm{Y}^{(1)}_{\mathsf{id}})^{\hat{\mu}^* \theta_\mu} \cdot \Omega_{\mathsf{id}}^{\mathrm{x} \psi_\mu (\hat{\mu}^* + \hat{r}^*)} \cdot \left(\prod_{i \in [n]} \mathcal{H}_1(\Delta, \ell_i, \mathsf{pk}_{\mathsf{id}})^{f_i} \right)^{\mathrm{x} s_{\mathsf{id}}} \\
&= \left(v^{\psi_\mu (\hat{\mu}^* + \hat{r}^* - \sum_{i \in [n]} f_i t_{v,i})} \right)^{\mathrm{x} s_{\mathsf{id}}} \cdot \\
&\cdot \left(g_1^{\sum_{i \in [n]} t_{g,i} f_i + \theta_\mu \hat{\mu}^* + \psi_\mu \omega_{\mathsf{id}} (\hat{\mu}^* + \hat{r}^*)} \right)^{\mathrm{x} s_{\mathsf{id}}} \\
&= v^{\square} g_1^{\spadesuit}.
\end{aligned}
$$

Given this, extraction follows from the procedure outlined in the coming sections but with $\clubsuit = \diamond := 0$ and $\square, \spadesuit$ as above.

Abort Probability. At the end of the game $\mathscr{A}$ returns a forgery $(\mathcal{P}^*, \Delta^*, \hat{\mu}^*, \sigma^*)$, where $\sigma^* := (\hat{r}^* \sigma^*_r := (\Gamma^*_r, R^*, \Omega^*_r, \tilde{\pi}^*_R, \tilde{\pi}^*_{\Omega_r}), \sigma^*_\mu := (\Gamma^*_\mu, M^*, \Omega^*_\mu, \tilde{\pi}^*_M, \tilde{\pi}^*_{\Omega_\mu} \tilde{\pi}^*_{\bar{\Omega}}))$, alongside an algebraic representation of all the group elements. Let $\{K_i\}_{i \in [k]}$ denote the set of all group elements sent to the adversary during the simulation. The reduction aborts if: (1) any of inner product proofs is computed dishonestly; or (2) the signature $\hat{\sigma}^*_r$ is a forgery

We being to bound probability of the first abort condition. Let,

$$
\begin{aligned}
[R^*] &:= (R^*, a_1, \dots, a_k),\\
[\Omega_r^*] &:= (\Omega_\mu^*, b_1, \dots, b_k)\\
[M^*] &:= (M^*, c_1, \dots, c_k),\\
[\Omega_\mu^*] &:= (\Omega_\mu^*, d_1, \dots, d_k)\\
[\Omega^*] &:= (\Omega^*, e_1, \dots, e_k)
\end{aligned}
$$

be the representation of the five commitments R^*, Ω_r^*, M^*, Ω_μ^* and Ω^* in the inner product proofs $\tilde{\pi}_R^*, \tilde{\pi}_{\Omega_r^*}, \tilde{\pi}_M^*, \tilde{\pi}_{\Omega_\mu^*}$ and $\tilde{\pi}_{\Omega^*}$ respectively.

The reduction then checks whether,

- $a_i == 0$ for all $K_i \notin \{\mathrm{Y}^{(1)}_{\mathsf{id},\mathsf{rnd}}\}_{\mathsf{id}\in\mathcal{P}^*}$,
- $b_i == 0$ for all $K_i \notin \{\Omega_{\mathsf{id},\mathsf{rnd}}\}_{\mathsf{id}\in\mathcal{P}^*}$,
- $c_i == 0$ for all $K_i \notin \{\mathrm{Y}^{(1)}_{\mathsf{id},\mu}\}_{\mathsf{id}\in\mathcal{P}^*}$,
- $d_i == e_i == 0$ for all $K_i \notin \{\Omega_{\mathsf{id},\mu}\}_{\mathsf{id}\in\mathcal{P}^*}$,
- $\sum_{i\in[n]} a_i == \sum_{i\in[n]} b_i == \sum_{i\in[n]} e_i == \hat{r}^*$,
- $\sum_{i\in[n]} c_i == \sum_{i\in[n]} d_i == \hat{\mu}^*$,

and aborts if any of the equality checks fail. Additionally, the reduction then checks whether for all $\mathsf{id} \in \mathcal{P}^*$ that $c_i == b_j$ for i, j such that $K_i = \Omega_{\mathsf{id},\mathsf{rnd}}$ and $K_j = \Omega_{\mathsf{id},\mu}$. Note that given an honestly generated proofs equality should hold, given the verification of the parallel construction of $\tilde{\pi}_{\Omega_r^*}$ and $\tilde{\pi}_{\Omega^*}$ $\mathscr{B}^{\mathsf{coCDH}^*}$ aborts, if this is not the case.

The probability of $\mathscr{B}^{\mathsf{coCDH}^*}$ aborting due to any of the above equality checks failing is negligible, more specifically it is upper-bounded by the probability of breaking the security of the inner product argument. Indeed, in Lemma A.1 we show that if the adversary returns a representation of the commitment (M^*, Ω_μ^* and Ω^* in the forgery) that assigns nonzero exponents to elements outside the instance (represented in the lemma by the vector $\mathbf{f}$), then there exists a reduction that solves the discrete logarithm problem (dl). Lemma A.1 analyzes the generic inner product argument, which we then proved secure also for simplified inner product argument. This gives that the probability of the second abort condition is upper bounded by $\mathsf{Adv}^{\mathsf{ip-ext}}_{\widetilde{\mathsf{InPrd}},\mathsf{R}^{\widetilde{\mathsf{InPrd}}}}(\widetilde{\mathcal{P}_{\mathsf{alg}}}, \mathcal{E}, \lambda)$, that is, if $\mathscr{B}^{\mathsf{coCDH}^*}$ aborts with non-negligible probability, we can construct a PPT cheating prover $\widetilde{\mathcal{P}_{\mathsf{alg}}}$ that breaks the security of $\widetilde{\mathsf{InPrd}}$ for any extractor $\mathcal{E}$ exploiting $\mathscr{A}$.

Clearly the probability of the second abort condition is upper-bounded by $\mathsf{Adv}^{\mathsf{HUFRMA}}_{\mathsf{mkhs}}(\mathscr{A}, \lambda)$, which follows from Theorem D.4 and the fact that $\mathscr{B}^{\mathsf{coCDH}^*}$ can detect if $\hat{\sigma}_r$ is a forgery by the approach described in the proof of smklhs for the different types of forgeries.

Combining the two results, we have that the probability that $\mathscr{B}^{\mathsf{coCDH}^*}$ aborts is upper bounded by

$$
\begin{aligned}
\Pr[\mathscr{B}^{\mathsf{coCDH}^*}\ \mathit{aborts}] \leq & \mathsf{Adv}^{\mathsf{HUFRMA}}_{\mathsf{mkhs}}(\mathscr{A}, \lambda)\\
& + \mathsf{Adv}^{\mathsf{ip-ext}}_{\widetilde{\mathsf{InPrd}},\mathsf{R}^{\widetilde{\mathsf{InPrd}}}}(\widetilde{\mathcal{P}_{\mathsf{alg}}}, \mathcal{E}, \lambda).
\end{aligned} \tag{13}
$$

The bound in (13) shows that the reduction aborts only with negligible probability (assuming the buildling blocks are secure, as in the theorem statement).

Extraction. Without loss of generality, let the identities involved in $\mathcal{P}^*$ be $(\mathsf{id}_1, \mathsf{id}_2, \dots, \mathsf{id}_t)$. Remember that these all the identities in $\mathcal{P}^*$ are honest, as corruption queries are dismissed due to [20, Prop. 1]. If the reduction does not abort we have that,

$$
\begin{aligned}
M^* &:= \prod_{\mathsf{id}\in\mathcal{P}^*} (\mathrm{Y}^{(1)}_{\mathsf{id},\mu})^{c_{\mathsf{id}}}, \quad \Omega_\mu^* := \prod_{\mathsf{id}\in\mathcal{P}^*} (\Omega_{\mathsf{id},\mu})^{d_{\mathsf{id}}},\\
&\text{and} \quad \Omega^* := \prod_{\mathsf{id}\in\mathcal{P}^*} (\Omega_{\mathsf{id},\mu})^{\hat{r}_{\mathsf{id}}},
\end{aligned}
$$

and by excluding all zeros from the representations we can write, $[M^*] := (M^*, a_{\mathsf{id}_1}, \dots, a_{\mathsf{id}_t}), [\Omega_\mu^*] := (\Omega_\mu^*, b_{\mathsf{id}_1}, \dots, b_{\mathsf{id}_t})$ and $[\Omega^*] := (\Omega^*, \hat{r}_{\mathsf{id}_1}, \dots, \hat{r}_{\mathsf{id}_t})$.

Furthermore, as the simulation has not aborted, it holds that $\Omega_r^* := \prod_{\mathsf{id}\in\mathcal{P}^*} \Omega_{id,r}^{\hat{r}_{\mathsf{id}}}$ and $\Omega^* := \prod_{\mathsf{id}\in\mathcal{P}^*} \Omega_{id,\mu}^{\hat{r}_{\mathsf{id}}}$, for $\hat{r}_{\mathsf{id}} := \sum_{i\in\mathsf{I}_{\mathsf{id}}} r_i f_i^*$.

By the correctness of the adpt.smklhs scheme the Γ^* component in σ^* must satisfy the verification check in line 19 in adpt.Verify:

$$
\Gamma^* == (M^*)^{\theta_\mu} \cdot (\Omega_\mu^*)^{\mathrm{x}\psi_\mu} \cdot (\Omega^*)^{\mathrm{x}\psi_\mu} \cdot \left(\prod_{\mathsf{id}\in\mathcal{P}^*} h_{\mathsf{id}}^{\mathrm{x}s_{\mathsf{id},\mu}} \right),
$$

where

$$
\begin{aligned}
M^* &:= \prod_{\mathsf{id}\in\mathcal{P}^*} (\mathrm{Y}^{(1)}_{\mathsf{id},\mu})^{c_{\mathsf{id}}} = \prod_{\mathsf{id}\in\mathcal{P}^*} g_1^{\mathrm{x}s_{\mathsf{id},\mu}c_{\mathsf{id}}},\\
\Omega_\mu^* &:= \prod_{\mathsf{id}\in\mathcal{P}^*} (\Omega_{\mathsf{id},\mu})^{d_{\mathsf{id}}} = \prod_{\mathsf{id}\in\mathcal{P}^*} (v^{s_{\mathsf{id},\mu}} \cdot g_1^{\omega_{\mathsf{id},\mu}})^{d_{\mathsf{id}}},\\
\Omega^* &:= \prod_{\mathsf{id}\in\mathcal{P}^*} (\Omega_{\mathsf{id},\mu})^{\hat{r}_{\mathsf{id}}} = \prod_{\mathsf{id}\in\mathcal{P}^*} (v^{s_{\mathsf{id},\mu}} \cdot g_1^{\omega_{\mathsf{id},\mu}})^{\hat{r}_{\mathsf{id}}},\\
h_{\mathsf{id}} &:= \prod_{i\in\mathsf{I}_{\mathsf{id}}} \mathcal{H}_1(\Delta, \ell_i, \mathsf{pk}_{\mathsf{id}})^{f_i^*} = \prod_{i\in\mathsf{I}_{\mathsf{id}}} \left(v^{-t_{v,i}} g_1^{t_{g,i}} \right)^{f_i^*}.
\end{aligned}
$$

Abstracting away from the variable names Γ^* has the following structure: $\Gamma^* = v^{\diamond+\square} g_1^{\clubsuit+\spadesuit}$, where $\diamond$ and $\clubsuit$ collect the exponents linked to all non-target datasets/identities/labels and $\square$ and $\spadesuit$ correspond to exponents linked to the target datasets/identities/labels, where target and non target datasets/identities/labels will be defined momentarily for the different types of forgeries. Our aim is to extract from Γ^* its v component *i.e.,* $v^{\mathrm{x}\triangle}$ where $\triangle$ is a known non-zero value.

Type-1 Forgeries The reduction can detect if the adversary outputs a type-1 forgery by checking if the dataset identifier Δ^* defined in the labelled program $\mathcal{P}^*$ was queried to the sign oracle, *i.e.,* if $(\Delta^*, \cdot, \cdot) \in L_\sigma$. If that is not the case, then $\mathscr{B}^{\mathsf{coCDH}^*}$ extracts from the forgery a witness to the co-CDH* instance as follows. Note that this condition implies that the $r^*, \hat{\sigma}_r^* = (\Gamma^*, R^*, \Omega^*, \tilde{\pi}_R^*, \tilde{\pi}_\Omega^*)$ part of the adversary's output is a type-1 forgery against the HUFRMA game. As discussed in the paragraph **Abort Probability** the reduction aborts whenever $\hat{\sigma}_r^*$ is such a forgery (condition (2)). Hence, this execution path is never taken by the reduction.

Type-2 Forgeries Let $\hat{\mu}_{\mathsf{id}} := \sum_{i\in\mathsf{I}_{\mathsf{id}}} \mu_i f_i^*$, then the reduction can detect which identities the adversary targeted by checking if either c_{id} or d_{id} equals $\hat{\mu}_{\mathsf{id}}$. Next we define the set of targeted identities to be, $\mathsf{ID}_{\mathsf{target}} := \{\mathsf{id} \in \mathcal{P}^* : c_{\mathsf{id}} \vee d_{\mathsf{id}} \neq \hat{\mu}_{\mathsf{id}}\}$. The reduction $\mathscr{B}^{\mathsf{coCDH}^*}$ looks for type-2 forgeries against the identities $\mathsf{id} \in \mathsf{ID}_{\mathsf{target}}$, i.e,

forgeries that satisfy

$$\begin{cases} \mathsf{id} \in \mathsf{ID}_{\mathsf{target}} \subset \mathcal{P}^* \\ \forall \mathsf{id} \in \mathsf{ID}_{\mathsf{target}} \ : \ c_{\mathsf{id}} \neq \hat{\mu}_{\mathsf{id}} \vee d_{\mathsf{id}} \neq \hat{\mu}_{\mathsf{id}} \end{cases} \tag{14}$$

Remark that since $\mathscr{A}$ outputs a type-2 forgery it follows that $\hat{\mu}^* := \sum_{\mathsf{id}\in\mathcal{P}^*} d_{\mathsf{id}} \neq \sum_{\mathsf{id}\in\mathcal{P}^*} \hat{\mu}_{\mathsf{id}}$, thus there must exist at least one $\mathsf{id} \in \mathsf{ID}_{\mathsf{target}}$ such that $d_{\mathsf{id}} \neq \hat{\mu}_{\mathsf{id}}$, and analogously for c_{id}.

If the adversary output a type-2 forgery, $\mathscr{B}^{\text{coCDH}^*}$ successfully extracts an answer to the co-CDH* challenger from the type-2 forgery against the target identities, $\mathsf{ID}_{\mathsf{target}}$.

By definition $c_{\mathsf{id}} = d_{\mathsf{id}} = \hat{\mu}_{\mathsf{id}}$ for all non-target identities, $\mathsf{id} \in \mathcal{P}^* \backslash \mathsf{ID}_{\mathsf{target}}$. Thus, $\diamondsuit = 0$ and $\clubsuit \cdot \mathsf{x}^{-1} = \sum_{\mathsf{id}\in\mathcal{P}^*\backslash\mathsf{ID}_{\mathsf{target}}} s_{\mathsf{id},\mu}\big(\theta_\mu \hat{\mu}_{\mathsf{id}} + \sum_{i\in\mathsf{I}_{\mathsf{id}}} t_{g,i} f_i^*\big) + \psi_\mu \omega_{\mathsf{id},\mu}(\hat{\mu}_{\mathsf{id}} + \hat{r}_{\mathsf{id}})$. On the other hand, for the target identities it holds that, $\spadesuit \cdot \mathsf{x}^{-1} = \sum_{\mathsf{id}\in\mathsf{ID}_{\mathsf{target}}} s_{\mathsf{id},\mu}\big(\theta_\mu c_{\mathsf{id}} + \sum_{i\in\mathsf{I}_{\mathsf{id}}} t_i f_i^*\big) + \psi_\mu \omega_{\mathsf{id},\mu}(d_{\mathsf{id}} + \hat{r}_{\mathsf{id}})$. The value of $\clubsuit$ and $\spadesuit$ are known to the reduction, thus $\mathscr{B}^{\text{coCDH}^*}$ can remove the "g_1 component" from Γ^* by computing

$$\Gamma_0 := \frac{\Gamma^*}{g_1^{\clubsuit+\spadesuit}}.$$

At this point we are left with $\Gamma_0 := v^{\square}$, where $\square := \mathsf{x} \cdot \triangle$ and $\triangle = \sum_{\mathsf{id}\in\mathcal{P}^*\backslash\mathsf{ID}_{\mathsf{target}}} s_{\mathsf{id},\mu}\psi_\mu \left(d_{\mathsf{id}} - \sum_{i\in\mathsf{I}_{\mathsf{id}}} \mu_i f_i^*\right)$, the expression is obtained by expanding $\hat{r}_{\mathsf{id}} := \sum r_i f_i^*$ where r_i is as defined in Equation (12). It holds that $\triangle \neq 0$ by the conditions stated in System (14). Finally, to extract the solution to the co-CDH* challenger, $\mathscr{B}$ simply computes

$$\Gamma_1 := \Gamma_0^{\left(\triangle^{-1}\right)} := v^{\mathsf{x}}.$$

Type-3 Forgeries The reduction can detect which labels the adversary targeted by checking which labels $\ell_i^* \in \mathcal{P}^*$ was not queried to the sign oracle, *i.e.,* $(\Delta^*, \ell_i^*, \cdot, \cdot) \notin L_\sigma$. Note that this condition is implies that $\hat{\sigma}_r^* = (\Gamma^*, R^*, \Omega^*, \tilde{\pi}_R^*, \tilde{\pi}_\Omega^*)$ is a type-3 forgery against HUFRMA. As discussed in the **Abort Probability** paragraph, condition (2) makes the reduction abort this happens. Hence, this execution path is never taken by the reduction.

Success Probability. The success probability of $\mathscr{B}^{\text{coCDH}^*}$ in winning the coCDH* game can be bounded by the minimum success probability depending on the type of forgery, that is,

$$\begin{aligned} &\mathsf{Adv}^{\text{coCDH}^*}_{\mathbb{G}_1,\mathbb{G}_2}(\mathscr{B}^{\text{coCDH}^*}, \lambda) \geq \\ &\left(1 - \frac{1}{p}\right) \cdot \mathsf{Adv}^{\mathsf{HUFCMA}}_{\mathsf{mkhs}}(\mathscr{A}, \lambda) \cdot \left(1 - \Pr[\mathscr{B}^{\text{coCDH}^*} \ \textit{aborts}]\right) \\ &\geq \frac{1}{2}\mathsf{Adv}^{\mathsf{HUFCMA}}_{\mathsf{mkhs}} \cdot \Bigg(1 - \Big(\mathsf{Adv}^{\mathsf{wHUFCMA}}_{\mathsf{mkhs}}(\mathscr{A}, \lambda) + \\ &\qquad + \mathsf{Adv}^{\mathsf{ip-ext}}_{\widetilde{\mathsf{InPrd}},\mathsf{R}^{\widetilde{\mathsf{InPrd}}}}(\widetilde{\mathcal{P}_{\mathsf{alg}}}, \mathcal{E}, \lambda)\Big)\Bigg) \end{aligned}$$

where we used the bound $\left(1 - \frac{1}{p}\right) \geq \frac{1}{2}$ and Equation (13).