This is the full version of a paper which appears in Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7 2017, Revised Selected Papers, Springer, LNCS.

Homomorphic Proxy Re-Authenticators and Applications to Verifiable Multi-User Data Aggregation

David Derler[‡], Sebastian Ramacher[‡,‖], and Daniel Slamanig[‡]

IAIK, Graz University of Technology, Graz, Austria
{dderler,sramacher,dslamanig}@iaik.tugraz.at

Abstract. We introduce the notion of *homomorphic proxy re-authenticators*, a tool that adds security and verifiability guarantees to multi-user data aggregation scenarios. It allows distinct sources to authenticate their data under their own keys, and a proxy can transform these single signatures or message authentication codes (MACs) to a MAC under a receiver's key without having access to it. In addition, the proxy can evaluate arithmetic circuits (functions) on the inputs so that the resulting MAC corresponds to the evaluation of the respective function. As the messages authenticated by the sources may represent sensitive information, we also consider hiding them from the proxy and other parties in the system, except from the receiver.

We provide a general model and two modular constructions of our novel primitive, supporting the class of linear functions. On our way, we establish various novel building blocks. Most interestingly, we formally define the notion and present a construction of *homomorphic proxy re-encryption*, which may be of independent interest. The latter allows users to encrypt messages under their own public keys, and a proxy can re-encrypt them to a receiver's public key (without knowing any secret key), while also being able to evaluate functions on the ciphertexts. The resulting re-encrypted ciphertext then holds an evaluation of the function on the input messages.

1 Introduction

Proxy re-cryptography [BBS98] is a powerful concept which allows proxies to transform cryptographic objects under one key to cryptographic objects under another key using a transformation key (a so called re-key). In particular, proxy re-encryption has shown to be of great practical interest in cloud scenarios such as data storage [CD16, BBL16], data sharing [XXW+16], publish-subscribe [BGP+16] as well as cloud-based identity management [NAL12, NA14, ZSSH14, SSZ14]. In contrast, other proxy re-primitives, and in particular proxy re-signatures (or MACs), seem to unleash their full potential not before considering them in combination with homomorphic properties on the message space.

[‡] Supported by EU H2020 project PRISMACLOUD, grant agreement n°644962.
[‖] Supported by EU H2020 project CREDENTIAL, grant agreement n°653454.

Interestingly, however, this direction has received no attention so far. To this end, we introduce the notion of homomorphic proxy re-authenticators (HPRAs), which allows distinct senders to authenticate data under their own keys, and an evaluator (aggregator) can transform these single signatures or message authentication codes (MACs) to a MAC under a receiver's key without knowing it. Most importantly, the aggregator can evaluate arithmetic circuits (functions) on the inputs so that the resulting MAC corresponds to the evaluation of the respective function. Furthermore, we investigate whether we can hide the input messages from the aggregator. On the way to solve this, we formally define the notion of homomorphic proxy re-encryption (HPRE). Data aggregation is the central application of our framework, but it is not limited to this application.

Motivation. Data aggregation is an important task in the Internet of Things (IoT) and cloud computing. We observe a gap in existing work as the important issue of end-to-end authenticity and verifiability of computations on the data (aggregation results) is mostly ignored. We address this issue and propose a versatile non-interactive solution which is tailored to a multi-user setting. The additional authenticity features of our solution add robustness to errors occurring during transmission or aggregation even in the face of a non-trusted aggregator.

Multi-User Data Aggregation. Assume a setting where n senders, e.g., sensor nodes, regularly report data to some entity denoted the aggregator. The aggregator collects the data and then reports computations (evaluations of functions) on these data to a receiver. For example, consider environmental monitoring of hydroelectric plants being located in a mountainous region, where small sensors are used for monitoring purposes. Due to the lack of infrastructure (e.g., very limited cell coverage) sensors are not directly connected to the Internet and collected data is first sent to a gateway running at the premise of some telecommunication provider. This gateway aggregates the data and forwards it to some cloud service operated by the receiver.

Obviously, when the involved parties communicate via public networks, security related issues arise. Apart from achieving security against outsiders, there are also security and privacy related issues with respect to the involved parties.

In general, we identify three main goals. (1) End-to-end authenticity, i.e., protecting data items from unauthorized manipulation and preserving the source authenticity. (2) Concealing the original data from the aggregator and the receiver, and, even further, concealing the result of the computation from the aggregator. Clearly, in (2) we also want to conceal data from any outsider. (3) Establishing independent secret keys for the involved parties so that they do not share a single secret. Latter facilitates a dynamic setting.

Below, we present such an aggregation scenario, discuss why straightforward solutions fall short, and sketch our solution. Then, we discuss the problems popping up when we require stronger privacy guarantees and show how our primitives help to overcome these issues.

Authenticity & Input Privacy. In our first scenario, the n senders each hold their own signing key and within every period sender i reports a signed data item d_i to the aggregator. The aggregator must be able to evaluate functions $f \in \mathcal{F}$ (where

$\mathcal{F}$ is some suitable class of functions, e.g., linear functions) on $d_1, \ldots, d_n$ so that a receiver will be convinced of the authenticity of the data and the correctness of the computation without fully trusting the aggregator (recall the end-to-end authenticity requirement). Moreover, although the inputs to the aggregator are not private, we still want them to be hidden relative to the function f, i.e., so that a receiver only learns what is revealed by f and $\hat{d} = f(d_1, \ldots, d_n)$, as a receiver might not need to learn the single input values.

A central goal is that the single data sources have individual keys. Thus, we can not directly employ homomorphic signatures (or MACs). Also the recent concept of multikey-homomorphic signatures [FMNP16, DS16, LTWC16] does not help: even though they allow homomorphic operations on the key space, they do not consider transformations to some specific target key.[1] With HPRAs we can realize this, as the aggregator (who holds re-keys from the senders to some receiver) can transform all the single signatures or MACs to a MAC under the receiver's key (without having access to it). Moreover, due to the homomorphic property, a MAC which corresponds to the evaluation of a function f on the inputs can be computed. The receiver can then verify the correctness of the computation, i.e., that $\hat{d} = f(d_1, \ldots, d_n)$, and the authenticity of the used inputs (without explicitly learning them) using its independent MAC key.

Adding Output Privacy. In our second scenario, we additionally want data privacy guarantees with respect to the aggregator. This can be crucial if the aggregator is running in some untrusted environment, e.g., the cloud. We achieve this by constructing an output private HPRA. In doing so, one has to answer the question as how to confidentially provide the result of the computation to the receiver and how to guarantee the authenticity (verifiability) of the computation. We tackle this issue by introducing a HPRE where the homomorphism is compatible to the one of the HPRA. The sources then additionally encrypt the data under their own keys and the aggregator re-encrypts the individual ciphertexts to a ciphertext under a receiver's key and evaluates the same function f as on the MACs on the ciphertexts. This enables the receiver to decrypt the result $\hat{d}$ using its *own* decryption key and to verify the MAC on $\hat{d}$ together with a description of the function f. In addition, we use a trick to prevent public verifiability of the signatures from the single data sources, as public verifiability potentially leaks the signed data items which trivially would destroy output privacy.

Contribution. Our contributions in this paper can be summarized as follows.

– We introduce the notion of homomorphic proxy re-authenticators (HPRA). Our framework tackles multi-user data aggregation in a dynamic setting. For the first time, we thereby consider independent keys of the single parties, the verifiability of the evaluation of general functions on the authenticated inputs by the sources, as well as privacy with respect to the aggregator.

[1] While the homomorphic properties might allow one to define a function mapping to a target key, it is unclear whether handing over the description of such a function to a proxy would maintain the security requirements posed by our application.

– As a means to achieve the strong privacy requirements imposed by our security model, we formally define the notion of homomorphic proxy re-encryption (HPRE), which may be of independent interest.
– We present two modular constructions of HPRA schemes for the class $\mathcal{F}_{\mathsf{lin}}$ of linear functions, which differ regarding the strength of the provided privacy guarantees. On our way, we establish various novel building blocks. Firstly, we present a linearly homomorphic MAC which is suitable to be used in our construction. Secondly, to achieve the stronger privacy guarantees, we construct a HPRE scheme for linear functions. All our proofs are modular in the sense that we separately prove the security of our building blocks; our overall proofs then build upon the results obtained for the building blocks. Thus, our building blocks may as well easily be used in other constructions.

Related Work. Subsequently, we review related work. As our focus is on non-interactive approaches, we omit interactive approaches where clients download all the data, decrypt them locally, compute a function, and send the results back along with a zero-knowledge proof of correctness (as, e.g., in [DL11]).

Proxy Re-Cryptography. Proxy re-encryption (PRE) [BBS98] allows a semi-trusted proxy to transform a message encrypted under the key of some party into a ciphertext to the same message under a key of another party, where the proxy performing the re-encryption learns nothing about the message. This primitive has been introduced in [BBS98], further studied in [ID03] and the first strongly secure constructions have been proposed by Ateniese et al. in [AFGH06]. Boneh et al. construct PRE in the symmetric setting [BLMR13]. Follow-up work focuses on even stronger (IND-CCA2 secure) schemes (cf. [CH07,LV11,NAL15,NAL16]). Since we, however, require certain homomorphic properties, we focus on IND-CPA secure schemes (as IND-CCA2 security does not allow any kind of malleability). In previous work by Ayday et al. [ARHR13], a variant of the linearly homomorphic Paillier encryption scheme and proxy encryption in the sense of [ID03] were combined. Here, the holder of a key splits the key and gives one part to the proxy and one to the sender; with the drawback that the secret key is exposed when both collude. We are looking for proxy re-encryption that is homomorphic, works in a multi-user setting but is collusion-safe and non-interactive, i.e., re-encryption keys can be computed by the sender using only the public key of the receiver without any interaction and a collusion of sender and proxy does not reveal the receiver's key. Also note that, as our focus is on practically efficient constructions, we do not build upon fully homomorphic encryption [Gen09], which allows to build HPRE using the rather expensive bootstrapping technique. In concurrent work Ma et al. [MLO16] follow this approach and propose a construction of a PRE scheme with homomorphic properties which additionally achieves key privacy. They build upon [GSW13] using the bootstrapping techniques in [AP14] and apply some modifications for key privacy. While their construction can be seen as a HPRE in our sense, they do not formally define a corresponding security model and we are not aware of a suitable formalization for our purposes.

Proxy re-signatures, i.e., the signature analogue to proxy re-encryption, have been introduced in [BBS98] and formally studied in [ID03]. Later, [AH05] in-

troduced stronger security definitions, constructions and briefly discussed some applications. However, the schemes in [AH05] and follow up schemes [LV08] do not provide a homomorphic property and it is unclear how they could be extended. The concept of *homomorphic proxy re-authenticators*, which we propose, or a related concept, has to the best of our knowledge not been studied before.

Homomorphic Authenticators. General (non-interactive) verifiable computing techniques (cf. [WB15] for a recent overview) are very expressive, but usually prohibitive regarding proof computation (proof size and verification can, however, be very small and cheap respectively). In addition, the function and/or the data needs to be fixed at setup time and inputs are not authenticated. Using homomorphic authenticators allows evaluations of functions on authenticated inputs under a single key (cf. [Cat14] for a recent overview). They are dynamic with respect to the authenticated data and the evaluated function, and also efficient for interesting classes of functions. Evaluating results is typically not more efficient than computing the function (unless using an amortized setting [BFR13, CFW14]). Yet, they provide benefits when saving bandwidth is an issue and/or the inputs need to be hidden from evaluators (cf. [LDPW14, CMP14]). Computing on data authenticated under different keys using so called multi-key homomorphic authenticators [FMNP16, DS16, LTWC16], has only very recently been considered. Even though they are somewhat related, they are no replacement for what we are proposing in this paper.

Aggregator-Oblivious Encryption (AOE). AOE [RN10, SCR+11] considers data provided by multiple producers, which is aggregated by a semi-honest aggregator. The aggregator does not learn the single inputs but only the final result. Follow-up work [JL13, LEM14, BJL16] improved this approach in various directions. Furthermore, [CSS12] introduced a method to achieve fault tolerance, being applicable to all previous schemes. There are also other lines of work on data aggregation, e.g., [LC13, CCMT09], [LCP14, GMP14]. Very recently, [LEÖM15] combined AOE with homomorphic tags to additionally provide verifiability of the aggregated results. Here, every user has a tag key and the aggregator additionally aggregates the tags. Verification can be done under a pre-distributed combined fixed tag key. Their approach is limited to a single function (the sum) and requires a shared secret key-setting, which can be problematic.

In all previous approaches it is impossible to hide the outputs (i.e., the aggregation results) from the aggregator. In contrast to only hiding the inputs, we additionally want to hide the outputs. In addition, we do not want to assume a trusted distribution of the keys, but every sender should authenticate and encrypt under his own key and the aggregator can then perform re-operations (without any secret key) to the receiver.

2 Preliminaries

Unless stated otherwise, all algorithms run in polynomial time and return a special symbol $\perp$ on error. By $y \leftarrow \mathcal{A}(x)$, we denote that y is assigned the output of the potentially probabilistic algorithm $\mathcal{A}$ on input x and fresh random

coins (we may also use sans serif font to denote algorithms). Similarly, $y \xleftarrow{R} S$ means that y is assigned a uniformly random value from a set S. If a and b are strings, $a\|b$ is the concatenated string and $\vec{a}\|b$ means extending the vector $\vec{a}$ with element b. For a sequence of vectors $(\vec{v}_i)_{i\in[n]}$ of length ℓ, we use $f((\vec{v}_i)_{i\in[n]})$ to denote the element-wise application of the function f, i.e., $f((\vec{v}_i)_{i\in[n]}) \coloneqq (f(v_{i1})_{i\in[n]}, \ldots, f((v_{i\ell})_{i\in[n]}))$. We let $[n] \coloneqq \{1, \ldots, n\}$ and let $\Pr[\Omega : E]$ denote the probability of an event E over the probability space Ω. A function $\varepsilon(\cdot) : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is called negligible, iff it vanishes faster than every inverse polynomial, i.e., $\forall\, k : \exists\, n_k : \forall\, n > n_k : \varepsilon(n) < n^{-k}$. A polynomial function is denoted by $\mathsf{poly}(\cdot)$.

Let $\mathbb{G}_1 = \langle g \rangle$, $\mathbb{G}_2 = \langle \hat{g} \rangle$, and $\mathbb{G}_T$ be cyclic groups of prime order q. A paring $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable, bilinear, non-degenerate map. For simplicity we present our results in the (symmetric) Type-1 setting where $\mathbb{G}_1 = \mathbb{G}_2$. We stress that there are tools [AGH15, AHO16] to automatically translate them to the more efficient (asymmetric) Type-3 setting. Henceforth we use BG to denote a description of a bilinear group and use boldface letters to denote elements in $\mathbb{G}_T$. We formally define bilinear group generation and the required computational hardness assumptions in Appendix A.1.

Linearly Homomorphic MACs (HOM-MACs). Our subsequent definition is inspired by [AB09]. Let $\mathcal{F}_{\mathsf{lin}}$ denote the class of linear functions.

Definition 1 (HOM-MAC). *A* HOM-MAC *is a tuple* $(\mathcal{P}, \mathcal{G}, \mathcal{S}, \mathcal{C}, \mathcal{V})$ *of algorithms defined as:*

$\mathcal{P}(\kappa, \ell)$: *Takes a security parameter* κ *and an upper bound* ℓ *on the vector length as input and outputs public parameters* pp.

$\mathcal{G}(\mathsf{pp})$: *Takes the public parameters* pp *as input and outputs a secret key* sk.

$\mathcal{S}(\mathsf{sk}, \vec{v}, \mathsf{id}, \tau)$: *Takes a MAC key* sk*, a vector* $\vec{v}$*, an identifier* id*, and a tag* τ *as input, and outputs a MAC* μ.

$\mathcal{C}(f, (\mu_i)_{i\in[n]})$: *Takes a function* $f \in \mathcal{F}_{\mathsf{lin}}$ *and a sequence of valid MACs* $(\mu_i)_{i\in[n]}$ *on vectors* $(\vec{v}_i)_{i\in[n]}$ *as input, and outputs a MAC* μ *on* $\vec{v} = f((\vec{v}_i)_{i\in[n]})$.

$\mathcal{V}(\mathsf{sk}, \vec{v}, \mu, \tau, (\mathsf{id}_i)_{i\in[n]}, f)$: *Takes a MAC key* sk*, a vector* $\vec{v}$*, a MAC* μ*, a tag* τ*, a sequence of identifiers* $(\mathsf{id}_i)_{i\in[n]}$*, and a function* $f \in \mathcal{F}_{\mathsf{lin}}$ *as input, and outputs a bit.*

A linearly homomorphic MAC is required to be correct and unforgeable. We postpone the formal definitions to Appendix A.2.

Proxy Re-Encryption. A proxy re-encryption (PRE) scheme is an encryption scheme that allows a proxy to transform a message m encrypted under public key rpk_A of party A into a ciphertext to m under rpk_B for another party B, so that the proxy learns nothing about m. A PRE scheme is called *non-interactive* if party A can produce a re-encryption key from A to B locally by having access to its private key and only B's public key, *collusion-safe* if the proxy colluding with either of the parties can not recover the other parties private key, *unidirectional* if a re-encryption key only allows transformations in one direction (e.g., from A to B), and *single-use* if one ciphertext can be transformed only once. For our formal definitions, we largely follow [AFGH06].

Definition 2 (PRE). *A* PRE *is a tuple* $(\mathcal{P}, \mathcal{G}, \vec{\mathcal{E}}, \vec{\mathcal{D}}, \mathcal{RG}, \mathcal{RE})$ *of algorithms, where* $\vec{\mathcal{E}} = (\mathcal{E}^i)_{i\in[2]}$ *and* $\vec{\mathcal{D}} = (\mathcal{D}^i)_{i\in[2]}$*, which are defined as follows:*

$\mathcal{P}(1^\kappa)$: *Takes a security parameter* κ *and outputs parameters* pp.
$\mathcal{G}(\mathsf{pp})$: *Takes parameters* pp *and outputs a key pair* $(\mathsf{rsk}, \mathsf{rpk})$.
$\mathcal{RG}(\mathsf{rsk}_A, \mathsf{rpk}_B)$: *Takes a secret key* rsk_A *and a public key* rpk_B *and outputs a re-encryption key* $\mathsf{rk}_{A\to B}$.
$\mathcal{E}^i(\mathsf{rpk}, m)$: *Takes a public key* rpk *and a message* m *and outputs a ciphertext* c.
$\mathcal{RE}(\mathsf{rk}_{A\to B}, c_A)$: *Takes a re-encryption key* $\mathsf{rk}_{A\to B}$ *and a ciphertext* c_A *under* rpk_A*, and outputs a re-encrypted ciphertext* c_B *for* rpk_B.
$\mathcal{D}^i(\mathsf{rsk}, c)$: *Takes a secret key* rsk *and a ciphertext* c*, and outputs* m.

A PRE scheme needs to be correct. This notion requires that for all security parameters $\kappa \in \mathbb{N}$, all honestly generated parameters $\mathsf{pp} \leftarrow \mathcal{P}(1^\kappa)$, all key pairs $(\mathsf{rsk}_A, \mathsf{rpk}_A) \leftarrow \mathcal{G}(\mathsf{pp})$, $(\mathsf{rsk}_B, \mathsf{rpk}_B) \leftarrow \mathcal{G}(\mathsf{pp})$, all re-encryption keys $\mathsf{rk}_{A\to B} \leftarrow \mathcal{RG}(\mathsf{rsk}_A, \mathsf{rpk}_B)$, all messages m it holds with probability one that

$$\forall\, i \in [2]\ \exists\, j \in [2]\ :\ \mathcal{D}^j(\mathsf{rsk}_A, \mathcal{E}^i(\mathsf{rpk}_A, m)) = m, \text{ and}$$
$$\exists\, i \in [2]\ \exists\, j \in [2]\ :\ \mathcal{D}^j(\mathsf{rsk}_B, \mathcal{RE}(\mathsf{rk}_{A\to B}, \mathcal{E}^i(\mathsf{pk}_A, m))) = m.$$

Thereby i and j determine the level of the ciphertexts. We will henceforth use the following semantics: first-level ciphertexts ($\mathcal{E}^1$) cannot be re-encrypted by a proxy, whereas second-level ciphertexts ($\mathcal{E}^2$) can be re-encrypted.

In addition, a PRE needs to be IND-CPA secure. We, henceforth, only require a relaxed IND-CPA notion which we term IND-CPA$^-$. It is clearly implied by the original IND-CPA notion from [AFGH06] (some oracles are omitted and the adversary only gets to see a second-level ciphertext).

Definition 3 (IND-CPA$^-$). *A* PRE *is* IND-CPA$^-$ *secure, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\varepsilon(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathcal{P}(1^\kappa), b \xleftarrow{R} \{0,1\}, (\mathsf{sk}_t, \mathsf{pk}_t) \leftarrow \mathcal{G}(\mathsf{pp}),\\ (\mathsf{sk}_h, \mathsf{pk}_h) \leftarrow \mathcal{G}(\mathsf{pp}), \mathsf{rk}_{t\to h} \leftarrow \mathcal{RG}(\mathsf{sk}_t, \mathsf{pk}_h),\\ (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}_t, \mathsf{pk}_h, \mathsf{rk}_{t\to h}),\\ c \leftarrow \mathcal{E}^2(m_b, \mathsf{pk}_t), b^\star \leftarrow \mathcal{A}(\mathsf{st}, c)\end{array} : b = b^\star\right] \leq 1/2 + \varepsilon(\kappa).$$

We remark that $\mathcal{RG}$ as defined in [AFGH06] also takes the target secret key to cover interactive schemes. As we only deal with non-interactive ones, we omit it.

3 Homomorphic Proxy Re-Authenticators

We introduce homomorphic proxy re-authenticators (HPRAs) and rigorously formalize a suitable security model. Our goal is to obtain a flexible framework with various possible instantiations. Accordingly, our definitions are rather generic. We stress that both the source and receiver re-key generation, besides the secret key of the executing party, only require public inputs, i.e., are non-interactive.

Definition 4 (HPRA). *A homomorphic proxy re-authenticator* (HPRA) *for class* $\mathcal{F}$ *of functions is a tuple of PPT algorithms* (Gen, SGen, VGen, Sign, Verify, SRGen, VRGen, Agg, AVerify)*, where* Verify *is optional. They are defined as follows:*

$\mathsf{Gen}(1^\kappa, \ell)$: *Takes security parameter* κ *and vector length* ℓ *and outputs parameters* pp.

$\mathsf{SGen}(\mathsf{pp})$: *Takes parameters* pp *as input, and outputs a signer key* $(\mathsf{id}, \mathsf{sk}, \mathsf{pk})$.

$\mathsf{VGen}(\mathsf{pp})$: *Takes parameters* pp*, and outputs a MAC key* mk *and auxiliary information* aux.

$\mathsf{Sign}(\mathsf{sk}, \vec{m}, \tau)$: *Takes a signer secret key* sk*, a message vector* $\vec{m}$*, and a tag* τ *as input, and outputs a signature* σ.

$\mathsf{Verify}(\mathsf{pk}, \vec{m}, \tau, \sigma)$: *Takes a signer public key* pk*, a message vector* $\vec{m}$*, a tag* τ*, and a signature* σ *as input, and outputs a bit* b.

$\mathsf{SRGen}(\mathsf{sk}_i, \mathsf{aux})$: *Takes a signer secret key* sk_i*, some auxiliary information* aux*, and outputs a re-encryption key* rk_i.

$\mathsf{VRGen}(\mathsf{pk}_i, \mathsf{mk}, \mathsf{rk}_i)$: *Takes a signer public key* pk_i *and a MAC key* mk*, as well as a re-encryption key* rk_i *as input, and outputs an aggregation key* ak_i.

$\mathsf{Agg}((\mathsf{ak}_i)_{i\in[n]}, (\sigma_i)_{i\in[n]}, \tau, f)$: *Takes* n *aggregation keys* $(\mathsf{ak}_i)_{i\in[n]}$*,* n *signatures* $(\sigma_i)_{i\in[n]}$*, a tag* τ*, and a function* $f \in \mathcal{F}$ *as input, and outputs an aggregate authenticated message vector* Λ.

$\mathsf{AVerify}(\mathsf{mk}, \Lambda, \mathsf{ID}, f)$: *Takes a MAC key* mk*, an aggregate authenticated message vector* Λ*,* n *identifiers* $\mathsf{ID} = (\mathsf{id}_i)_{i\in[n]}$*, and a function* $f \in \mathcal{F}$*. It outputs a message vector and a tag* $(\vec{m}, \tau)$ *on success and* $(\perp, \perp)$ *otherwise.*

Security Properties. Below we define the oracles, where the public parameters and the keys generated in the security games are implicitly available to the oracles. While most oracle definitions are fairly easy to comprehend and therefore not explicitly explained, we note that the RoS oracle is used to model the requirement that signatures do not leak the signed data in a real-or-random style. The environment maintains the initially empty sets HU and CU of honest and corrupted users (CU is only set in the output privacy game). Further, it maintains the initially empty sets S, RK and AK of signer, re-encryption and aggregation keys, and an initially empty set SIG of message-identity pairs.

$\mathsf{SG}(i)$: If $\mathtt{S}[i] \neq \perp$ return $\perp$. Otherwise run $(\mathsf{id}_i, \mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{SGen}(\mathsf{pp})$, set $\mathtt{S}[i] \leftarrow (\mathsf{id}_i, \mathsf{sk}_i, \mathsf{pk}_i)$, and, if $i \notin \mathtt{CU}$ set $\mathtt{HU} \leftarrow \mathtt{HU} \cup \{i\}$. Return $(\mathsf{id}_i, \mathsf{pk}_i)$.

$\mathsf{SKey}(i)$: If $i \notin \mathtt{HU}$ return $\perp$. Otherwise return $\mathtt{S}[i]$.

$\mathsf{Sig}((j_i)_{i\in[n]}, (\vec{m}_i)_{i\in[n]})$: If $\mathtt{S}[j_i] = \perp$ for any $i \in [n]$, or there exists $u, v \in [n], u \neq v$ so that $j_u = j_v$, return $\perp$. Otherwise sample a random tag τ and compute $(\sigma_{j_i} \leftarrow \mathsf{Sign}(\mathtt{S}[j_i][2], \vec{m}_i, \tau))_{i\in[n]}$, set $\mathtt{SIG}[\tau] \leftarrow \mathtt{SIG}[\tau] \cup \{(\vec{m}_i, \mathtt{S}[j_i][1])\}$ for $i \in [n]$, and return $(\sigma_{j_i})_{i\in[n]}$ and τ.

$\mathsf{RoS}((j_i)_{i\in[n]}, (\vec{m}_i)_{i\in[n]}, b)$: If $\mathtt{S}[j_i] = \perp$ or $j_i \in \mathtt{CU}$ for any $i \in [n]$ return $\perp$. Otherwise sample τ uniformly at random and if $b = 0$ compute $(\sigma_{j_i} \leftarrow \mathsf{Sign}(\mathtt{S}[j_i][2], \vec{m}_i, \tau))_{i\in[n]}$. Else choose $(\vec{r}_i)_{i\in[n]} \xleftarrow{R} (\mathcal{M}^\ell)^n$ where $\mathcal{M}$ is the message space and compute $(\sigma_{j_i} \leftarrow \mathsf{Sign}(\mathtt{S}[j_i][2], \vec{r}_i, \tau))_{i\in[n]}$. Finally, return $(\sigma_{j_i})_{i\in[n]}$.

$\mathsf{SR}(i)$: If $\mathtt{S}[i] = \perp \vee \mathtt{RK}[i] \neq \perp$ return $\perp$. Else, set $\mathtt{RK}[i] \leftarrow \mathsf{SRGen}(\mathtt{S}[i][2], \mathsf{aux})$ and return $\mathtt{RK}[i]$.

$\mathsf{VR}(i)$: If $\mathtt{S}[i] = \bot \ \vee \ \mathtt{RK}[i] = \bot \ \vee \ \mathtt{AK}[i] \neq \bot$ return $\bot$. Else, set $\mathtt{AK}[i] \leftarrow \mathsf{VRGen}(\mathtt{S}[i][3], \mathsf{mk}, \mathtt{RK}[i])$.

$\mathsf{VRKey}(i)$: Return $\mathtt{AK}[i]$.

$\mathsf{A}((\sigma_{j_i})_{i\in[n]}, (j_i)_{i\in[n]}, \tau, f)$: Check validity of all σ_{j_i}, whether $f \in \mathcal{F}$, whether $\mathtt{SIG}[\tau] = \bot$, and return $\bot$ if any check fails. Further, check whether there exists $u, v \in [n], u \neq v$ so that $j_u = j_v$ and return $\bot$ if so. Obtain $(\mathsf{ak}_{j_i})_{i\in[n]}$ from $\mathtt{AK}$ and return $\bot$ if $\mathtt{AK}[j_i] = \bot$ for any $i \in [n]$. Set $\mathtt{SIG}[\tau] \leftarrow \bigcup_{i\in[n]}\{(\vec{m}_{j_i}, \mathtt{S}[j_i][1])\}$ and return $\Lambda \leftarrow \mathsf{Agg}((\mathsf{ak}_{j_i})_{i\in[n]}, (\sigma_{j_i})_{i\in[n]}, \tau, f)$.

We require a HPRA to be correct, signer unforgeable, aggregator unforgeable, and input private. While we omit the obvious correctness definition, we introduce the remaining notions below. Signer unforgeability requires that, as long as the aggregator remains honest, no coalition of dishonest signers can produce a valid aggregate authenticated message vector Λ with respect to function $f \in \mathcal{F}$ so that Λ is outside of the range of f evaluated on arbitrary combinations of actually signed vectors. Aggregator unforgeability is the natural counterpart of signer unforgeability, where the aggregator is dishonest while the signers are honest.[2]

Definition 5 (T-Unforgeability). *Let* $\mathsf{T} \in \{\mathit{Signer}, \mathit{Aggregator}\}$. *A* HPRA *for class* $\mathcal{F}$ *is* T*-unforgeable, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\epsilon(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Gen}(1^\kappa, \ell),\\ (\mathsf{mk}, \mathsf{aux}) \leftarrow \mathsf{VGen}(\mathsf{pp}),\\ (\Lambda^\star, \mathsf{ID}^\star, f^\star) \leftarrow \mathcal{A}^{\mathcal{O}_\mathsf{T}}(\mathsf{pp}, \mathsf{aux}),\\ (\vec{m}, \tau) \leftarrow \mathsf{AVerify}(\mathsf{mk}, \Lambda^\star, \mathsf{ID}^\star, f^\star)\end{array} : \begin{array}{r}\vec{m} \neq \bot \ \wedge \ f^\star \in \mathcal{F} \ \wedge\\ 0 < n, \ell \leq \mathsf{poly}(\kappa) \ \wedge\\ \left(\nexists \ (\vec{m}_j)_{j\in[n]} : (\forall \ j \in [n] :\right.\\ (\vec{m}_j, \mathsf{id}_j^\star) \in \mathtt{SIG}[\tau]) \ \wedge\\ \left. f^\star((\vec{m}_j)_{j\in[n]}) = \vec{m}\right)\end{array}\right] \leq \varepsilon(\kappa),$$

where $\mathcal{O}_\mathsf{T} := \{\mathsf{SG}(\cdot), \mathsf{SKey}(\cdot), \mathsf{SR}(\cdot), \mathsf{VR}(\cdot), \mathsf{A}(\cdot,\cdot,\cdot)\}$ *for* $\mathsf{T} = \mathit{Signer}$ *and* $\mathcal{O}_\mathsf{T} := \{\mathsf{SG}(\cdot), \mathsf{Sig}(\cdot,\cdot), \mathsf{SR}(\cdot), \mathsf{VR}(\cdot), \mathsf{VRKey}(\cdot)\}$ *for* $\mathsf{T} = \mathit{Aggregator}$.

Input privacy captures the requirement that an aggregate authenticated message vector does not leak more about the inputs to f as the evaluation result and the description of f would leak on their own.

Definition 6 (Input Privacy). *A* HPRA *for class* $\mathcal{F}$ *is input private if for all* $\kappa \in \mathbb{N}$*, for all* $f \in \mathcal{F}$ *implicitly defining* n*, for all tags* τ*, and for all* $(\vec{m}_{11}, \ldots, \vec{m}_{n1})$ *and* $(\vec{m}_{12}, \ldots, \vec{m}_{n2})$ *where* $f(\vec{m}_{11}, \ldots, \vec{m}_{n1}) = f(\vec{m}_{12}, \ldots, \vec{m}_{n2})$*, for all* $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\kappa, \ell)$*, for all* $(\mathsf{mk}, \mathsf{aux}) \leftarrow \mathsf{VGen}(\mathsf{pp})$*, for all* $((\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{SGen}(\mathsf{pp}))_{i\in[n]}, (\mathsf{ak}_i \leftarrow \mathsf{SRGen}(\mathsf{sk}_i, \mathsf{aux}, \mathsf{VRGen}(\mathsf{pk}_i, \mathsf{mk})))_{i\in[n]}$*, the following distributions are identical:*

$$\{\mathsf{Agg}((\mathsf{ak}_i)_{i\in[n]}, (\mathsf{Sign}(\mathsf{sk}_i, \vec{m}_{i1}, \tau))_{i\in[n]}, \tau, f)\},$$
$$\{\mathsf{Agg}((\mathsf{ak}_i)_{i\in[n]}, (\mathsf{Sign}(\mathsf{sk}_i, \vec{m}_{i2}, \tau))_{i\in[n]}, \tau, f)\}.$$

[2] It is impossible to consider both, signers and aggregators, to be dishonest at the same time, as such a coalition could essentially authenticate everything. This is in contrast to the setting of proxy re-encryption, where it makes sense to model security in the face of receivers colluding with the proxy.

Additionally, a HPRA may provide output privacy. It models that the aggregator neither learns the inputs nor the result of the evaluation of f.

Definition 7 (Output Privacy). *A* HPRA *for class $\mathcal{F}$ is output private, if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that:*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Gen}(1^\kappa, \ell), (\mathtt{CU}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}), b \xleftarrow{R} \{0,1\},\\ (\mathsf{mk}, \mathsf{aux}) \leftarrow \mathsf{VGen}(\mathsf{pp}), \mathcal{O} \leftarrow \{\mathsf{SG}(\cdot), \mathsf{SKey}(\cdot),\\ \mathsf{RoS}(\cdot,\cdot,b), \mathsf{SR}(\cdot), \mathsf{VR}(\cdot), \mathsf{VRKey}(\cdot)\},\\ b^* \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{aux}, \mathsf{st})\end{array} : b = b^*\right] \leq 1/2 + \varepsilon(\kappa).$$

4 An Input Private Scheme for Linear Functions

Now we present our first HPRA for the class $\mathcal{F}_{\mathsf{lin}}$ of linear functions. The main challenge we face is to construct a signature scheme with an associated HOM-MAC scheme, where the translation of the signatures under one key to a MAC under some other key works out. Since we believe that our HOM-MAC may as well be useful in other settings we present it as a standalone building block and then proceed with our full construction, where HOM-MAC is used as a submodule. Both build upon the ideas used in the signature scheme presented in [BFKW09].

A Suitable Linearly Homomorphic MAC. We present our HOM-MAC in Scheme 1. We can not recycle the security arguments from [BFKW09] as we require the ability to submit arbitrary tags τ to the Sig oracle (cf. Definition 14 in Appendix A.2). Thus we directly prove unforgeability.

$\underline{\mathcal{P}(\kappa, \ell)}$: Run $\mathsf{BG} \leftarrow \mathsf{BGGen}(1^\kappa)$, fix $H : \mathbb{Z}_q \to \mathbb{G}$, choose $(g_i)_{i\in[\ell]} \xleftarrow{R} (\mathbb{G}^*)^\ell$, and return $\mathsf{pp} \leftarrow (\mathsf{BG}, H, (g_i)_{i\in[\ell]}, \ell)$.

$\underline{\mathcal{G}(\mathsf{pp})}$: Choose $\alpha \xleftarrow{R} \mathbb{Z}_p$ and return $\mathsf{sk} \leftarrow (\mathsf{pp}, \alpha)$.

$\underline{\mathcal{S}(\mathsf{sk}, \vec{v}, \mathsf{id}, \tau)}$: Parse sk as (pp, α) and return $\mu \leftarrow e(H(\tau||\mathsf{id}) \cdot \prod_{j\in[\ell]} g_j^{v_j}, g^\alpha)$.

$\underline{\mathcal{C}(f, (\mu_i)_{i\in[n]})}$: Parse f as $(\omega_i)_{i\in[n]}$ and return $\mu \leftarrow \prod_{i\in[n]} \mu_i^{\omega_i}$.

$\underline{\mathcal{V}(\mathsf{sk}, \vec{v}, \mu, \tau, (\mathsf{id}_i)_{i\in[n]}, f)}$: Parse sk as (pp, α), f as $(\omega_i)_{i\in[n]}$, and output 1 if the following holds, and 0 otherwise: $\mu = e(\prod_{i\in[n]} H(\tau||\mathsf{id}_i)^{\omega_i} \prod_{j\in[\ell]} g_j^{v_j}, g^\alpha)$

Scheme 1: Linearly homomorphic MAC based on [BFKW09].

Lemma 1 (proven in Appendix B.1). *If the bilinear* DDH (BDDH) *assumption holds, then Scheme 1 is an unforgeable* HOM-MAC *in the ROM.*

Our Input Private Construction. In Scheme 2 we present our HPRA construction for the class $\mathcal{F}_{\mathsf{lin}}$. It allows to authenticate vectors of length ℓ, so

that the same function can be evaluated per vector component. In our application scenario we have $\ell = 1$. We allow one to parametrize our construction with an algorithm $\mathsf{Eval}(\cdot,\cdot)$, which defines how to compute $f \in \mathcal{F}_{\mathsf{lin}}$ on the message vector. When directly instantiating Scheme 2, Eval is defined as $\mathsf{Eval}(f, (\vec{m}_i)_{i\in[n]}) \coloneqq f((\vec{m}_i)_{i\in[n]})$.

$\underline{\mathsf{Gen}(1^\kappa, \ell)}$: Run $\mathsf{BG} \leftarrow \mathsf{BGGen}(1^\kappa)$, fix $H : \mathbb{Z}_q \to \mathbb{G}$, choose $(g_i)_{i\in[\ell]} \xleftarrow{R} \mathbb{G}^\ell$, and return $\mathsf{pp} \leftarrow (\mathsf{BG}, H, (g_i)_{i\in[\ell]}, \ell)$.

$\underline{\mathsf{SGen}(\mathsf{pp})}$: Choose $\beta \xleftarrow{R} \mathbb{Z}_q$, set $\mathsf{id} \leftarrow g^\beta$, $\mathsf{pk} \leftarrow (\mathsf{pp}, g^\beta, g^{1/\beta})$, $\mathsf{sk} \leftarrow (\mathsf{pk}, \beta)$, and return $(\mathsf{id}, \mathsf{sk}, \mathsf{pk})$.

$\underline{\mathsf{VGen}(\mathsf{pp})}$: Choose $\alpha \xleftarrow{R} \mathbb{Z}_q$, set $\mathsf{aux} \leftarrow \emptyset$, $\mathsf{mk} \leftarrow (\mathsf{pp}, \alpha)$ and return $(\mathsf{mk}, \mathsf{aux})$.

$\underline{\mathsf{Sign}(\mathsf{sk}, \vec{m}, \tau)}$: Parse sk as $(((\mathsf{BG}, H, (g_i)_{i\in[\ell]}, \ell), g^\beta, \cdot), \beta)$, compute and return $\sigma \leftarrow (\sigma', \vec{m})$, where

$$\sigma' \leftarrow \left(H(\tau||g^\beta) \cdot \textstyle\prod_{i=1}^{\ell} g_i^{m_i}\right)^\beta.$$

$\underline{\mathsf{Verify}(\mathsf{pk}, \vec{m}, \tau, \sigma)}$: Parse pk as $((\mathsf{BG}, H, (g_i)_{i\in[\ell]}, \ell), g^\beta, \cdot)$, and σ as $(\sigma', \vec{m}')$, and return 1 if the following holds and 0 otherwise:

$$e(H(\tau||g^\beta) \cdot \textstyle\prod_{i=1}^{\ell} g_i^{m_i}, g^\beta) = e(\sigma, g) \quad \wedge \quad \vec{m} = \vec{m}'.$$

$\underline{\mathsf{SRGen}(\mathsf{sk}_i, \mathsf{aux})}$: Return $\mathsf{rk}_i \leftarrow \emptyset$.

$\underline{\mathsf{VRGen}(\mathsf{pk}_i, \mathsf{mk}, \mathsf{rk}_i)}$: Parse pk_i as $(\cdot, \cdot, g^{1/\beta_i})$, mk as $(\cdot, \alpha)$, and return $\mathsf{ak}_i \leftarrow (g^{1/\beta_i})^\alpha$.

$\underline{\mathsf{Agg}((\mathsf{ak}_i)_{i\in[n]}, (\sigma_i)_{i\in[n]}, \tau, f)}$: Parse f as $(\omega_i)_{i\in[n]}$, and for $i \in [n]$ parse σ_i as $(\sigma_i', \vec{m}_i)$ and return $\Lambda \leftarrow (\mathsf{Eval}(f, (\vec{m}_i)_{i\in[n]}), \mu, \tau)$, where

$$\mu \leftarrow \textstyle\prod_{i\in[n]} e(\sigma_i'^{\omega_i}, \mathsf{ak}_i).$$

$\underline{\mathsf{AVerify}(\mathsf{mk}, \Lambda, \mathsf{ID}, f)}$: Parse mk as (pp, α), Λ as $(\vec{m}, \mu, \tau)$, ID as $(g^{\beta_i})_{i\in[n]}$ and f as $(\omega_i)_{i\in[n]}$ and return $(\vec{m}, \tau)$ if the following holds, and $(\perp, \perp)$ otherwise:

$$\mu' = \left(\textstyle\prod_{i=1}^{n} e(g^{\omega_i}, H(\tau||g^{\beta_i})) \cdot e(\prod_{i=1}^{\ell} g_i^{m_i}, g)\right)^\alpha$$

Scheme 2: HPRA scheme for $\mathcal{F}_{\mathsf{lin}}$ parametrized by Eval.

Theorem 1 (proven in Appendix B.3). *If* HOM-MAC *in Scheme 1 is unforgeable and the* eBCDH *assumption holds, then Scheme 2 represents a signer unforgeable, aggregator unforgeable and input private* HPRA *for class* $\mathcal{F}_{\mathsf{lin}}$ *in the ROM.*

5 Adding Output Privacy

An additional goal is that the aggregator neither learns the input nor the output (output privacy). On our way to achieve this, we formally define the notion of homomorphic proxy-re encryption (HPRE) and develop an instantiation for $\mathcal{F}_{\mathsf{lin}}$. Based on this, we extend Scheme 2 to additionally provide output privacy.

5.1 Homomorphic Proxy Re-Encryption

A homomorphic proxy re-encryption scheme (HPRE) is a PRE which additionally allows the homomorphic evaluation of functions on the ciphertexts. This functionality firstly allows to aggregate messages encrypted under the same public key, and, secondly, to transform the ciphertext holding the evaluation of a function to a ciphertext for another entity, when given the respective proxy re-encryption key. We stress that if the initial ciphertexts are with respect to different public keys, then one can use the respective re-encryption keys to transform them to a common public key before evaluating the function. More formally:

Definition 8 (HPRE). *A* HPRE *for the class $\mathcal{F}$ of functions is a* PRE *with an additional evaluation algorithm $\mathcal{EV}$.*

$\mathcal{EV}(f, \vec{c})$: *This algorithm takes a function $f \in \mathcal{F}$, and a vector of ciphertexts $\vec{c} = (c_i)_{i\in[n]}$ to messages $(m_i)_{i\in[n]}$ all under public key* pk*, and outputs a ciphertext c to message $f((m_i)_{i\in[n]})$ under* pk*.*

Additionally, we require the following compactness notion (analogous to [CF15]).

Definition 9 (Compactness). *A* HPRE *for class $\mathcal{F}$ is called compact if for all $f \in \mathcal{F}$ the running time of the algorithms $\vec{\mathcal{D}}$ is bounded by a fixed polynomial in the security parameter κ.*

Besides the straightforward adoption of correctness, IND-CPA$^-$ remains identical ($\mathcal{EV}$ is a public algorithm). However, we require an IND-CPA$^-$ variant, where the adversary may adaptively choose the targeted user. To the best of our knowledge, such a notion does not exist for PRE. We introduce such a notion (termed mt-IND-CPA$^-$) and show that it is implied by the conventional IND-CPA notions.

Definition 10 (mt-IND-CPA$^-$). *A* (H)PRE *is* mt-IND-CPA$^-$ *secure, if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\varepsilon(\cdot)$ such that*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathcal{P}(1^\kappa), b \xleftarrow{R} \{0,1\},\\ (\mathsf{sk}_h, \mathsf{pk}_h) \leftarrow \mathcal{G}(\mathsf{pp}), \mathcal{O} \leftarrow \{\mathsf{G}(\cdot), \mathsf{RG}(\cdot)\},\\ (m_0, m_1, i^\star, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{pk}_h),\\ c \leftarrow \mathcal{E}^2(m_b, \mathsf{pk}_{i^\star}), b^\star \leftarrow \mathcal{A}(\mathsf{st}, c)\end{array} : b = b^\star\right] \leq 1/2 + \varepsilon(\kappa),$$

where the environment holds an initially empty list HU. G *and* RG *are defined as:*

$\mathsf{G}(i)$: *If* $\mathtt{HU}[i] \neq \perp$ *return* $\perp$. *Otherwise, run* $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathcal{G}(\mathsf{pp})$, *set* $\mathtt{HU}[i] \leftarrow (\mathsf{sk}_i, \mathsf{pk}_i)$, *and return* pk_i.

$\mathsf{RG}(i)$: *If* $\mathtt{HU}[i] = \perp$ *return* $\perp$. *Otherwise, set* $\mathsf{rk}_{i\to h} \leftarrow \mathcal{RG}(\mathtt{HU}[i][1], \mathsf{pk}_h)$ *and return* $\mathsf{rk}_{i\to j}$.

Lemma 2 (proven in Appendix B.2). *Every* IND-CPA$^-$ *(and thus every* IND-CPA*) secure* PRE *also satisfies* mt-IND-CPA$^-$ *security.*

HPRE Construction for Linear Functions (HPRE$_{\mathsf{lin}}$). We state our construction in Scheme 3. Essentially, we build upon the PRE scheme in [AFGH06, third attempt] and turn it into a HPRE$_{\mathsf{lin}}$. For the desired homomorphism we use a standard trick in the context of ElGamal-like encryption schemes: we encode messages $m \in \mathbb{Z}_q$ into the exponent and encrypt $\mathbf{g}^m$. Decryption then yields $m' = \mathbf{g}^m$ and one additionally needs to compute $m = \log_{\mathbf{g}} m'$ to obtain m. Thus, for the schemes to remain efficient, the size of the message space needs to be polynomial in the security parameter. While this might sound quite restrictive, we stress that in practical settings one deals with numerical values where messages in the order of millions to billions are by far sufficient. Thus, this type of decryption is not a limitation and entirely practical.

$\underline{\mathcal{P}(1^\kappa)}$: Run $\mathsf{BG} \leftarrow \mathsf{BGGen}(1^\kappa)$, and return $\mathsf{pp} \leftarrow \mathsf{BG}$.

$\underline{\mathcal{G}(\mathsf{pp})}$: Choose $(a_1, a_2) \xleftarrow{R} \mathbb{Z}_q^2$, and return $(\mathsf{rsk}_A, \mathsf{rpk}_A) \leftarrow ((a_1, a_2), (\mathbf{g}^{a_1}, g^{a_2}))$.

$\underline{\mathcal{RG}(\mathsf{rsk}_A, \mathsf{rpk}_B)}$: Parse rsk_A as $(a_{1A}, \cdot)$ and rpk_B as $(\cdot, g^{a_{2B}})$ and return $\mathsf{rk}_{A\to B} \leftarrow (g^{a_{2B}})^{a_{1A}}$.

$\underline{\mathcal{E}^1(\mathsf{rpk}, m)}$: Parse rpk as $(\mathbf{g}^{a_1}, \cdot)$, choose $k \xleftarrow{R} \mathbb{Z}_q$, and return $c \leftarrow (\mathbf{g}^k, \mathbf{g}^m \cdot (\mathbf{g}^{a_1})^k, 1)$

$\underline{\mathcal{E}^2(\mathsf{rpk}, m)}$: Parse rpk as $(\mathbf{g}^{a_1}, \cdot)$, choose $k \xleftarrow{R} \mathbb{Z}_q$, and return $c \leftarrow (g^k, \mathbf{g}^m \cdot (\mathbf{g}^{a_1})^k, 2)$

$\underline{\mathcal{RE}(\mathsf{rk}_{A\to B}, c_A)}$: Parse c_A as $(c_1, c_2, 2)$ and return $c \leftarrow (e(c_1, \mathsf{rk}_{A\to B}), c_2, R)$

$\underline{\mathcal{D}^1(\mathsf{rsk}, c)}$: Parse c as (c_1, c_2, c_3) and rsk as (a_1, a_2), and return $\mathbf{g}^m \leftarrow c_2 \cdot c_1^{-a_1}$ if $c_3 = 1$ and $\mathbf{g}^m \leftarrow c_2 \cdot c_1^{-1/a_2}$ if $c_3 = R$.

$\underline{\mathcal{D}^2(\mathsf{rsk}, c)}$: Parse c as $(c_1, c_2, 2)$ and rsk as (a_1, a_2), and return $\mathbf{g}^m \leftarrow c_2 \cdot e(g, c_1^{-a_1})$.

$\underline{\mathcal{EV}(f, \vec{c})}$: Parse f as $(\omega_1, \ldots, \omega_n)$ and $\vec{c}$ as $(c_i)_{i\in[n]}$, and return $c \leftarrow \prod_{i\in[n]} c_i^{\omega_i}$, where multiplication and exponentiation is component-wise.

Scheme 3: HPRE$_{\mathsf{lin}}$ based on [AFGH06, third attempt].

As $\mathcal{EV}$ is a public algorithm it does not influence IND-CPA security. Thus, our argumentation is identical to [AFGH06] and we can use the following theorem.

Theorem 2 (cf. [AFGH06]). *If the* eDBDH *assumption holds in* $(\mathbb{G}, \mathbb{G}_T)$ *then Scheme 3 is an* IND-CPA *secure* HPRE$_{\mathsf{lin}}$.

We note that compactness of Scheme 3 (Definition 9) is easy to verify.

$\mathsf{HPRE}_{\mathsf{lin}}$ for Vectors. We extend $\mathsf{HPRE}_{\mathsf{lin}}$ to vectors over $\mathbb{Z}_q$, while preserving the support for re-encryption and the homomorphic properties. It turns out that we can employ a communication efficient solution. That is, borrowing the idea of randomness re-use from [BBKS07] and applying it to $\mathsf{HPRE}_{\mathsf{lin}}$, we can reduce the size of the ciphertexts as long as no re-encryption is performed. Upon setup, we have to fix a maximal length ℓ of the message vectors. The secret and the public keys are then of the form $\mathsf{rsk} \leftarrow (\mathsf{rsk}_i)_{i\in[\ell]} = ((a_{1i}, a_{2i}))_{i\in[\ell]}$, $\mathsf{rpk} \leftarrow (\mathsf{rpk}_i)_{i\in[\ell]} = ((\mathbf{g}^{a_{1i}}, g^{a_{2i}}))_{i\in[\ell]}$, where $(a_{1i}, a_{2i})_{i\in[\ell]} \xleftarrow{R} (\mathbb{Z}_q^2)^\ell$. First and second level encryption are defined as

$$\mathcal{E}^1_\ell(\mathsf{rpk}, \vec{m}) := (\mathbf{g}^k, (\mathbf{g}^{m_i} \cdot \mathsf{rpk}_i[1]^k)_{i\in[\ell]}, 1), \text{ and}$$
$$\mathcal{E}^2_\ell(\mathsf{rpk}, \vec{m}) := (g^k, (\mathbf{g}^{m_i} \cdot \mathsf{rpk}_i[1]^k)_{i\in[\ell]}, 2), \text{ respectively.}$$

Decryption $\mathcal{D}^j_\ell(\cdot,\cdot)$ of a ciphertext $(c[1], (c[i+1])_{i\in[\ell]}, j)$ is defined as $\mathcal{D}^1_\ell(\mathsf{rsk}, \vec{c}) := (c[i+1] \cdot c[1]^{-\mathsf{rsk}_i[1]})_{i\in[\ell]}$, and $\mathcal{D}^2_\ell(\mathsf{rsk}, \vec{c}) := (c[i+1] \cdot e(c[1], g^{-\mathsf{rsk}_i[1]}))_{i\in[\ell]}$. Re-encryption key generation is $\mathcal{RG}_\ell(\mathsf{rsk}_A, \mathsf{rpk}_B) := (((\mathsf{rpk}_B)_i[2])^{(\mathsf{rsk}_A)_i[1]})_{i\in[\ell]}$. From a second level ciphertext $\vec{c}_A$ for A and a re-encryption key $\mathsf{rk}_{A\to B}$, one can compute a ciphertext $\vec{c}_B$ for B as $\vec{c}_B \leftarrow \mathcal{RE}(\mathsf{rk}_{A\to B}, \vec{c}_A) := ((e(c_A[1], \mathsf{rk}_{A\to B}[i]), c_A[i+1]))_{i\in[\ell]}$. Note that re-encrypted ciphertexts have a different form. Thus we do not need to add the level as suffix. Decryption $\mathcal{D}^1_\ell(\cdot,\cdot)$ for re-encrypted ciphertexts is $\mathcal{D}^1_\ell(\mathsf{rsk}, (c_i)_{i\in[\ell]}) := (c_i[2] \cdot c_i[1]^{-1/\mathsf{rsk}_i[2]})_{i\in[\ell]}$.

Theorem 3. *If the* eDBDH *assumption holds, then the extension of* $\mathsf{HPRE}_{\mathsf{lin}}$ *as described above, yields an* IND-CPA *secure* $\mathsf{HPRE}_{\mathsf{lin}}$ *for vectors.*

Proof (sketch). IND-CPA security of the original scheme implies Theorem 3 under a polynomial loss: using ℓ hybrids, where in hybrid i $(1 \le i \le \ell)$ the i-th ciphertext component is exchanged by random under the original strategy in [AFGH06].

Combining the theorem above with Lemma 2 yields:

Corollary 1. *The extension of* $\mathsf{HPRE}_{\mathsf{lin}}$ *as described above yields an* mt-IND-CPA$^-$ *secure* $\mathsf{HPRE}_{\mathsf{lin}}$ *for vectors.*

5.2 Putting the Pieces Together: Output Privacy

Our idea is to combine Scheme 2 with the $\mathsf{HPRE}_{\mathsf{lin}}$ presented above. In doing so, we face some obstacles. First, a naïve combination of those primitives does not suit our needs: one can still verify guesses for signed messages using solely the signatures, since signatures are publicly verifiable. Second, switching to a MAC for the data sources is also no option, as this would require an interactive re-key generation. This is excluded by our model as we explicitly want to avoid it. Thus, we pursue a different direction and turn the signatures used in Scheme 2 into a MAC-like primitive by blinding a signature with a random element g^r. An

aggregated MAC holding an evaluation of f is then blinded by $g^{f(\ldots,r,\ldots)}$, i.e., the receiver needs to evaluate the function f on the all blinding values from the single sources. Now the question arises as how to transmit the blinding values to the receiver. Using our $\mathsf{HPRE}_{\mathsf{lin}}$ for vectors yields an arguably elegant solution: by treating the randomness as an additional vector component, we can use the re-encryption features of the $\mathsf{HPRE}_{\mathsf{lin}}$. More importantly, by executing the $\mathcal{EV}$ algorithm the aggregator simultaneously evaluates the function f on the data and on the randomness so that the receiver can directly obtain the blinding value $f(\ldots, r, \ldots)$ upon decryption.

$\underline{\mathsf{Gen}(1^\kappa, \ell)}$: Fix a homomorphic $\mathsf{PRE} = (\mathcal{P}, \mathcal{G}, \vec{\mathcal{E}}, \vec{\mathcal{D}}, \mathcal{RG}, \mathcal{RE}, \mathcal{EV})$ for class $\mathcal{F}_{\mathsf{lin}}$ and the $\mathsf{HPRA}(\mathcal{EV}) = (\mathbf{Gen}, \mathbf{SGen}, \mathbf{VGen}, \mathbf{Sign}, \mathbf{Verify}, \mathbf{SRGen}, \mathbf{VRGen}, \mathbf{Agg}, \mathbf{AVerify})$ from Scheme 2 such that $\mathcal{M}_{\mathsf{PRA}} \subseteq \mathcal{M}_{\mathsf{PRE}}$, run $\mathsf{pp}_s \leftarrow \mathbf{Gen}(1^\kappa, \ell)$, $\mathsf{pp}_e \leftarrow \mathcal{P}(1^\kappa, \ell + 1)$, and return $\mathsf{pp} \leftarrow (\mathsf{pp}_s, \mathsf{pp}_e)$.

$\underline{\mathsf{SGen}(\mathsf{pp})}$: Run $(\mathbf{id}, \mathbf{sk}, \mathbf{pk}) \leftarrow \mathbf{SGen}(\mathsf{pp}_s)$, $(\mathsf{rsk}, \mathsf{rpk}) \leftarrow \mathcal{G}(\mathsf{pp}_e)$, and return $(\mathsf{id}, \mathsf{sk}, \mathsf{pk}) \leftarrow (\mathbf{id}, (\mathbf{sk}, \mathsf{rsk}, \mathsf{rpk}), \mathbf{pk})$.

$\underline{\mathsf{VGen}(\mathsf{pp})}$: Run $(\mathbf{mk}, \mathbf{aux}) \leftarrow \mathbf{VGen}(\mathsf{pp}_s)$, $(\mathsf{rsk}, \mathsf{rpk}) \leftarrow \mathcal{G}(\mathsf{pp}_e)$, and return $(\mathsf{mk}, \mathsf{aux}) \leftarrow ((\mathbf{mk}, \mathsf{rsk}), (\mathbf{aux}, \mathsf{rpk}))$.

$\underline{\mathsf{Sign}(\mathsf{sk}, \vec{m}, \tau)}$: Parse sk as $(\mathbf{sk}, \cdot, \mathsf{rpk})$, choose $r \xleftarrow{R} \mathbb{Z}_q$, and return $\sigma \leftarrow (\sigma' \cdot g^r, \vec{c})$, where

$$(\sigma', \cdot) \leftarrow \mathbf{Sign}(\mathbf{sk}, \vec{m}, \tau) \text{ and } \vec{c} \leftarrow \mathcal{E}^2_{\ell+1}(\mathsf{rpk}, \vec{m}||r).$$

$\underline{\mathsf{SRGen}(\mathsf{sk}_i, \mathsf{aux})}$: Parse sk_i as $(\mathbf{sk}_i, \mathsf{rsk}_i, \mathsf{rpk}_i)$ and aux as $(\mathbf{aux}, \mathsf{rpk})$. Obtain $\mathbf{rk}_i \leftarrow \mathbf{SRGen}(\mathbf{sk}_i, \mathbf{aux})$ and $\mathsf{prk}_i \leftarrow \mathcal{RG}(\mathsf{rsk}_i, \mathsf{rpk})$, and return $\mathsf{rk}_i \leftarrow (\mathbf{rk}_i, \mathsf{prk}_i)$.

$\underline{\mathsf{VRGen}(\mathsf{pk}_i, \mathsf{mk}, \mathsf{rk}_i)}$: Parse pk_i as $\mathbf{pk}_i$ and mk as $(\mathbf{mk}, \cdot)$, obtain $\mathbf{ak}_i \leftarrow \mathbf{VRGen}(\mathbf{pk}_i, \mathbf{mk})$ and return $\mathsf{ak}_i \leftarrow (\mathbf{ak}_i, \mathsf{rk}_i)$.

$\underline{\mathsf{Agg}((\mathsf{ak}_i)_{i\in[n]}, (\sigma_i)_{i\in[n]}, \tau, f)}$: For $i \in [n]$ parse ak_i as $(\mathbf{ak}_i, (\mathbf{rk}_i, \mathsf{prk}_i))$, σ_i as $(\sigma'_i, \vec{c}_i)$. Output $\Lambda \leftarrow (\vec{c}', \mu, \tau)$, where

$$(\vec{c}'_i \leftarrow \mathcal{RE}(\mathsf{prk}_i, \vec{c}_i))_{i\in[n]},\ (\vec{c}', \mu, \tau) \leftarrow \mathbf{Agg}((\mathbf{ak}_i)_{i\in[n]}, (\sigma'_i, \vec{c}'_i)_{i\in[n]}, f).$$

$\underline{\mathsf{AVerify}(\mathsf{mk}, \Lambda, \mathsf{ID}, f)}$: Parse mk as $(\mathbf{mk}, \mathsf{rsk})$ and Λ as $(\vec{c}, \mu, \tau)$, obtain $\vec{m}'||r \leftarrow \mathcal{D}^1_{\ell+1}(\mathsf{rsk}, \vec{c})$ and return $(\vec{m}, \tau)$ if the following holds, and $(\perp, \perp)$ otherwise:

$$\mathbf{AVerify}(\mathbf{mk}, (\vec{m}, \mu \cdot (\mathbf{g}^r)^{-1}, \tau), \mathsf{ID}, f) = 1$$

Scheme 4: Output private HPRA scheme for $\mathcal{F}_{\mathsf{lin}}$ with $\mathsf{Eval} \coloneqq \mathcal{EV}$

Note on the Instantiation. Augmenting Scheme 2 to obtain Scheme 4 using $\mathsf{HPRE}_{\mathsf{lin}}$ requires an alternative decryption strategy for the vector component containing r, as r is uniformly random in $\mathbb{Z}_q$ and can thus not be efficiently recov-

ered. Fortunately, obtaining $r \in \mathbb{Z}_q$ is not required, as g^r (resp. $\mathbf{g}^r$) is sufficient to unblind the signature (resp. MAC). Those values are efficiently recoverable.

Theorem 4 (proven in Appendix B.4). *If Scheme 2 is signer and aggregator unforgeable, and* $\mathsf{HPRE}_{\mathsf{lin}}$ *for vectors is* $\mathsf{mt\text{-}IND\text{-}CPA}^{-}$ *secure, then Scheme 4 is a signer and aggregator unforgeable, input and output private* HPRA *for class* $\mathcal{F}_{\mathsf{lin}}$.

6 Conclusion

In this paper we introduce the notion of homomorphic proxy re-authenticators. This concept covers various important issues in the multi-user data aggregation setting not considered by previous works. We present two provably secure and practically efficient instantiations of our novel concept, which differ regarding the strength of the privacy guarantees. Our schemes are modular in the sense that they are constructed from building blocks which may as well be useful in other settings. One important building block is the concept of homomorphic proxy re-encryption, which we also introduce and construct in this paper.

Acknowledgements. We thank David Nuñez for his valuable comments on a draft of this paper.

References

[AB09] Shweta Agrawal and Dan Boneh. Homomorphic macs: Mac-based integrity for network coding. In *ACNS*, 2009.

[AFGH06] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9(1), 2006.

[AGH15] Joseph A. Akinyele, Christina Garman, and Susan Hohenberger. Automating fast and secure translations from type-i to type-iii pairing schemes. In *CCS*, 2015.

[AH05] Giuseppe Ateniese and Susan Hohenberger. Proxy re-signatures: new definitions, algorithms, and applications. In *CCS*, 2005.

[AHO16] Masayuki Abe, Fumitaka Hoshino, and Miyako Ohkubo. Design in type-i, run in type-iii: Fast and scalable bilinear-type conversion using integer programming. In *CRYPTO*, 2016.

[AP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In *CRYPTO*, 2014.

[ARHR13] Erman Ayday, Jean Louis Raisaro, Jean-Pierre Hubaux, and Jacques Rougemont. Protecting and evaluating genomic privacy in medical tests and personalized medicine. In *WPES*, 2013.

[BBKS07] Mihir Bellare, Alexandra Boldyreva, K. Kurosawa, and Jessica Staddon. Multirecipient encryption schemes: How to save on bandwidth and computation without sacrificing security. *IEEE Trans. Information Theory*, 53(11), 2007.

[BBL16] Olivier Blazy, Xavier Bultel, and Pascal Lafourcade. Two secure anonymous proxy-based data storages. In *SECRYPT*, pages 251–258, 2016.

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT*, 1998.

[BFKW09] Dan Boneh, David Mandell Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *PKC*, 2009.

[BFR13] Michael Backes, Dario Fiore, and Raphael M. Reischuk. Verifiable delegation of computation on outsourced data. In *CCS*, 2013.

[BGP+16] Cristian Borceaa, Arnab "Bobby" Deb Guptaa, Yuriy Polyakova, Kurt Rohloffa, and Gerard Ryana. Picador: End-to-end encrypted publish-subscribe information distribution with proxy re-encryption. *Future Generation Comp. Syst.*, 62:119–127, 2016.

[BJL16] Fabrice Benhamouda, Marc Joye, and Benoît Libert. A new framework for privacy-preserving aggregation of time-series data. *ACM Trans. Inf. Syst. Secur.*, 18(3), 2016.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *CRYPTO*, pages 410–428, 2013.

[Boy08] Xavier Boyen. The uber-assumption family. In *Pairing*, 2008.

[Cat14] Dario Catalano. Homomorphic signatures and message authentication codes. In *SCN*, 2014.

[CCMT09] Claude Castelluccia, Aldar C.-F. Chan, Einar Mykletun, and Gene Tsudik. Efficient and provably secure aggregation of encrypted data in wireless sensor networks. *ACM Trans. Sen. Netw.*, 5(3), 2009.

[CD16] Sébastien Canard and Julien Devigne. Highly privacy-protecting data sharing in a tree structure. *Future Generation Comp. Syst.*, 62:119–127, 2016.

[CF15] Dario Catalano and Dario Fiore. Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In *CCS*, 2015.

[CFW14] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *CRYPTO*, 2014.

[CH07] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *CCS*, pages 185–194, 2007.

[CMP14] Dario Catalano, Antonio Marcedone, and Orazio Puglisi. Authenticating computation on groups: New homomorphic primitives and applications. In *ASIACRYPT*, 2014.

[CSS12] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Privacy-preserving stream aggregation with fault tolerance. In *Financial Crypto*, 2012.

[DL11] George Danezis and Benjamin Livshits. Towards ensuring client-side computational integrity. In *CCSW*, 2011.

[DS16] David Derler and Daniel Slamanig. Key-homomorphic signatures and applications to multiparty signatures. *Cryptology ePrint Archive*, 2016:792, 2016.

[FMNP16] Dario Fiore, Aikaterini Mitrokotsa, Luca Nizzardo, and Elena Pagnin. Multi-key homomorphic authenticators. In *ASIACRYPT*, pages 499–530, 2016.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[GMP14] Felix Günther, Mark Manulis, and Andreas Peter. Privacy-enhanced participatory sensing with collusion resistance and data aggregation. In *CANS*, 2014.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, pages 75–92, 2013.

[ID03] Anca-Andreea Ivan and Yevgeniy Dodis. Proxy cryptography revisited. In *NDSS*, 2003.

[JL13] Marc Joye and Benoît Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In *Financial Crypto*, 2013.

[LC13] Qinghua Li and Guohong Cao. Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error. In *PETS*, 2013.

[LCP14] Qinghua Li, Guohong Cao, and Thomas F. La Porta. Efficient and privacy-aware data aggregation in mobile sensing. *IEEE Trans. Dep. Sec. Comput.*, 11(2), 2014.

[LDPW14] Junzuo Lai, Robert H. Deng, HweeHwa Pang, and Jian Weng. Verifiable computation on outsourced encrypted data. In *ESORICS*, 2014.

[LEM14] Iraklis Leontiadis, Kaoutar Elkhiyaoui, and Refik Molva. Private and dynamic time-series data aggregation with trust relaxation. In *CANS*, 2014.

[LEÖM15] Iraklis Leontiadis, Kaoutar Elkhiyaoui, Melek Önen, and Refik Molva. PUDA - privacy and unforgeability for data aggregation. In *CANS*, 2015.

[LTWC16] Russell W. F. Lai, Raymond K. H. Tai, Harry W. H. Wong, and Sherman S. M. Chow. A zoo of homomorphic signatures: Multi-key and key-homomorphism. Cryptology ePrint Archive, Report 2016/834, 2016.

[LV08] Benoît Libert and Damien Vergnaud. Multi-use unidirectional proxy re-signatures. In *CCS*, 2008.

[LV11] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. *IEEE Trans. Information Theory*, 57(3), 2011.

[MLO16] Chunguang Ma, Juyan Li, and Weiping Ouyang. A homomorphic proxy re-encryption from lattices. In *ProvSec 2016*, Nov. 2016.

[NA14] David Nuñez and Isaac Agudo. Blindidm: A privacy-preserving approach for identity management as a service. *Int. J. Inf. Sec.*, 13(2):199–215, 2014.

[NAL12] David Nuñez, Isaac Agudo, and Javier Lopez. Integrating openid with proxy re-encryption to enhance privacy in cloud-based identity services. In *CloudCom*, pages 241–248, 2012.

[NAL15] David Nuñez, Isaac Agudo, and Javier Lopez. A parametric family of attack models for proxy re-encryption. In *CSF*, pages 290–301, 2015.

[NAL16] David Nuñez, Isaac Agudo, and Javier Lopez. On the application of generic cca-secure transformations to proxy re-encryption. *Security and Communication Networks*, 9(12):1769–1785, 2016.

[RN10] Vibhor Rastogi and Suman Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD*, 2010.

[SCR+11] Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *NDSS*, 2011.

[SSZ14] Daniel Slamanig, Klaus Stranacher, and Bernd Zwattendorfer. User-centric identity as a service-architecture for eids with selective attribute disclosure. In *SACMAT*, pages 153–164, 2014.

[WB15] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, 58(2), 2015.

[XXW+16] Peng Xu, Jun Xu, Wei Wang, Hai Jin, Willy Susilo, and Deqing Zou. Generally hybrid proxy re-encryption: A secure data sharing among cryptographic clouds. In *AsiaCCS*, pages 913–918, 2016.

[ZSSH14] Bernd Zwattendorfer, Daniel Slamanig, Klaus Stranacher, and Felix Hörandner. A federated cloud identity broker-model for enhanced privacy via proxy re-encryption. In *CMS*, pages 92–103, 2014.

A Additional Background Material

A.1 Cryptographic Assumptions

First, we formally definite a bilinear group generation.

Definition 11. *A bilinear-group generation algorithm* BGGen *is a PPT algorithm that takes a security parameter* κ *and outputs a bilinear group description* $\mathsf{BG} = (q, \mathbb{G}, \mathbb{G}_T, e, g, \mathbf{g})$ *with* $\mathbb{G} = \langle g \rangle$ *and* $\mathbb{G}_T = \langle \mathbf{g} \rangle$*, both of order* q *being a prime of bitlength* κ *and a pairing* $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$*.*

Subsequently, we recall the bilinear decisional Diffie-Hellman assumption, a standard assumption in the bilinear setting.

Definition 12 (BDDH)**.** *The bilinear decisional Diffie-Hellman assumption holds relative to* $\mathsf{BG} \leftarrow \mathsf{BGGen}(1^\kappa)$*, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\varepsilon(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l} r, s, t, u \xleftarrow{R} \mathbb{Z}_q, b \xleftarrow{R} \{0,1\}, \\ b^\star \leftarrow \mathcal{A}\left(\mathsf{BG}, g^r, g^s, g^t, \mathbf{g}^{b \cdot rst + (1-b)u}\right) \end{array} : b = b^\star \right] \leq 1/2 + \varepsilon(\kappa).$$

In addition, we introduce a plausible assumption which follows from the Uber-assumption [Boy08] with $R = S = \langle 1, 1/U, U, V, W \rangle, T = \langle 1 \rangle, f = \langle UVW \rangle$.

Definition 13 (eBCDH)**.** *The extended bilinear computational Diffie-Hellman assumption holds relative to* $\mathsf{BG} \leftarrow \mathsf{BGGen}(1^\kappa)$*, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\varepsilon(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l} u, v, w \xleftarrow{R} \mathbb{Z}_q \\ \mathbf{h} \leftarrow \mathcal{A}(\mathsf{BG}, g^{1/u}, g^u, g^v, g^w) \end{array} : \mathbf{h} = e(g,g)^{uvw} \right] \leq \varepsilon(\kappa).$$

A.2 Security Definitions for Linearly Homomorphic MACS

While we omit the obvious correctness definition, we recall the unforgeability definition. The environment maintains a list $\mathtt{SIG}$ which is initially empty and the sign oracle Sig is defined as:

$\mathsf{Sig}((\vec{v}_i)_{i\in[n]}, (\mathsf{id}_i)_{i\in[n]}, \tau)$: If $\mathtt{SIG}[\tau] \neq \bot$ or there exists $u, v \in [n], u \neq v$ so that $\mathsf{id}_u = \mathsf{id}_v$ return $\bot$. Otherwise, compute $\mu_i \leftarrow \mathcal{S}(\mathsf{sk}, \vec{v}_i, \mathsf{id}_i, \tau)$ for $i \in [n]$, set $\mathtt{SIG}[\tau] \leftarrow \{(\vec{v}_i, \mathsf{id}_i)\}_{i\in[n]}$ and return $(\mu_i)_{i\in[n]}$.

Definition 14. *A* $\mathsf{HOM\text{-}MAC}$ *is unforgeable if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\varepsilon(\cdot)$ *such that:*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathcal{P}(1^\kappa, \ell), \mathsf{sk} \leftarrow \mathcal{G}(\mathsf{pp}), \\ (\vec{y}^\star, \mu^\star, \tau^\star, (\mathsf{id}_i^\star)_{i\in[n]}, \\ f^\star) \leftarrow \mathcal{A}^{\mathsf{Sig}(\cdot,\cdot,\cdot)}(\mathsf{pp}) \end{array} : \begin{array}{r} \mathcal{V}(\mathsf{sk}, \vec{y}^\star, \mu^\star, \tau^\star, (\mathsf{id}_i^\star)_{i\in[n]}, f^\star) = 1 \wedge \\ f^\star \in \mathcal{F}_{\mathsf{lin}} \wedge (\nexists (\vec{v}_j)_{j\in[n]} : \\ (\forall j \in [n] : (\vec{v}_j, \mathsf{id}_j^\star) \in \mathsf{S}[\tau^\star]) \wedge \\ f^*((\vec{v}_j)_{j\in[n]}) = \vec{y}^\star) \end{array} \right] \leq \varepsilon(\kappa).$$

B Security Proofs

B.1 Proof of Lemma 1

We weaken BDDH to BDDH′, where the adversary is given $(\mathsf{BG}, g^a, \mathbf{g}^b, \mathbf{g}^c) \in \mathbb{G}_1 \times {\mathbb{G}_T}^2$ with $a, b \xleftarrow{R} \mathbb{Z}_q$ and needs to decide whether $c = ab$, or c is random in $\mathbb{Z}_q$. Clearly, BDDH′ is weaker than BDDH: given a BDDH instance $(\mathsf{BG}, g^a, g^b, g^c, \mathbf{g}^d)$ one can use the BDDH′ distinguisher on $(\mathsf{BG}, g^a, e(g^b, g^c), \mathbf{g}^d)$.

Proof. We prove Lemma 1 under BDDH′ (and, therefore, BDDH) and let $q_{\mathsf{R}} \leq \mathsf{poly}(\kappa)$ be the number of random oracle queries (also including the calls we use internally in the reduction). We obtain a BDDH′ instance $(\mathsf{BG}, g^a, \mathbf{g}^b, \mathbf{g}^c)$, choose $((x_i, y_i) \xleftarrow{R} {\mathbb{Z}_q}^2)_{i \in [q_{\mathsf{R}}]}$, set $((g^{a_i}, \mathbf{g}^b, \mathbf{g}^{c_i}) \leftarrow ((g^a)^{x_i} g^{y_i}), \mathbf{g}^b, (\mathbf{g}^c)^{x_i} (\mathbf{g}^b)^{y_j})_{i \in [q_{\mathsf{R}}]}$, choose $(u_i)_{i \in [\ell]} \xleftarrow{R} \mathbb{Z}_q^\ell$, and set $(g_i \leftarrow g^{u_i})_{i \in [\ell]}$. Finally, set $\mathsf{pp} \leftarrow (\mathsf{BG}, H, (g_i)_{i \in [\ell]})$, start $\mathcal{A}$ on pp and simulate the oracles as follows. We use a counter j initialized to 1 and an initially empty list $\mathtt{H}$.

$H(x)$: If $x \notin \mathtt{H}$ set $\mathtt{H}[x] \leftarrow (g^{a_j}, \mathbf{g}^{c_j})$ and $j \leftarrow j + 1$. Return $\mathtt{H}[x][1]$.
$\mathsf{Sig}((\vec{v}_i)_{i \in [n]}, (\mathsf{id}_i)_{i \in [n]}, \tau)$: As the original oracle, except that the values $(\mu_i)_{i \in [n]}$ are computed as follows. For $i \in [n]$ call $H(\tau || \mathsf{id}_i)$, set $\mu_i \leftarrow \mathtt{H}[\tau || \mathsf{id}_i][2] \cdot \prod_{j \in [\ell]} (\mathbf{g}^b)^{v_{i_j} \cdot u_j}$ and return $(\mu_i)_{i \in [n]}$.

Now, if the BDDH′ instance is valid the simulation is perfect. If the BDDH′ instance is invalid the responses of the Sig oracle are uniformly random and independent values from $\mathbb{G}_T$ and therefore do not reveal anything about the MAC key b. In this case the adversary can only guess a forgery with probability $1/q$. Both cases are computationally indistinguishable. □

B.2 Proof of Lemma 2

We directly start with IND-CPA$^-$ as IND-CPA straight-forwardly implies IND-CPA$^-$.

Proof. We prove the lemma by bounding the success probability in the mt-IND-CPA$^-$ game relative to the IND-CPA$^-$ bound. Therefore, we let q_{G} be the number of queries to the G oracle.

Game 0: The mt-IND-CPA$^-$ game.
Game 1: As Game 0, but we guess the index $i^\star$ beforehand. If our guess is wrong we abort.
Transition$^{0 \to 1}$: We have that $\Pr[S_0] = q_{\mathsf{G}} \cdot \Pr[S_1]$.
Game 2: As Game 1, but we engage with an IND-CPA′ challenger, obtain $(\mathsf{pp}, \mathsf{pk}_t, \mathsf{pk}_h, \mathsf{rk}_{t \to h})$, set $\mathtt{HU}[i^\star] \leftarrow (\bot, \mathsf{pk}_t, \mathsf{rk}_{t \to h})$ and start $\mathcal{A}$ on $(\mathsf{pp}, \mathsf{pk}_h)$. Furthermore, we simulate the oracles when queried for user $i^\star$ as follows:
$\mathsf{G}(i^\star)$: Return $\mathtt{HU}[i^\star][2]$ (i.e., pk_t).
$\mathsf{RG}(i^\star)$: Return $\mathtt{HU}[i^\star][3]$ (i.e., $\mathsf{rk}_{t \to h}$).
The oracle calls for the remaining indexes are simulated honestly.
Transition$^{1 \to 2}$: This change is conceptual.

Whenever an adversary in Game 2 outputs its guess $b^\star$, we can forward $b^\star$ to the $\mathsf{IND\text{-}CPA}^-$ challenger and win whenever the adversary wins Game 2. As we have that $\Pr[S_1] = \Pr[S_2] \leq \epsilon_{\mathsf{cpa}^-}(\kappa)$, and, thus, $\Pr[S_0] \leq q_{\mathsf{G}} \cdot \epsilon_{\mathsf{cpa}^-}(\kappa)$, this concludes the proof. □

B.3 Proof of Theorem 1

We prove Theorem 1 by proving Lemmata 3-5.

Lemma 3. *If Scheme 1 is unforgeable, then Scheme 2 is signer unforgeable.*

Proof. We show that an efficient adversary $\mathcal{A}$ against signer unforgeability can efficiently be turned into an efficient adversary $\mathcal{B}$ against unforgeability of Scheme 1 by presenting a reduction $\mathcal{R}$, which interacts with the unforgeability challenger of Scheme 1 and simulates the environment for $\mathcal{A}$, i.e., so that $\mathcal{B} = (\mathcal{A}, \mathcal{R})$.

$\mathcal{R}$ obtains pp from $\mathcal{C}$, starts $\mathcal{A}$ on pp and $\mathsf{aux} = \emptyset$ and simulates the environment for $\mathcal{A}$ as follows.

$\mathsf{VR}(i)$: If $\mathtt{S}[i] = \bot \ \vee \ \mathtt{RK}[i] = \bot$ return $\bot$, otherwise set $\mathtt{AK}[i] \leftarrow \top$.

$\mathsf{A}((\sigma_{j_i})_{i\in[n]}, (j_i)_{i\in[n]}, \tau, f)$: Check whether any $(\mathsf{Verify}(\mathsf{pk}_{j_i}, \vec{m}_{j_i}, \tau, \sigma_{j_i}) = 0)_{i\in[n]}$, whether there is any duplicate index j in $(j_i)_{i\in[n]}$, whether $\mathtt{SIG}[\tau] \neq \bot$, or whether $\mathtt{AK}[j_i] = \bot$ for any $i \in [n]$, and return $\bot$ if so. Otherwise, set $\mathtt{SIG}[\tau] \leftarrow \bigcup_{i\in[n]}\{(\vec{m}_{j_i}, \mathsf{id}_{j_i})\}$ compute $(\mu_i)_{i\in[n]} \leftarrow \mathcal{C}.\mathsf{Sig}((\vec{m}_{j_i})_{i\in[n]}, (\mathsf{id}_{j_i})_{i\in[n]}, \tau)$ and return $\Lambda \leftarrow (\mathsf{Eval}(f, (m_{j_i})_{i\in[n]}), \mu, \tau)$, where $\mu \leftarrow \prod_{i\in[n]} \mu_i^{\omega_i}$.

The oracles SG, SKey, and SR are simulated honestly. If $\mathcal{A}$ eventually outputs a forgery $(\Lambda^\star, \mathsf{ID}^\star, f^\star)$ the reduction $\mathcal{R}$ can parse $\Lambda^\star$ as $(\vec{m}, \mu, \tau)$, forward $(\vec{m}, \mu, \tau, \mathsf{ID}^\star, f^\star)$ to $\mathcal{C}$ and wins the unforgeability game with the same probability as $\mathcal{A}$ breaks HPRA unforgeability. □

Lemma 4. *If the* eBCDH *assumption holds, then Scheme 2 is aggregator unforgeable in the ROM.*

Our proof is along the lines of [BFKW09], but we require a slightly different assumption (for eBCDH, see Appendix A.1).

Proof (Aggregator Unforgeability). We construct an efficient algorithm $\mathcal{R}$ which turns an efficient algorithm $\mathcal{A}$ breaking aggregator unforgeability into an efficient eBCDH solver. $\mathcal{R}$ internally maintains the lists $\mathtt{Rnd}$ and $\mathtt{H}$, which are initially empty. $\mathcal{R}$ obtains an extended bilinear CDH instance $\mathsf{eBCDH}_\kappa \leftarrow (\mathsf{BG}, g^{1/\beta}, g^\beta, g^{\alpha/\beta}, g^\gamma)$ relative to the security parameter κ and runs the following modified Gen algorithm Gen' to obtain pp.

$\mathsf{Gen}'(\mathsf{eBCDH}_\kappa, \ell)$: For $i \in [\ell]$, choose $\mathtt{Rnd}[i] \leftarrow (s_i, t_i) \xleftarrow{R} \mathbb{Z}_q^2$ and set $g_i \leftarrow (g^\gamma)^{s_i} g^{t_i}$. Fix $H : \mathbb{Z}_q \to \mathbb{G}$ and return $\mathsf{pp} \leftarrow (\mathsf{BG}, H, (g_i)_{i\in[\ell]}, \ell)$

Then, $\mathcal{R}$ starts $\mathcal{A}$ on pp and $\mathsf{aux} = \emptyset$, where the oracles are simulated as follows and we assume that all oracles have implicit access to the state of $\mathcal{R}$ (in particular to $\mathtt{Rnd}$ and $\mathtt{H}$):

$H(x)$: If $x \in \mathtt{H}$ return $\mathtt{H}[x][1]$. Otherwise choose $(\rho, \nu) \xleftarrow{R} \mathbb{Z}_q^2$, set $\mathtt{H}[x] \leftarrow ((g^\gamma)^\rho g^\nu, \rho, \nu)$ and return $\mathtt{H}[x][1]$.

$\mathsf{SG}(i)$: Choose $\xi \xleftarrow{R} \mathbb{Z}_q^*$, and set $\mathtt{S}[i] \leftarrow (\xi, (g^\beta)^\xi, (g^{1/\beta})^{1/\xi})$. Then, return $((g^\beta)^\xi, (g^{1/\beta})^{1/\xi})$.

$\mathsf{Sig}((j_i)_{i\in[n]}, (\vec{m}_{j_i})_{i\in[n]})$: As the original oracle, except: choose $\tau \xleftarrow{R} \mathbb{Z}_q$ and for all $i \in [n]$ choose $\nu_i \xleftarrow{R} \mathbb{Z}_q$. If $\tau || \mathtt{S}[j_i][2] \in \mathtt{H}$ for any $i \in [n]$ abort. Otherwise, for all $i \in [n]$, set $\rho_i \leftarrow -\sum_{k\in[\ell]} \mathtt{Rnd}[k][1] \cdot m_{j_i}[k]$, and $\mathtt{H}[\tau || \mathtt{S}[j_i][2]] \leftarrow ((g^\gamma)^{\rho_i} g^{\nu_i}, \rho_i, \nu_i)$, compute $\delta \leftarrow \nu_i + \sum_{k\in[\ell]} m_{j_i}[k] \cdot \mathtt{Rnd}[k][2]$ and return $(\sigma_{j_i} \leftarrow \mathtt{S}[j_i][2]^\delta)_{i\in[n]}$ and τ.

$\mathsf{VR}(i)$: If $\mathtt{S}[i] = \bot \ \vee \ \mathtt{RK}[i] = \bot \ \vee \ \mathtt{AK}[i] \neq \bot$ return $\bot$. Otherwise, set $\mathtt{AK}[i] \leftarrow (g^{\alpha/\beta})^{1/\mathtt{S}[i][1]}$.

The oracles SR and VRKey are simulated honestly. If $\mathcal{A}$ eventually outputs a valid forgery $(\Lambda^\star, \mathsf{ID}^\star, f^\star) = ((\vec{m}^\star, \mu^\star, \tau^\star), \mathsf{ID}^\star, f^\star)$ with $f^\star = (\omega_i^\star)_{i\in[n]}$, we know that it is of the form

$$\mu^\star = \big(e(\textstyle\prod_{i\in[\ell]} g_i^{m_i^\star}, g) \cdot \prod_{i\in[n]} e(g^{\omega_i^\star}, H(\tau^\star || \mathsf{id}_i^\star))\big)^\alpha = (\mathbf{g}^{\alpha\gamma})^{\sum_{i\in[\ell]} m_i^\star \cdot \mathtt{Rnd}[i][1]} (\mathbf{g}^\alpha)^{\sum_{i\in[\ell]} m_i^\star \cdot \mathtt{Rnd}[i][2]} \cdot (\mathbf{g}^{\alpha\gamma})^{\sum_{i\in[n]} \omega_i^\star \cdot \mathtt{H}[\tau^\star || \mathsf{pk}_i^\star][2]} (\mathbf{g}^\alpha)^{\sum_{i\in[n]} \omega_i^\star \cdot \mathtt{H}[\tau^\star || \mathsf{pk}_i^\star][3]}.$$

Then we let

$$\psi = \textstyle\sum_{i\in[\ell]} m_i^\star \cdot \mathtt{Rnd}[i][1] + \sum_{i\in[n]} \omega_i^\star \cdot \mathtt{H}[\tau^\star || \mathsf{pk}_i^\star][2],$$
$$\upsilon = \textstyle\sum_{i\in[\ell]} m_i^\star \cdot \mathtt{Rnd}[i][2] + \sum_{i\in[n]} \omega_i^\star \cdot \mathtt{H}[\tau^\star || \mathsf{pk}_i^\star][3],$$

and output $\mathbf{g}^{\alpha\gamma} \leftarrow (\mu^\star \cdot e(g^\beta, g^{\alpha/\beta})^{-\upsilon})^{1/\psi}$ as eBCDH solution. The simulation is negligibly close to the original game: all values are identically distributed; the signature is uniquely determined by the responses of the random oracle and the key, and, thus, also identically distributed as a real signature. Collisions in the answers of the random oracle only occur with negligible probability. Consequently, also an abort happens with negligible probability. What remains is to analyze the probability that the forgery output by $\mathcal{A}$ is of the form that we actually extract $\mathbf{g}^{\alpha\gamma}$, i.e., we have that $\psi \neq 0$ which we define as event E. By the same argumentation as [BFKW09] we obtain that $\Pr[\neg E] = 1/q$ which concludes the proof (cf. [BFKW09, Proof of Theorem 6] for a more detailed probability analysis). □

Lemma 5. *Scheme 2 is input private.*

Proof. Both input privacy ensembles are identical. □

B.4 Proof of Theorem 4

We prove Theorem 4 by proving Lemma 6-8.

Lemma 6. *If Scheme 2 is signer and aggregator unforgeable, then Scheme 4 is so as well.*

Proof. Additional encryption does not influence unforgeability. □

Lemma 7. *Scheme 4 is input private.*

Proof. Both input privacy ensembles are identical. □

Lemma 8. *If* $\mathsf{HPRE}_{\mathsf{lin}}$ *for vectors is* $\mathsf{mt\text{-}IND\text{-}CPA}^{-}$ *secure, then Scheme 4 is output private.*

Proof. We prove output privacy using a sequence of games, and let q_{RoS} be the cumulative number of signing calls within the RoS queries.
Game 0: The original privacy game with bit $b = 0$.
Game $1_i (1 \leq i < q_{\mathsf{RoS}})$**:** As Game 0, but we modify i-th Sign run within RoS as follows:

$\mathsf{Sign}(\mathsf{sk}, \vec{m}, \tau)$: Parse sk as $(\mathbf{sk}, \mathsf{rsk}, \mathsf{rpk})$, choose $r \xleftarrow{R} \mathbb{Z}_q$, $\boxed{\text{choose } \vec{\rho} \xleftarrow{R} \mathbb{Z}_q^{\ell+1}}$, and return $\sigma \leftarrow (\sigma' \cdot g^r, \vec{c})$, where $(\sigma', \cdot) \leftarrow \mathbf{Sign}(\mathbf{sk}, \vec{m}, \tau)$ and $\vec{c} \leftarrow \mathcal{E}_{\ell+1}^2(\mathsf{rpk}, \boxed{\vec{\rho}'})$.

Transition$^{1_i \to 1_{i+1}}$**:** We show that the success probability of a distinguisher $\mathsf{D}^{1_i \to 1_{i+1}}$ is bounded by $\varepsilon_{\mathsf{mu\text{-}cpa}^{-}}(\kappa)$. To do so, we present a hybrid game, which, based on the bits chosen by the challengers, interpolates between Game 0 and Game 1_1 (resp. Game 1_i and Game 1_{i+1}). Thereby, we use $\mathcal{C}_\kappa$ to denote an $\mathsf{mt\text{-}IND\text{-}CPA}^{-}$ challenger for $\mathsf{HPRE}_{\mathsf{lin}}$ for vectors. Firstly, we run the modified Gen algorithm Gen':

$\mathsf{Gen}'(1^\kappa, \ell)$: Run $\mathsf{pp}_s \leftarrow \mathbf{Gen}(1^\kappa, \ell)$, obtain $\boxed{(\mathsf{pp}_e, \mathsf{pk}_h)}$ $\boxed{\leftarrow \mathcal{C}_\kappa}$, store pk_h and return $\mathsf{pp} \leftarrow (\mathsf{pp}_s, \mathsf{pp}_e)$.

Secondly, we run the modified VGen algorithm VGen':

$\mathsf{VGen}'(\mathsf{pp})$: Run $(\mathbf{mk}, \mathbf{aux}) \leftarrow \mathbf{VGen}(\mathsf{pp}_s)$, $\boxed{\mathsf{rpk} \leftarrow \mathsf{pk}_h}$, set $\boxed{\mathsf{rsk} \leftarrow \bot}$. Return $(\mathsf{mk}, \mathsf{aux}) \leftarrow ((\mathbf{mk}, \mathsf{rsk}), (\mathbf{aux}, \mathsf{rpk}))$.

Thirdly, we modify the oracles SG as well as SR.

$\mathsf{SG}(i)$: If $\mathtt{S}[i] \neq \bot$ return $\bot$. Otherwise, run $(\mathsf{id}_i, \mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{SGen}'(\mathsf{pp}, i, \mathtt{CU})$, set $\mathtt{S}[i] \leftarrow (\mathsf{id}_i, \mathsf{sk}_i, \mathsf{pk}_i)$ and return $(\mathsf{id}_i, \mathsf{pk}_i)$, where SGen' is defined as follows:
$\mathsf{SGen}'(\mathsf{pp}, i, \mathtt{CU})$: Run $(\mathbf{id}, \mathbf{sk}, \mathbf{pk}) \leftarrow \mathbf{SGen}(\mathsf{pp}_s)$. If $i \in \mathtt{CU}$, run $(\mathsf{rsk}, \mathsf{rpk}) \leftarrow \mathcal{G}(\mathsf{pp}_e)$. Otherwise, obtain $\mathsf{rpk} \leftarrow \mathcal{C}_\kappa.\mathsf{G}(i)$, set $\mathsf{rsk} \leftarrow \bot$. In any case set $(\mathsf{id}, \mathsf{sk}, \mathsf{pk}) \leftarrow (\mathbf{id}, (\mathbf{sk}, \mathsf{rsk}, \mathsf{rpk}), \mathbf{pk})$.

$\mathsf{SR}(i)$: If $\mathtt{S}[i] = \bot \vee \mathtt{RK}[i] \neq \bot$ return $\bot$. Otherwise, if $i \in \mathtt{CU}$ set $\mathtt{RK}[i] \leftarrow \mathsf{SRGen}(\mathtt{S}[i][2], \mathsf{aux})$ and return $\mathtt{RK}[i]$. If $i \notin \mathtt{CU}$ simulate the oracle as for $i \in \mathtt{CU}$, but with the following modified SRGen algorithm SRGen':
$\mathsf{SRGen}'(\mathsf{sk}_i, \mathsf{aux})$: Parse sk_i as $(\mathbf{sk}_i, \bot, \mathsf{rpk}_i)$ and aux as $(\mathbf{aux}, \mathsf{rpk})$. Return $\mathsf{rk}_i \leftarrow (\mathbf{rk}_i, \mathsf{prk}_i)$, where $\mathbf{rk}_i \leftarrow \mathbf{SRGen}(\mathbf{sk}_i, \mathbf{aux})$, and $\mathsf{prk}_i \leftarrow \mathcal{C}_\kappa.\mathsf{RG}(i)$.

Finally, we modify the i-th Sign run within RoS as follows:

$\mathsf{Sign}(\mathsf{sk}_i, \vec{m}, \tau)$: Parse sk_i as $(\mathbf{sk}_i, \perp, \mathsf{rpk}_i)$, choose $r \xleftarrow{R} \mathbb{Z}_q$, choose $\vec{\rho} \xleftarrow{R} \mathbb{Z}_q^{\ell+1}$, and return $\sigma \leftarrow (\sigma' \cdot g^r, \vec{c})$, where $(\sigma', \cdot) \leftarrow \mathbf{Sign}(\mathbf{sk}_i, \vec{m}, \tau)$, and $\vec{c} \leftarrow \mathcal{C}_\kappa(\vec{m}||r, \vec{\rho}, i)$.

Here, $\vec{c} \leftarrow \mathcal{C}_\kappa(\vec{m}||r, \vec{\rho}, i)$ denotes that a challenge ciphertext with respect to messages $\vec{m}||r$ and $\vec{\rho}$, and signer i is obtained from the challenger. The oracles SKey, VR, and VRKey, are simulated honestly. Now, the bit b chosen by $\mathcal{C}_\kappa$ switches between the distributions in Game i and Game $i+1$.

In Game $1_{q_{\mathsf{RoS}}}$ all ciphertexts are encryptions of random values and the g^r's unconditionally hide the signed messages in the signatures (signatures can be viewed as Pedersen commitments under randomness r), i.e., signatures are distributed as signatures on random vectors, and we are in the game where $b = 1$; the distinguishing probability between the original Game 0 and Game $1_{q_{\mathsf{RoS}}}$ is negligible. □