

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

David Derler¹(✉), Sebastian Ramacher¹, and Daniel Slamanig²

¹ IAIK, Graz University of Technology, Graz, Austria
{dderler,sramacher}@iaik.tugraz.at

² AIT Austrian Institute of Technology, Vienna, Austria
daniel.slamanig@ait.ac.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 [11] is a powerful concept which allows proxies to transform cryptographic objects under one key to cryptographic objects under another

The full version of this paper is available as IACR Cryptology ePrint Archive Report 2017/086. All authors have been supported by EU H2020 project PRISMACLOUD, grant agreement no. 644962. S. Ramacher has additionally been supported by EU H2020 project CREDENTIAL, grant agreement no. 653454.

Work done while Daniel Slamanig was still at IAIK, Graz University of Technology, Graz, Austria.

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 [12, 16], data sharing [49], publish-subscribe [15] as well as cloud-based identity management [41, 42, 47, 50]. 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. 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 and 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, \dots, 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 $\hat{d} = f(d_1, \dots, d_n)$, 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 [25, 26, 33] 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 $\hat{d} = f(d_1, \dots, d_n)$, 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.

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.

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.
- 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}_{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 [24]).

Proxy Re-Cryptography. Proxy re-encryption (PRE) [11] 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 [11], further studied in [30] and the first strongly secure constructions have been proposed by Ateniese et al. in [5]. Boneh et al. construct PRE in the symmetric setting [14]. Follow-up work focuses on even stronger (IND-CCA2 secure) schemes (cf. [17, 39, 43, 44]). 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. [7], a variant of the linearly homomorphic Paillier encryption scheme and proxy encryption in the sense of [30] 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 [27], which allows to build HPRE using the rather expensive bootstrapping technique. In concurrent work Ma et al. [40] follow this approach and propose a construction of a PRE scheme with

homomorphic properties which additionally achieves key privacy. They build upon [28] using the bootstrapping techniques in [4] 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 [11] and formally studied in [30]. Later, [6] introduced stronger security definitions, constructions and briefly discussed some applications. However, the schemes in [6] and follow up schemes [38] 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. [48] 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. [19] 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 [8,21]). Yet, they provide benefits when saving bandwidth is an issue and/or the inputs need to be hidden from evaluators (cf. [22,32]). Computing on data authenticated under different keys using so called multi-key homomorphic authenticators [25,26,33], 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 [45,46] 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 [10,31,34] improved this approach in various directions. Furthermore, [23] 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., [18,36], [29,37]. Very recently, [35] 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, $y \leftarrow^R S$ 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., $f((\vec{v}_i)_{i \in [n]}) := (f(v_{i1})_{i \in [n]}, \dots, f(v_{i\ell})_{i \in [n]})$. We let $[n] := \{1, \dots, n\}$ and let $\Pr[\Omega : E]$ denote the probability of an event E over the probability space Ω . A function $\varepsilon(\cdot) : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ 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 $\text{poly}(\cdot)$.

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 pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ 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 [1, 3] to automatically translate them to the more efficient (asymmetric) Type-3 setting. Henceforth we use \mathbf{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 the full version.

Linearly Homomorphic MACs. Our definition is inspired by [2] and covers homomorphic MACs for the family of linear function classes $\{\mathcal{F}_{\text{pp}}^{\text{lin}}\}$, further referred to as HOM-MAC.

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 , determining the message space \mathcal{M}^ℓ , function class $\mathcal{F}_{\text{pp}}^{\text{lin}}$ containing functions $f : (\mathcal{M}^\ell)^n \rightarrow \mathcal{M}^\ell$, as well as a tag space being exponentially large in κ , where $\ell, n \leq \text{poly}(\kappa)$.
- $\mathcal{G}(\text{pp})$: Takes the public parameters pp as input and outputs a secret key sk .
- $\mathcal{S}(\text{sk}, \vec{v}, \text{id}, \tau)$: Takes a MAC key sk , a vector \vec{v} , an identifier id , and a tag τ as input, and outputs a MAC μ .
- $\mathcal{C}(\text{pp}, f, (\mu_i)_{i \in [n]})$: Takes public parameters pp , a function $f \in \mathcal{F}_{\text{pp}}^{\text{lin}}$ 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 $\vec{v} = f((\vec{v}_i)_{i \in [n]})$.
- $\mathcal{V}(\text{sk}, \vec{v}, \mu, \tau, (\text{id}_i)_{i \in [n]}, f)$: Takes a MAC key sk , a vector \vec{v} , a MAC μ , a tag τ , a sequence of identifiers $(\text{id}_i)_{i \in [n]}$, and a function $f \in \mathcal{F}_{\text{pp}}^{\text{lin}}$ as input, and outputs a bit.

A linearly homomorphic MAC is required to be correct and unforgeable. Formal definitions are presented in the full version.

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 [5].

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 .

$\mathcal{G}(\text{pp})$: Takes parameters pp and outputs a key pair (rsk, rpk) .

$\mathcal{RG}(\text{rsk}_A, \text{rpk}_B)$: Takes a secret key rsk_A and a public key rpk_B and outputs a re-encryption key $\text{rk}_{A \rightarrow B}$.

$\mathcal{E}^i(\text{rpk}, m)$: Takes a public key rpk and a message m and outputs a ciphertext c .

$\mathcal{RE}(\text{rk}_{A \rightarrow B}, c_A)$: Takes a re-encryption key $\text{rk}_{A \rightarrow B}$ and a ciphertext c_A under rpk_A , and outputs a re-encrypted ciphertext c_B for rpk_B .

$\mathcal{D}^i(\text{rsk}, c)$: 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 $\text{pp} \leftarrow \mathcal{P}(1^\kappa)$, all key pairs $(\text{rsk}_A, \text{rpk}_A) \leftarrow \mathcal{G}(\text{pp})$, $(\text{rsk}_B, \text{rpk}_B) \leftarrow \mathcal{G}(\text{pp})$, all re-encryption keys $\text{rk}_{A \rightarrow B} \leftarrow \mathcal{RG}(\text{rsk}_A, \text{rpk}_B)$, all messages m it holds with probability one that

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

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 [5] (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

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

We remark that \mathcal{RG} as defined in [5] 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 a family of function classes $\{\mathcal{F}_{\text{pp}}\}$ is a tuple of PPT algorithms (Gen, SGen, VGen, Sign, Verify, SRGen, VRGen, Agg, AVerify), where Verify is optional. They are defined as follows:*

- $\text{Gen}(1^\kappa, \ell)$: Takes security parameter κ and vector length ℓ and outputs parameters pp , determining the message space \mathcal{M}^ℓ , function class \mathcal{F}_{pp} containing functions $f : (\mathcal{M}^\ell)^n \rightarrow \mathcal{M}^\ell$, as well as a tag space being exponentially large in κ , where $\ell, n \leq \text{poly}(\kappa)$.
- $\text{SGen}(\text{pp})$: Takes parameters pp as input, and outputs a signer key $(\text{id}, \text{sk}, \text{pk})$.
- $\text{VGen}(\text{pp})$: Takes parameters pp , and outputs a MAC key mk and auxiliary information aux .
- $\text{Sign}(\text{sk}, \vec{m}, \tau)$: Takes a signer secret key sk , a message vector \vec{m} , and a tag τ as input, and outputs a signature σ .
- $\text{Verify}(\text{pk}, \vec{m}, \tau, \sigma)$: 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 .
- $\text{SRGen}(\text{sk}_i, \text{aux})$: Takes a signer secret key sk_i , some auxiliary information aux , and outputs a re-encryption key rk_i .
- $\text{VRGen}(\text{pk}_i, \text{mk}, \text{rk}_i)$: 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 .
- $\text{Agg}((\text{ak}_i)_{i \in [n]}, (\sigma_i)_{i \in [n]}, \tau, f)$: Takes n aggregation keys $(\text{ak}_i)_{i \in [n]}$, n signatures $(\sigma_i)_{i \in [n]}$, a tag τ , and a function $f \in \mathcal{F}_{\text{pp}}$ as input, and outputs an aggregate authenticated message vector Λ .
- $\text{AVerify}(\text{mk}, \Lambda, \text{ID}, f)$: Takes a MAC key mk , an aggregate authenticated message vector Λ , n identifiers $\text{ID} = (\text{id}_i)_{i \in [n]}$, and a function $f \in \mathcal{F}_{\text{pp}}$. It outputs a message vector and a tag (\vec{m}, τ) 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.

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

SKey(i): If $i \notin \text{HU}$ return \perp . Otherwise return $\text{S}[i]$.
Sig((j_i) $_{i \in [n]}$, (\vec{m}_i) $_{i \in [n]}$): If $\text{S}[j_i] = \perp$ 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 ($\sigma_{j_i} \leftarrow \text{Sign}(\text{S}[j_i][2], \vec{m}_i, \tau)$) $_{i \in [n]}$, set $\text{SIG}[\tau] \leftarrow \text{SIG}[\tau] \cup \{(\vec{m}_i, \text{S}[j_i][1])\}$ for $i \in [n]$, and return (σ_{j_i}) $_{i \in [n]}$ and τ .
RoS((j_i) $_{i \in [n]}$, (\vec{m}_i) $_{i \in [n]}$, b): If $\text{S}[j_i] = \perp$ or $j_i \in \text{CU}$ for any $i \in [n]$ return \perp . Otherwise sample τ uniformly at random and if $b = 0$ compute ($\sigma_{j_i} \leftarrow \text{Sign}(\text{S}[j_i][2], \vec{m}_i, \tau)$) $_{i \in [n]}$. Else choose (\vec{r}_i) $_{i \in [n]} \xleftarrow{R} (\mathcal{M}^\ell)^n$ where \mathcal{M} is the message space and compute ($\sigma_{j_i} \leftarrow \text{Sign}(\text{S}[j_i][2], \vec{r}_i, \tau)$) $_{i \in [n]}$. Finally, return (σ_{j_i}) $_{i \in [n]}$.
SR(i): If $\text{S}[i] = \perp \vee \text{RK}[i] \neq \perp$ return \perp . Else, set $\text{RK}[i] \leftarrow \text{SRGen}(\text{S}[i][2], \text{aux})$ and return $\text{RK}[i]$.
VR(i): If $\text{S}[i] = \perp \vee \text{RK}[i] = \perp \vee \text{AK}[i] \neq \perp$ return \perp . Else, set $\text{AK}[i] \leftarrow \text{VRGen}(\text{S}[i][3], \text{mk}, \text{RK}[i])$.
VRKey(i): Return $\text{AK}[i]$.
A((σ_{j_i}) $_{i \in [n]}$, (j_i) $_{i \in [n]}$, τ , f): Check validity of all σ_{j_i} , whether $f \in \mathcal{F}_{\text{pp}}$, whether $\text{SIG}[\tau] = \perp$, and return \perp if any check fails. Further, check whether there exists $u, v \in [n]$, $u \neq v$ so that $j_u = j_v$ and return \perp if so. Obtain (ak_{j_i}) $_{i \in [n]}$ from AK and return \perp if $\text{AK}[j_i] = \perp$ for any $i \in [n]$. Set $\text{SIG}[\tau] \leftarrow \bigcup_{i \in [n]} \{(\vec{m}_{j_i}, \text{S}[j_i][1])\}$ and return $\Lambda \leftarrow \text{Agg}((\text{ak}_{j_i})_{i \in [n]}, (\sigma_{j_i})_{i \in [n]}, \tau, f)$.

We require a HPRA to be correct, signer unforgeable, aggregator unforgeable, and input private. We formally define those notions below. Intuitively, correctness requires that everything works as intended if everyone behaves honestly.

Definition 5 (Correctness). *A HPRA for a family of function classes $\{\mathcal{F}_{\text{pp}}\}$ is correct, if for all κ , for all $\ell \leq \text{poly}(\kappa)$, for all $\text{pp} \leftarrow \text{Gen}(1^\kappa, \ell)$ determining \mathcal{F}_{pp} , for all $n \leq \text{poly}(\kappa)$, for all (($\text{id}_i, \text{sk}_i, \text{pk}_i$) $\leftarrow \text{SGen}(\text{pp})$) $_{i \in [n]}$, for all (mk, aux) $\leftarrow \text{VGen}(\text{pp})$, for all (\vec{m}_i) $_{i \in [n]}$, for all τ , for all ($\sigma_i \leftarrow \text{Sign}(\text{sk}_i, \vec{m}_i, \tau)$) $_{i \in [n]}$, for all ($\text{ak}_i \leftarrow \text{VRGen}(\text{pk}_i, \text{mk}, \text{SRGen}(\text{sk}_i, \text{aux}))$) $_{i \in [n]}$, for all $f \in \mathcal{F}_{\text{pp}}$, for all $\Lambda^* \leftarrow \text{Agg}((\text{ak}_i)_{i \in [n]}, (\sigma_i)_{i \in [n]}, \tau, f)$ it holds that $(\text{Verify}(\text{pk}_i, \vec{m}_i, \tau, \sigma_i) = 1)$ $_{i \in [n]}$ and that $\text{AVerify}(\text{mk}, \Lambda^*, \text{ID}, f) = 1$, where we sometimes omit to make the domains of the values over which we quantify explicit for brevity.*

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 $f \in \mathcal{F}_{\text{pp}}$ 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

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.

Definition 6 (T-Unforgeability). Let $T \in \{\text{Signer}, \text{Aggregator}\}$. A HPRA for family of function classes $\{\mathcal{F}_{\text{pp}}\}$ is T -unforgeable, if for all PPT adversaries \mathcal{A} there is a negligible function $\epsilon(\cdot)$ such that

$$\Pr \left[\begin{array}{l} \text{pp} \leftarrow \text{Gen}(1^\kappa, \ell), \\ (\text{mk}, \text{aux}) \leftarrow \text{VGen}(\text{pp}), \\ (\Delta^*, \text{ID}^*, f^*) \leftarrow \mathcal{A}^{\mathcal{O}_T}(\text{pp}, \text{aux}), \\ (\vec{m}, \tau) \leftarrow \text{AVerify}(\text{mk}, \Delta^*, \text{ID}^*, f^*) \end{array} : \begin{array}{l} \vec{m} \neq \perp \wedge f^* \in \mathcal{F}_{\text{pp}} \wedge \\ 0 < n, \ell \leq \text{poly}(\kappa) \wedge \\ (\nexists (\vec{m}_j)_{j \in [n]} : (\forall j \in [n] : \\ (\vec{m}_j, \text{id}_j^*) \in \text{SIG}[\tau]) \wedge \\ f^*((\vec{m}_j)_{j \in [n]}) = \vec{m}) \end{array} \leq \epsilon(\kappa), \right.$$

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

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 7 (Input Privacy). A HPRA for a family of function classes $\{\mathcal{F}_{\text{pp}}\}$ is input private if for all $\kappa \in \mathbb{N}$, for all $\ell \leq \text{poly}(\kappa)$, for all $\text{pp} \leftarrow \text{Gen}(1^\kappa, \ell)$ determining \mathcal{F}_{pp} , for all $f \in \mathcal{F}_{\text{pp}}$ implicitly defining n , for all tags τ , and for all $(\vec{m}_{11}, \dots, \vec{m}_{n1})$ and $(\vec{m}_{12}, \dots, \vec{m}_{n2})$ where $f(\vec{m}_{11}, \dots, \vec{m}_{n1}) = f(\vec{m}_{12}, \dots, \vec{m}_{n2})$, for all $(\text{mk}, \text{aux}) \leftarrow \text{VGen}(\text{pp})$, for all $((\text{sk}_i, \text{pk}_i) \leftarrow \text{SGen}(\text{pp}))_{i \in [n]}$, $(\text{ak}_i \leftarrow \text{SRGen}(\text{sk}_i, \text{aux}, \text{VRGen}(\text{pk}_i, \text{mk})))_{i \in [n]}$, the following distributions are identical:

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

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 8 (Output Privacy). A HPRA for a family of function classes $\{\mathcal{F}_{\text{pp}}\}$ is output private, if for all PPT adversaries \mathcal{A} there is a negligible function $\epsilon(\cdot)$ such that:

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

4 An Input Private Scheme for Linear Functions

Now we present our first HPRA for the family of linear function classes $\{\mathcal{F}_{\text{pp}}^{\text{lin}}\}$. 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

$\mathcal{P}(\kappa, \ell)$: Run $\text{BG} \leftarrow \text{BGGen}(1^\kappa)$, fix $H : \mathbb{Z}_q \rightarrow \mathbb{G}$, choose $(g_i)_{i \in [\ell]} \xleftarrow{R} (\mathbb{G}^*)^\ell$, and return $\text{pp} \leftarrow (\text{BG}, H, (g_i)_{i \in [\ell]}, \ell)$.
$\mathcal{G}(\text{pp})$: Choose $\alpha \xleftarrow{R} \mathbb{Z}_p$ and return $\text{sk} \leftarrow (\text{pp}, \alpha)$.
$\mathcal{S}(\text{sk}, \vec{v}, \text{id}, \tau)$: Parse sk as (pp, α) and return $\mu \leftarrow e(H(\tau \text{id}) \cdot \prod_{j \in [\ell]} g_j^{v_j}, g^\alpha)$.
$\mathcal{C}(\text{pp}, f, (\mu_i)_{i \in [n]})$: Parse f as $(\omega_i)_{i \in [n]}$ and return $\mu \leftarrow \prod_{i \in [n]} \mu_i^{\omega_i}$.
$\mathcal{V}(\text{sk}, \vec{v}, \mu, \tau, (\text{id}_i)_{i \in [n]}, f)$: Parse sk as (pp, α) , f as $(\omega_i)_{i \in [n]}$, and output 1 if the following holds, and 0 otherwise: $\mu = e(\prod_{i \in [n]} H(\tau \text{id}_i)^{\omega_i} \prod_{j \in [\ell]} g_j^{v_j}, g^\alpha)$

Scheme 1. Linearly homomorphic MAC based on [13].

submodule. Both build upon the ideas used in the signature scheme presented in [13].

A Suitable Linearly Homomorphic MAC. We present our HOM-MAC in Scheme 1. We can not recycle the security arguments from [13] as we require the ability to submit arbitrary tags τ to the Sig oracle. Thus we directly prove unforgeability.

Lemma 1 (Proven in the full version). *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 family of linear function classes $\{\mathcal{F}_{\text{pp}}^{\text{lin}}\}$. 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 $\text{Eval}(\cdot, \cdot)$, which defines how to compute $f \in \mathcal{F}_{\text{pp}}^{\text{lin}}$ on the message vector. When directly instantiating Scheme 2, Eval is defined as $\text{Eval}(f, (\vec{m}_i)_{i \in [n]}) := f((\vec{m}_i)_{i \in [n]})$.

Theorem 1 (Proven in the full version). *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 the family of linear function classes $\{\mathcal{F}_{\text{pp}}^{\text{lin}}\}$ 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 the family of linear function classes $\{\mathcal{F}_{\text{pp}}^{\text{lin}}\}$. Based on this, we extend Scheme 2 to additionally provide output privacy.

$\text{Gen}(1^\kappa, \ell)$: Run $\text{BG} \leftarrow \text{BGGen}(1^\kappa)$, fix $H : \mathbb{Z}_q \rightarrow \mathbb{G}$, choose $(g_i)_{i \in [\ell]} \xleftarrow{R} \mathbb{G}^\ell$, and return $\text{pp} \leftarrow (\text{BG}, H, (g_i)_{i \in [\ell]}, \ell)$.
$\text{SGen}(\text{pp})$: Choose $\beta \xleftarrow{R} \mathbb{Z}_q$, set $\text{id} \leftarrow g^\beta$, $\text{pk} \leftarrow (\text{pp}, g^\beta, g^{1/\beta})$, $\text{sk} \leftarrow (\text{pk}, \beta)$, and return $(\text{id}, \text{sk}, \text{pk})$.
$\text{VGen}(\text{pp})$: Choose $\alpha \xleftarrow{R} \mathbb{Z}_q$, set $\text{aux} \leftarrow \emptyset$, $\text{mk} \leftarrow (\text{pp}, \alpha)$ and return (mk, aux) .
$\text{Sign}(\text{sk}, \vec{m}, \tau)$: Parse sk as $((\text{BG}, H, (g_i)_{i \in [\ell]}, \ell), g^\beta, \cdot, \beta)$, compute and return $\sigma \leftarrow (\sigma', \vec{m})$, where
$\sigma' \leftarrow \left(H(\tau \ g^\beta) \cdot \prod_{i=1}^{\ell} g_i^{m_i} \right)^\beta.$
$\text{Verify}(\text{pk}, \vec{m}, \tau, \sigma)$: Parse pk as $((\text{BG}, H, (g_i)_{i \in [\ell]}, \ell), g^\beta, \cdot)$, and σ as (σ', \vec{m}') , and return 1 if the following holds and 0 otherwise:
$e(H(\tau \ g^\beta) \cdot \prod_{i=1}^{\ell} g_i^{m_i}, g^\beta) = e(\sigma, g) \quad \wedge \quad \vec{m} = \vec{m}'.$
$\text{SRGen}(\text{sk}_i, \text{aux})$: Return $\text{rk}_i \leftarrow \emptyset$.
$\text{VRGen}(\text{pk}_i, \text{mk}, \text{rk}_i)$: Parse pk_i as $(\cdot, \cdot, g^{1/\beta_i})$, mk as (\cdot, α) , and return $\text{ak}_i \leftarrow (g^{1/\beta_i})^\alpha$.
$\text{Agg}((\text{ak}_i)_{i \in [n]}, (\sigma_i)_{i \in [n]}, \tau, f)$: Parse f as $(\omega_i)_{i \in [n]}$, and for $i \in [n]$ parse σ_i as (σ'_i, \vec{m}_i) and return $\Lambda \leftarrow (\text{Eval}(f, (\vec{m}_i)_{i \in [n]}), \mu, \tau)$, where
$\mu \leftarrow \prod_{i \in [n]} e(\sigma_i'^{\omega_i}, \text{ak}_i).$
$\text{AVerify}(\text{mk}, \Lambda, \text{ID}, f)$: Parse mk as (pp, α) , Λ as (\vec{m}, μ, τ) , ID as $(g^{\beta_i})_{i \in [n]}$ and f as $(\omega_i)_{i \in [n]}$ and return (\vec{m}, τ) if the following holds, and (\perp, \perp) otherwise:
$\mu' = \left(\prod_{i=1}^n e(g^{\omega_i}, H(\tau \ g^{\beta_i})) \cdot e\left(\prod_{i=1}^{\ell} g_i^{m_i}, g\right) \right)^\alpha$

Scheme 2. HPRA scheme for the family of linear function families $\{\mathcal{F}_{\text{pp}}^{\text{lin}}\}$ parametrized by Eval.

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 9 (HPRE). *A HPRE for the family of function classes $\{\mathcal{F}_{\text{pp}}\}$ is a PRE with an additional evaluation algorithm \mathcal{EV} .*

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

Additionally, we require the following compactness notion (analogous to [20]).

Definition 10 (Compactness). A HPRE for the family of function classes $\{\mathcal{F}_{\mathbf{pp}}\}$ is called compact if for all $\mathbf{pp} \leftarrow \mathcal{P}(1^\kappa)$ and for all $f \in \mathcal{F}_{\mathbf{pp}}$ the running time of the algorithms \vec{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 11 (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

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

where the environment holds an initially empty list HU . \mathcal{G} and RG are defined as:

$\mathcal{G}(i)$: If $\text{HU}[i] \neq \perp$ return \perp . Otherwise, run $(\text{sk}_i, \text{pk}_i) \leftarrow \mathcal{G}(\mathbf{pp})$, set $\text{HU}[i] \leftarrow (\text{sk}_i, \text{pk}_i)$, and return pk_i .
 $\text{RG}(i)$: If $\text{HU}[i] = \perp$ return \perp . Otherwise, set $\text{rk}_{i \rightarrow h} \leftarrow \mathcal{RG}(\text{HU}[i][1], \text{pk}_h)$ and return $\text{rk}_{i \rightarrow j}$.

Lemma 2 (proven in the full version). Every IND-CPA^- (and thus every IND-CPA) secure PRE also satisfies mt-IND-CPA^- security.

HPREConstruction for the Family of Linear Function Classes. We state our construction in Scheme 3. Essentially, we build upon the PRE scheme in [5, third attempt] and turn it into a HPRE for the family of linear function classes $\{\mathcal{F}_{\mathbf{pp}}^{\text{lin}}\}$, henceforth referred to as HPRE_{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.

As \mathcal{EV} is a public algorithm it does not influence IND-CPA security. Thus, our argumentation is identical to [5] and we can use the following theorem.

$\mathcal{P}(1^\kappa)$: Run $\text{BG} \leftarrow \text{BGen}(1^\kappa)$, and return $\text{pp} \leftarrow \text{BG}$.
$\mathcal{G}(\text{pp})$: Choose $(a_1, a_2) \xleftarrow{R} \mathbb{Z}_q^2$, and return $(\text{rsk}_A, \text{rpk}_A) \leftarrow ((a_1, a_2), (\mathbf{g}^{a_1}, g^{a_2}))$.
$\mathcal{RG}(\text{rsk}_A, \text{rpk}_B)$: Parse rsk_A as (a_{1A}, \cdot) and rpk_B as $(\cdot, g^{a_{2B}})$ and return $\text{rk}_{A \rightarrow B} \leftarrow (g^{a_{2B}})^{a_{1A}}$.
$\mathcal{E}^1(\text{rpk}, m)$: Parse rpk as $(\mathbf{g}^{a_1}, \cdot)$, choose $k \xleftarrow{R} \mathbb{Z}_q$, and return $c \leftarrow (\mathbf{g}^k, \mathbf{g}^m \cdot (\mathbf{g}^{a_1})^k, 1)$
$\mathcal{E}^2(\text{rpk}, m)$: Parse rpk as $(\mathbf{g}^{a_1}, \cdot)$, choose $k \xleftarrow{R} \mathbb{Z}_q$, and return $c \leftarrow (g^k, \mathbf{g}^m \cdot (\mathbf{g}^{a_1})^k, 2)$
$\mathcal{RE}(\text{rk}_{A \rightarrow B}, c_A)$: Parse c_A as $(c_1, c_2, 2)$ and return $c \leftarrow (e(c_1, \text{rk}_{A \rightarrow B}), c_2, R)$
$\mathcal{D}^1(\text{rsk}, c)$: 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 $c_3 = R$.
$\mathcal{D}^2(\text{rsk}, c)$: Parse c as $(c_1, c_2, 2)$ and rsk as (a_1, a_2) , and return $\mathbf{g}^m \leftarrow c_2 \cdot e(g, c_1^{-a_1})$.
$\mathcal{EV}(\text{pp}, f, \vec{c})$: Parse f as $(\omega_1, \dots, \omega_n)$ 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_{lin} based on [5, third attempt].

Theorem 2 (cf. [5]). *If the eDBDH assumption holds in $(\mathbb{G}, \mathbb{G}_T)$ then Scheme 3 is an IND-CPA secure HPRE_{lin} .*

We note that compactness of Scheme 3 (Definition 10) is easy to verify.

HPRE_{lin} for Vectors. We extend HPRE_{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 [9] and applying it to HPRE_{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 $\text{rsk} \leftarrow (\text{rsk}_i)_{i \in [\ell]} = ((a_{1i}, a_{2i}))_{i \in [\ell]}$, $\text{rpk} \leftarrow (\text{rpk}_i)_{i \in [\ell]} = ((\mathbf{g}^{a_{1i}}, g^{a_{2i}}))_{i \in [\ell]}$, where $(a_{1i}, a_{2i})_{i \in [\ell]} \xleftarrow{R} (\mathbb{Z}_q^2)^\ell$. First and second level encryption are defined as

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

Decryption $\mathcal{D}_\ell^j(\cdot, \cdot)$ of a ciphertext $(c[1], (c[i+1])_{i \in [\ell]}, j)$ is defined as $\mathcal{D}_\ell^1(\text{rsk}, \vec{c}) := (c[i+1] \cdot c[1]^{-\text{rsk}_i[1]})_{i \in [\ell]}$, and $\mathcal{D}_\ell^2(\text{rsk}, \vec{c}) := (c[i+1] \cdot e(c[1], g^{-\text{rsk}_i[1]}))_{i \in [\ell]}$. Re-encryption key generation is $\mathcal{RG}_\ell(\text{rsk}_A, \text{rpk}_B) := (((\text{rpk}_B)_i[2])^{(\text{rsk}_A)_i[1]})_{i \in [\ell]}$. From a second level ciphertext \vec{c}_A for A and a re-encryption key $\text{rk}_{A \rightarrow B}$, one can compute a ciphertext \vec{c}_B for B as $\vec{c}_B \leftarrow \mathcal{RE}(\text{rk}_{A \rightarrow B}, \vec{c}_A) := ((e(c_A[1], \text{rk}_{A \rightarrow B}[i]), c_A[i+1]))_{i \in [\ell]}$. Note that re-encrypted ciphertexts have a different form.

Thus we do not need to add the level as suffix. Decryption $\mathcal{D}_\ell^1(\cdot, \cdot)$ for re-encrypted ciphertexts is $\mathcal{D}_\ell^1(\text{rsk}, (c_i)_{i \in [\ell]}) := (c_i[2] \cdot c_i[1]^{-1/\text{rsk}_i[2]})_{i \in [\ell]}$.

Theorem 3. *If the eDBDH assumption holds, then the extension of HPRE_{lin} as described above, yields an IND-CPA secure HPRE_{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 ($1 \leq i \leq \ell$) the i -th ciphertext component is exchanged by random under the original strategy in [5].

Combining the theorem above with Lemma 2 yields:

Corollary 1. *The extension of HPRE_{lin} as described above yields an mt-IND-CPA⁻ secure HPRE_{lin} for vectors.*

5.2 Putting the Pieces Together: Output Privacy

Our idea is to combine Scheme 2 with the HPRE_{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 $g^{f(\dots, r, \dots)}$, 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 HPRE_{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 HPRE_{lin} . More importantly, by executing the $\mathcal{E}\mathcal{V}$ 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 $f(\dots, r, \dots)$ upon decryption.

Note on the Instantiation. Augmenting Scheme 2 to obtain Scheme 4 using HPRE_{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 recovered. 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 the full version). *If Scheme 2 is signer and aggregator unforgeable, and HPRE_{lin} for vectors is mt-IND-CPA⁻ secure, then Scheme 4 is a signer and aggregator unforgeable, input and output private HPRA for class \mathcal{F}_{lin} .*

$\text{Gen}(1^\kappa, \ell):$ Fix a homomorphic PRE = $(\mathcal{P}, \mathcal{G}, \vec{\mathcal{E}}, \vec{\mathcal{D}}, \mathcal{RG}, \mathcal{RE}, \mathcal{EV})$ for class \mathcal{F}_{in} and the HPRA $(\mathcal{EV}) = (\text{Gen}, \text{SGen}, \text{VGen}, \text{Sign}, \text{Verify}, \text{SRGen}, \text{VRGen}, \text{Agg}, \text{AVerify})$ from Scheme 2 such that $\mathcal{M}_{\text{PRA}} \subseteq \mathcal{M}_{\text{PRE}}$, run $\text{pp}_s \leftarrow \text{Gen}(1^\kappa, \ell)$, $\text{pp}_e \leftarrow \mathcal{P}(1^\kappa, \ell + 1)$, and return $\text{pp} \leftarrow (\text{pp}_s, \text{pp}_e)$.
$\text{SGen}(\text{pp}):$ Run $(\text{id}, \text{sk}, \text{pk}) \leftarrow \text{SGen}(\text{pp}_s)$, $(\text{rsk}, \text{rpk}) \leftarrow \mathcal{G}(\text{pp}_e)$, and return $(\text{id}, \text{sk}, \text{pk}) \leftarrow (\text{id}, (\text{sk}, \text{rsk}, \text{rpk}), \text{pk})$.
$\text{VGen}(\text{pp}):$ Run $(\text{mk}, \text{aux}) \leftarrow \text{VGen}(\text{pp}_s)$, $(\text{rsk}, \text{rpk}) \leftarrow \mathcal{G}(\text{pp}_e)$, and return $(\text{mk}, \text{aux}) \leftarrow ((\text{mk}, \text{rsk}), (\text{aux}, \text{rpk}))$.
$\text{Sign}(\text{sk}, \vec{m}, \tau):$ Parse sk as $(\text{sk}, \cdot, \text{rpk})$, choose $r \xleftarrow{R} \mathbb{Z}_q$, and return $\sigma \leftarrow (\sigma' \cdot g^r, \vec{c})$, where
$(\sigma', \cdot) \leftarrow \text{Sign}(\text{sk}, \vec{m}, \tau) \text{ and } \vec{c} \leftarrow \mathcal{E}_{\ell+1}^2(\text{rpk}, \vec{m} \ r).$
$\text{SRGen}(\text{sk}_i, \text{aux}):$ Parse sk_i as $(\text{sk}_i, \text{rsk}_i, \text{rpk}_i)$ and aux as (aux, rpk) . Obtain $\text{rk}_i \leftarrow \text{SRGen}(\text{sk}_i, \text{aux})$ and $\text{prk}_i \leftarrow \mathcal{RG}(\text{rsk}_i, \text{rpk})$, and return $\text{rk}_i \leftarrow (\text{rk}_i, \text{prk}_i)$.
$\text{VRGen}(\text{pk}_i, \text{mk}, \text{rk}_i):$ Parse pk_i as pk_i and mk as (mk, \cdot) , obtain $\text{ak}_i \leftarrow \text{VRGen}(\text{pk}_i, \text{mk})$ and return $\text{ak}_i \leftarrow (\text{ak}_i, \text{rk}_i)$.
$\text{Agg}((\text{ak}_i)_{i \in [n]}, (\sigma_i)_{i \in [n]}, \tau, f):$ For $i \in [n]$ parse ak_i as $(\text{ak}_i, (\text{rk}_i, \text{prk}_i))$, σ_i as (σ'_i, \vec{c}_i) . Output $\Lambda \leftarrow (\vec{c}', \mu, \tau)$, where
$(\vec{c}'_i \leftarrow \mathcal{RE}(\text{prk}_i, \vec{c}_i))_{i \in [n]}, (\vec{c}', \mu, \tau) \leftarrow \text{Agg}((\text{ak}_i)_{i \in [n]}, (\sigma'_i, \vec{c}'_i)_{i \in [n]}, f).$
$\text{AVerify}(\text{mk}, \Lambda, \text{ID}, f):$ Parse mk as (mk, rsk) and Λ as (\vec{c}, μ, τ) , obtain $\vec{m}' \ r \leftarrow \mathcal{D}_{\ell+1}^1(\text{rsk}, \vec{c})$ and return (\vec{m}, τ) if the following holds, and (\perp, \perp) otherwise:
$\text{AVerify}(\text{mk}, (\vec{m}, \mu \cdot (\mathbf{g}^r)^{-1}, \tau), \text{ID}, f) = 1$

Scheme 4. Output private HPRA scheme for the family of linear function classes $\{\mathcal{F}_{\text{pp}}^{\text{lin}}\}$ with \mathcal{F}_{in} with $\text{Eval}(\cdot, \cdot) := \mathcal{EV}(\text{pp}, \cdot, \cdot)$

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

1. Abe, M., Hoshino, F., Ohkubo, M.: Design in Type-I, run in Type-III: fast and scalable bilinear-type conversion using integer programming. In: CRYPTO 2016 (2016)
2. Agrawal, S., Boneh, D.: Homomorphic MACs: MAC-based integrity for network coding. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 292–305. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01957-9_18
3. Akinyele, J.A., Garman, C., Hohenberger, S.: Automating fast and secure translations from Type-I to Type-III pairing schemes. In: CCS 2015 (2015)
4. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 297–314. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_17
5. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.* **9**(1), 1–30 (2006)
6. Ateniese, G., Hohenberger, S.: Proxy re-signatures: new definitions, algorithms, and applications. In: CCS 2015 (2005)
7. Ayday, E., Raisaro, J.L., Hubaux, J., Rougemont, J.: Protecting and evaluating genomic privacy in medical tests and personalized medicine. In: WPES 2013 (2013)
8. Backes, M., Fiore, D., Reischuk, R.M.: Verifiable delegation of computation on outsourced data. In: CCS 2013 (2013)
9. Bellare, M., Boldyreva, A., Kurosawa, K., Staddon, J.: Multirecipient encryption schemes: How to save on bandwidth and computation without sacrificing security. *IEEE Trans. Inf. Theory* **53**(11), 3927–3943 (2007)
10. Benhamouda, F., Joye, M., Libert, B.: A new framework for privacy-preserving aggregation of time-series data. *ACM Trans. Inf. Syst. Secur.* **18**(3), 21 (2016)
11. Blaze, M., Bleumer, G., Strauss, M.: Divertible protocols and atomic proxy cryptography. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 127–144. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0054122>
12. Blazy, O., Bultel, X., Lafourcade, P.: Two secure anonymous proxy-based data storages. In: SECRYPT, pp. 251–258 (2016)
13. Boneh, D., Freeman, D., Katz, J., Waters, B.: Signing a linear subspace: signature schemes for network coding. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 68–87. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00468-1_5
14. Boneh, D., Lewi, K., Montgomery, H., Raghunathan, A.: Key homomorphic PRFs and their applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 410–428. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_23
15. Borceaa, C., Gupta, A.B.D., Polyakova, Y., Rohloff, K., Ryana, G.: Picador: End-to-end encrypted publish-subscribe information distribution with proxy re-encryption. *Future Gener. Comp. Syst.* **62**, 119–127 (2016)
16. Canard, S., Devigne, J.: Highly privacy-protecting data sharing in a tree structure. *Future Gener. Comp. Syst.* **62**, 119–127 (2016)
17. Canetti, R., Hohenberger, S.: Chosen-ciphertext secure proxy re-encryption. In: CCS, pp. 185–194 (2007)
18. Castelluccia, C., Chan, A.C.F., Mykletun, E., Tsudik, G.: Efficient and provably secure aggregation of encrypted data in wireless sensor networks. *ACM Trans. Sen. Netw.* **5**(3) (2009)

19. Catalano, D.: Homomorphic signatures and message authentication codes. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 514–519. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10879-7_29
20. Catalano, D., Fiore, D.: Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In: CCS 2015 (2015)
21. Catalano, D., Fiore, D., Warinschi, B.: Homomorphic signatures with efficient verification for polynomial functions. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 371–389. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_21
22. Catalano, D., Marcedone, A., Puglisi, O.: Authenticating computation on groups: new homomorphic primitives and applications. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 193–212. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_11
23. Chan, T.-H.H., Shi, E., Song, D.: Privacy-preserving stream aggregation with fault tolerance. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 200–214. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32946-3_15
24. Danezis, G., Livshits, B.: Towards ensuring client-side computational integrity. In: CCSW 2011 (2011)
25. Derler, D., Slamanig, D.: Key-homomorphic signatures and applications to multi-party signatures. Cryptology ePrint Archive **2016**, 792 (2016)
26. Fiore, D., Mitrokotsa, A., Nizzardo, L., Pagnin, E.: Multi-key homomorphic authenticators. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 499–530. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_17
27. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC 2009 (2009)
28. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_5
29. Günther, F., Manulis, M., Peter, A.: Privacy-enhanced participatory sensing with collusion resistance and data aggregation. In: Gritzalis, D., Kiayias, A., Askoxylakis, I. (eds.) CANS 2014. LNCS, vol. 8813, pp. 321–336. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12280-9_21
30. Ivan, A., Dodis, Y.: Proxy cryptography revisited. In: NDSS 2003 (2003)
31. Joye, M., Libert, B.: A scalable scheme for privacy-preserving aggregation of time-series data. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 111–125. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_10
32. Lai, J., Deng, R.H., Pang, H., Weng, J.: Verifiable computation on outsourced encrypted data. In: Kutyłowski, M., Vaidya, J. (eds.) ESORICS 2014. LNCS, vol. 8712, pp. 273–291. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11203-9_16
33. Lai, R.W.F., Tai, R.K.H., Wong, H.W.H., Chow, S.S.M.: A zoo of homomorphic signatures: Multi-key and key-homomorphism. Cryptology ePrint Archive, Report 2016/834 (2016)
34. Leontiadis, I., Elkhiyaoui, K., Molva, R.: Private and dynamic time-series data aggregation with trust relaxation. In: Gritzalis, D., Kiayias, A., Askoxylakis, I. (eds.) CANS 2014. LNCS, vol. 8813, pp. 305–320. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12280-9_20

35. Leontiadis, I., Elkhyaoui, K., Önen, M., Molva, R.: PUDA – privacy and unforgeability for data aggregation. In: Reiter, M., Naccache, D. (eds.) CANS 2015. LNCS, vol. 9476, pp. 3–18. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26823-1_1
36. Li, Q., Cao, G.: Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error. In: De Cristofaro, E., Wright, M. (eds.) PETS 2013. LNCS, vol. 7981, pp. 60–81. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39077-7_4
37. Li, Q., Cao, G., Porta, T.F.L.: Efficient and privacy-aware data aggregation in mobile sensing. *IEEE Trans. Dep. Sec. Comput.* **11**(2), 115–129 (2014)
38. Libert, B., Vergnaud, D.: Multi-use unidirectional proxy re-signatures. In: CCS 2008 (2008)
39. Libert, B., Vergnaud, D.: Unidirectional chosen-ciphertext secure proxy re-encryption. *IEEE Trans. Inf. Theory* **57**(3), 1786–1802 (2011)
40. Ma, C., Li, J., Ouyang, W.: A homomorphic proxy re-encryption from lattices. In: Chen, L., Han, J. (eds.) ProvSec 2016. LNCS, vol. 10005, pp. 353–372. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47422-9_21
41. Nuñez, D., Agudo, I.: BlindIdM: a privacy-preserving approach for identity management as a service. *Int. J. Inf. Sec.* **13**(2), 199–215 (2014)
42. Nuñez, D., Agudo, I., Lopez, J.: Integrating OpenID with proxy re-encryption to enhance privacy in cloud-based identity services. In: CloudCom, pp. 241–248 (2012)
43. Nuñez, D., Agudo, I., Lopez, J.: A parametric family of attack models for proxy re-encryption. In: CSF, pp. 290–301 (2015)
44. Nuñez, D., Agudo, I., Lopez, J.: On the application of generic CCA-secure transformations to proxy re-encryption. *Secur. Commun. Netw.* **9**(12), 1769–1785 (2016)
45. Rastogi, V., Nath, S.: Differentially private aggregation of distributed time-series with transformation and encryption. In: SIGMOD 2010 (2010)
46. Shi, E., Chan, T.H.H., Rieffel, E.G., Chow, R., Song, D.: Privacy-preserving aggregation of time-series data. In: NDSS 2011 (2011)
47. Slamanig, D., Stranacher, K., Zwattendorfer, B.: User-centric identity as a service-architecture for eIDs with selective attribute disclosure. In: SACMAT, pp. 153–164 (2014)
48. Walfish, M., Blumberg, A.J.: Verifying computations without reexecuting them. *Commun. ACM* **58**(2), 74–84 (2015)
49. Xu, P., Xu, J., Wang, W., Jin, H., Susilo, W., Zou, D.: Generally hybrid proxy re-encryption: a secure data sharing among cryptographic clouds. In: AsiaCCS, pp. 913–918 (2016)
50. Zwattendorfer, B., Slamanig, D., Stranacher, K., Hörandner, F.: A federated cloud identity broker-model for enhanced privacy via proxy re-encryption. In: De Decker, B., Zúquete, A. (eds.) CMS 2014. LNCS, vol. 8735, pp. 92–103. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44885-4_8