Valerie Fetzer, Marcel Keller, Sven Maier, Markus Raiber, Andy Rupp*, and Rebecca Schwerdt

PUBA: Privacy-Preserving User-Data Bookkeeping and Analytics

Abstract: In this paper we propose Privacy-preserving User-data Bookkeeping & Analytics (PUBA), a building block destined to enable the implementation of business models (e.g., targeted advertising) and regulations (e.g., fraud detection) requiring user-data analysis in a privacy-preserving way. In PUBA, users keep an unlinkable but authenticated cryptographic logbook containing their historic data on their device. This logbook can only be updated by the operator while its content is not revealed. Users can take part in a privacy-preserving analytics computation, where it is ensured that their logbook is up-to-date and authentic while the potentially secret analytics function is verified to be privacy-friendly. Taking constrained devices into account, users may also outsource analytic computations (to a potentially malicious proxy not colluding with the operator). We model our novel building block in the Universal Composability framework and provide a practical protocol instantiation. To demonstrate the flexibility of PUBA, we sketch instantiations of privacy-preserving fraud detection and targeted advertising, although it could be used in many more scenarios, e.g. data analytics for multi-modal transportation systems. We implemented our bookkeeping protocols and an exemplary outsourced analytics computation based on logistic regression using the MP-SPDZ MPC framework. Performance evaluations using a smartphone as user device and more powerful hardware for operator and proxy suggest that PUBA for smaller logbooks can indeed be practical.

Keywords: MPC, Bookkeeping, Building-Block, Analytics, UC

DOI 10.2478/popets-2022-0054
Received 2021-08-31; revised 2021-12-15; accepted 2021-12-16.

***Corresponding Author: Andy Rupp:** University of Luxembourg and KASTEL SRL, E-mail: andy.rupp@uni.lu
Valerie Fetzer, Sven Maier, Markus Raiber, Rebecca Schwerdt: Karlsruhe Institute of Technology, KASTEL, E-mail: {firstname.surname}@kit.edu
Marcel Keller: CSIRO's Data61, E-mail: marcel.keller@data61.csiro.au

1 Introduction

Privacy-enhancing cryptographic protocols could be highly beneficial to citizens and our society in a multitude of user-centric scenarios including mobile payments, loyalty systems, toll collection, web search, participatory sensing, disease prediction, multi-modal transportation systems and pay-as-you-drive insurance. Also, operators of such systems should be encouraged to deploy strong privacy-enhancing technologies in view of EU's GDPR and the severe fines in case of violation.

However, despite their advantages, proposed cryptographic protocols to protect privacy in such scenarios are rarely in real-world use today. The reasons for that are manifold; aside from costs for development and deployment this is caused by a gap between what these protocols offer and what is required when targeting real-world applications: While academic proposals typically strive for "perfect" anonymity and a simple functionality, they neglect scenario-specific real-world requirements stemming from business models, laws, regulations, or user needs. Those requirements often conflict with the desire for perfect anonymity. For instance, fraud detection and anti-money laundering as mandated for digital payments or targeted advertising as performed in loyalty systems are all based on a customer's transaction history. Neglecting requirements of this kind when building privacy-preserving systems, e.g., an anonymous mobile payment or loyalty system, prevents a practical deployment. In this paper, we make an important step towards resolving this issue by proposing a flexible framework that enables the privacy- and authenticity-preserving bookkeeping and analytics of user history data. Our building block is secure in the Universal Composability framework and can thus be securely incorporated into privacy-preserving systems, e.g., anonymous mobile payments, in order to satisfy certain requirements that rely on the analysis of behavioral data of participating users, e.g., fraud detection.

For concreteness, let us consider two examples in more detail: Loyalty systems and mobile payments. In loyalty programs such as Optimum [43] in Canada or Nectar [3] in the UK, customers collect loyalty points for purchases in participating shops which they can re-

deem at a later point to pay for purchases, get vouchers, etc. Academic proposals like Black-Box Accumulators (BBAs) [36] exist, which realize this point collection and redemption functionality in a privacy-preserving way while protecting the operator from forgeries and double-redemption. However, BBA does not take the business model of loyalty system providers into account which typically consist in earning money by enabling advertising partners to place targeted ads and coupons. To this end the operator is keeping track of the users' purchase histories. Ad selection is typically done using a machine learning based classifier which maps purchase histories to product categories customers might find interesting. Advertisers promoting their products send ads for certain product categories to the operator along with the price they are willing to pay every time their ad is delivered by email or displayed on the loyalty program App.

NFC-based mobile payments are an increasingly popular method to conduct payments for goods or services at points-of-sale. This payment mode has been boosted with the recent roll-out of Google Pay and Apple Pay. While systems deployed in practice essentially realize a virtual credit or debit card—thus payments are linkable—privacy-preserving systems could, in theory, be built from anonymous e-cash, e.g., see [4]. However, those constructions do not take current regulations for mobile payments into account, e.g., fraud detection as required by the 2nd European Payment Services Directive (PSD2) [45]. Fraudulent transactions can occur due to lost or stolen payment devices or compromised payment credentials stored on the device. Fraud detection consists in continuously monitoring a customer's transaction history for anomalies or typical fraud patterns. This can be done using a set of rules or sophisticated machine learning algorithms. Based on these background checks it is then decided in realtime whether a newly initiated transaction should be granted or denied.

To realize the two above scenarios—as well as many others—in a privacy-preserving way, a new building-block is needed which ensures privacy-friendly bookkeeping as well as analytics of user history data: A user's actions generate data (e.g., transaction value, date/time, location, type of shop, etc.) observable by an operator. A logbook (solely) kept on the user's device should be updated with the observed data such that its content cannot be manipulated by the user while its current content cannot be learned by the operator. The former is particularly important for requirements like fraud or money laundering detection. More precisely, we need to ensure the authenticity, integrity, freshness, confidentiality, and unlinkability of the logbook. At the same time, the building block needs to allow for analytical computations, where users are incentivized or forced to take part in privacy-preserving analytics of their latest logbook. These computations must not leak the input(s), but the operator may only obtain some privacy-friendly statistics, classification, etc. as output. The user might also receive some output, e.g., targeted ads in case of a loyalty system. Also, in some cases, a slight update of the logbook with results from the analytics computation might be required, e.g., a risk level resulting from the background fraud detection checks. Depending on the scenario, the actual analytics function might need to be kept confidential, e.g., when it is considered intellectual property like the classifier for targeted advertising. In this case, it needs to be ensured that the operator cannot misuse this opportunity to selectively deploy privacy-violating functions.

We distinguish two types of analytics computations: outsourced heavyweight analytics and direct lightweight analytics. As the privacy-preserving evaluation of machine-learning classifiers (e.g., to determine a user's individual fraud risk level) might be computationally and communicationally very demanding, in particular when mobile user devices are involved, there needs to be a possibility to securely outsource this computation to a proxy server who acts on behalf of the user(s). Ideally, even a malicious proxy should not learn anything about user and operator inputs as well as the output provided to the user. In some scenarios, outsourced analytics should not block further interactions with the system, i.e., users should be able to continue updating their logbooks while analytics is running. For instance, in loyalty systems customers should be able to continue shopping and keep track of purchased items while the outsourced ad selection process (for a previous version of the logbook) is still in progress. Direct lightweight privacy-preserving analytics is jointly performed between the user's device and the operator. This type of analytics might be done prior to an update of the logbook to determine further actions, e.g., whether the payment transaction just initiated should be accepted in view of the customer's personal risk level.

1.1 Our Contribution

In this paper, we focus on properly formalizing and instantiating the functionality and security and privacy properties of a bookkeeping and analytics mechanism as sketched in the previous section. Although it is evident that numerous practical application scenarios

would benefit from privacy-preserving bookkeeping and analytics, to the best of our knowledge, this is the *first* work formally studying such a multi-purpose building block.

Our contribution consists in a formal UC-based modeling, protocols for managing the logbook along with security proofs, implementations and benchmarks of crucial protocols, as well as exemplary instantiations of fraud detection and targeted advertising in our framework. This work is one of very few which combines a complex, yet practical crypto system with a thorough UC security analysis. Some details are give below.

Security Model. Based on a high-level system architecture (cf. Section 2) including a set of intuitive but informal security properties (cf. Section 2.5), we designed a flexible ideal functionality (cf. Section 3) in the UC framework [16, 17]. Due to the page limit we only introduce parts of the functionality here, the full functionality can be found in Appendix D. Providing a modeling and a security analysis in the UC framework is essential as we expect PUBA to be used as a building block for larger privacy-preserving systems. A main challenge consisted in finding an appropriate trade-off between flexibility (to cover a broad set of applications) and complexity. Our functionality covers users, an operator, a trusted signing authority (verifying the privacy-friendliness of analytical functions), and proxies who interact in preparational tasks like User Registration and Sign Function Parameter as well as operational tasks such as Bookkeeping (which also covers Direct Analytics), Outsource, Outsourced Analytics, and Update. The correct computation of an application-specific function Δ contained in both Bookkeeping and Outsourced Analytics is managed by a subfunctionality $\mathcal{F}_{\text{PPA}}$ which provides independence of the used method of computation. The subfunctionality can, e.g., be instantiated with any secret-sharing based general MPC framework (for example based on the SPDZ [23] family) that treats Δ as a black-box, which allows for flexible instantiations and lets PUBA easily profit from performance advances in MPC. It is, however, also possible to provide non-black-box instantiations of $\mathcal{F}_{\text{PPA}}$ which exploit the structure of Δ, e.g., by letting the user prove in Zero-Knowledge that it was computed correctly. This allows for more efficient computations, which is especially desirable for the Direct Analytics included in the Bookkeeping task.

Provably Secure Protocols. We designed cryptographic protocols for the tasks mentioned above in Appendix E and provide a full security proof for our construction in Appendix G. Our main challenge was to carefully combine different primitives and techniques into protocols that are both secure in the UC framework and efficient at the same time. To this end, we follow a commit-sign-rerandomize-proof approach to achieve the desired security properties for our logbook. We build on Groth-Sahai [34] non-interactive zero-knowledge proofs, which are fairly practical and secure under standard assumptions (e.g., SXDH over bilinear groups). Furthermore, we use structure-preserving signatures [1] and homomorphic multi-commitments [2] for which statements (e.g., the knowledge of a signature on a commitment) can be efficiently proven using Groth-Sahai.

Implementation and Benchmarks. We implemented our protocols for User Registration, Bookkeeping, Outsource and Update as well as an exemplary Outsourced Analytics computation based on logistic regression using the MP-SPDZ MPC framework [40]. For the user side we used a Nexus 5X and a Galaxy S8 smartphone, whereas the operator and proxy computations were implemented on more powerful server hardware. Our measurements show that even fairly complex Bookkeeping involving permuting, setting, and adding values to entries can be done within about 2 seconds for a logbook containing 100 entries and assuming an NFC data rate of 424 kbit/s. Outsourcing the logbook for an analytics computation takes around 660 ms assuming a mobile data rate of 10 Mbit/s. Retrieving the results of the Outsourced Analytics and (potentially) updating the logbook can be done in 2 seconds assuming the same data rate. The privacy-preserving analytical computation itself takes around 380 ms for colocated servers of operator and proxy. These estimates show that privacy-preserving bookkeeping and analytics could indeed be practical.

Application: Fraud Detection. We sketch how a simple two-tier fraud detection could be realized with PUBA. Tier 1 enforces the user in the scope of an Outsourced Analytics computation to perform a more complex machine learning based fraud detection with the operator if a certain number of payment transactions has been reached. This results in some risk level which is stored in the logbook. Tier 2 is a Direct Analytics computation performed when a new payment transaction is initiated. It consists of checking simple rules taking the risk level into account to decide whether the transaction is accepted or declined. Assuming the hardware and data rates from above and a logbook storing the last 20 transaction records, where each record consists of 5 values, we estimate Tier 1 detection using logistic regression to take around 3 seconds (including Out-

sourcing and Update), where Tier 2 detection including Bookkeeping can be done in less than 2 seconds.

1.2 Related Work

To the best of our knowledge, this work is the first formally defining a multi-purpose privacy-preserving bookkeeping and analytics building block.

Yet, regarding bookkeeping, we partly use similar mechanisms as Black-Box Accumulators (BBA+) introduced in [36, 38]. BBA+ provides a framework for *privacy-preserving point collection and redemption* and does not consider data analytics. Moreover, its security is defined using a game-based approach, which leads to weaker security guarantees.

In [10] updatable anonymous credentials are introduced, which allow to dynamically modify the attributes of a credential by an update function after its creation. However, updatable credentials as defined in [10] do not satisfy our requirements. The input to the update function is solely provided by and only known to the user. Moreover, freshness of credentials and their use for analytical computations is not considered. Like [36, 38], they do not consider security in the UC model.

Kolesnikov et al. [42] present schemes that provide *input authentication* in the following sense: in their model, malicious clients want to perform secure multi-party computations with a semi-honest server and the scheme ensures that clients use the *same input* in multiple different computations. This can be considered as a weak form of bookkeeping with weaker requirements. While [42] ensures that users cannot use arbitrary inputs for the computations, they can be linked throughout interactions. Furthermore, in contrast to [42], our framework also works with malicious servers and provides additional security guarantees for honest users not directly inherited by the MPC.

Privacy-preserving analytics and data mining (without bookkeeping and input authentication) has been well-studied before. The proposed approaches from the literature either construct dedicated protocols exploiting the structure of existing methods (such as neural networks or logistic regression) commonly used for Machine Learning [7, 20], or propose techniques for solving a given analytical problem efficiently [24, 46, 51, 52].

Let us briefly consider the related work regarding the applications of PUBA more thoroughly studied in this paper: *targeted advertising in loyalty system* and *fraud detection in mobile payments*. To the best of our knowledge, privacy-preserving targeted advertising for loyalty programs has not been thoroughly analyzed before. The majority of work on privacy-preserving targeted advertising deals with an Internet scenario where an advertisement network (such as Google) wants to display targeted advertisements on Websites based on the User's prior Web activities. Besides a few exceptions, e.g., [8], security and privacy properties of proposed systems are not formally proven. Concerning ad selection, there are currently two prevalent approaches: client-based and proxy-based selection. In client-based ad selection, e.g., [8, 33, 47], typically a browser plugin keeps track of the user's online behavior and interests and classifies the user in order to determine the most relevant ad. This implies that the ad network's selection algorithm has to be public and the correctness of inputs and computation is not ensured. In proxy-based selection, e.g. [35], a Trusted Third Party (TTP) with secure channels to the user hides user data from the ad network. This implies a dependency on the TTP and the assumption that users send correct data.

To the best of our knowledge, barely any work has been devoted to privacy-preserving fraud detection for payments yet. The most interesting work [19] is performing an exploratory study on the performance of fully homomorphic encryption based classification of finance-related transaction data which is run by a fraud detection service provider. Authenticity and freshness of the data is assumed.

2 High Level Description

In this section, we provide a high-level introduction to the parties and tasks involved in the analytics-framework for PUBA. Note that we provide a further discussion in Appendix A.

2.1 Parties and Roles

PUBA considers four different types of parties:

Users $\mathcal{U}$. PUBA allows for arbitrarily many users to participate in the system. Each user collects data inside a personal logbook λ which they can provide for privacy-preserving computations. The main part of this data is called User History (UH) $\mathcal{UH}$. This part is authenticated and cannot be changed by the user at will, but is only updated through private bookkeeping tasks with the operator. The private data is represented by a *vector* of *slots*, each slot contains a $\mathbb{Z}_p$-element. Users may

be corrupted and maliciously collude with other corrupted parties. We assume that users participate using relatively weak hardware. For many real-world applications the user side of PUBA will probably be realized as a smartphone application.

The Operator $\mathcal{O}$. For any given instance of PUBA there is exactly one operator. This party is the central entity managing the system. The operator has an interest in evaluating analytical functions on the data collected by users. In certain scenarios, the precise details of these functions, however, may be subject to trade secrets and should not be leaked to any other party. We thus assume that the operator can *hide* all sensitive information inside Function Parameters (FPs) *fp* while the publicly known function Δ itself only provides the general structure. In real applications, FPs correspond to transition matrices of neural networks [39] or weights and features for linear regression. Being the operator's input to analytics computations they achieve the same level of privacy as the users' private data. Relying on FPs is without loss of generality [49] regarding the class of computable functions. The operator may be corrupted and maliciously collude with other corrupted parties apart from proxies (which does not affect user privacy).

The Trusted Signing Authority $\mathcal{T}$. Allowing arbitrary private FPs precludes any meaningful level of privacy for the users. To prevent malicious operators from using FPs which trivially undermine the users' privacy, we assume the existence of a Trusted Signing Authority (TSA): Before using FPs for computations, the operator has to let them be certified by the TSA to not violate privacy requirements. We elaborate on the difficulties of this task in Appendix A.1. PUBA enforces that only FPs certified by the TSA can be used during computations on the users' data. The TSA is an external entity and has to be trusted by users as well as the operator. We picture the TSA as a privacy-defending donation-funded NGO (like the Electronic Frontier Foundation) or the Federal Data Protection Officer.

Proxies $\mathcal{P}$. PUBA allows arbitrarily many proxies to participate. Proxies are used for computing analytical functions and to coordinate computations that require data from multiple users. We assume proxies to provide the computational power and bandwidth of a regular server. Proxies may be corrupted and collude with corrupted users. As long as proxies do not collude with the operator, PUBA guarantees all user data to be hidden from proxy and operator. Any user may potentially set up their own proxy, yet it is also possible that NGOs provide donation-funded proxy servers.

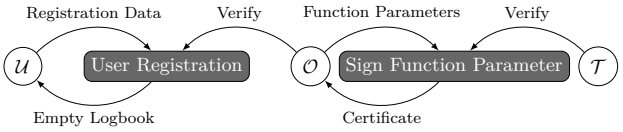

Fig. 1. Overview of the preparatory tasks.

2.2 Preparatory Tasks

Before users can participate in the system they have to register with the operator. Similarly, the operator has to have FPs signed by the TSA before they can use them. Both tasks have to be conducted only once (per user/FP respectively) before computations with them are possible. An overview of these tasks is provided in Fig. 1.

Sign Function Parameter. Any function computed with PUBA is split in two parts: A generic and publicly known function representation Δ (e.g., a neural network with padded neurons) and a corresponding set of FPs which determine the specifics (e.g., the transition matrices). These FPs constitute sensitive data held by the operator. Before the operator can use any FPs for computations, the TSA has to verify that the resulting function does not violate the required privacy standards. To that end, the TSA verifies the FPs according (but not limited) to the following criteria: 1) Is the output sufficiently general to not leak confidential user information? We assume a public catalog of requirements FPs have to fulfill. This also provides feedback for the users on the expected level of privacy. 2) Does this function use the slots of the UH according to their specification? We generally assume that the specification of the UH—the semantic interpretation of the individual slots—to be public knowledge. 3) Are there any additional leaks when combining these FPs with any of the previously certified FPs? We model the verification function such that the TSA can input all previously accepted FPs as auxiliary input. The actual Sign Function Parameter (SFP) task is depicted in Fig. 1: The operator inputs FPs which are then checked by the TSA against the privacy guidelines. If the FPs comply with the guidelines, the TSA provides the operator with some form of certificate. Our protocol uses signatures on the FPs. This certificate is a required input to any computation task that uses these FPs.

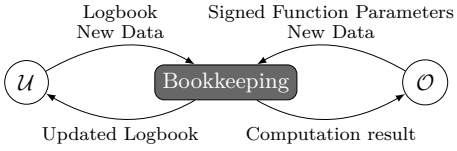

Fig. 2. Overview of the Bookkeeping task.

User Registration. Users store their private User History inside a personal logbook λ. Providing the user with an "empty" logbook which contains an authenticated initial UH is the purpose of the User Registration (UReg) task. The task is depicted in Fig. 1 and consists of three phases. In the first phase the user identifies themselves. This triggers the second phase in which the operator verifies that the user is not registered already—we require that each user can only have *at most* one logbook. Lastly both parties jointly compute the initial logbook containing an authenticated UH. For most scenarios, the UH is supposed to start out empty. For scenarios where the initial UH should contain some specific data, this task allows its computation based on appropriate FPs. The logbook λ is a requirement for participation in any of the other tasks. The UReg task is the *only* task identifying the user and only performed once.

2.3 Bookkeeping

A central focus of PUBA is privacy-preserving bookkeeping, which lets the operator and a user manipulate data with certain guarantees for both parties. An overview of this task can be found in Fig. 2.

Bookkeeping. The Bookkeeping (BK) task manipulates a single user's data. While the main purpose of this task is to provide an efficient targeted manipulation of the UH in an authenticated yet privacy-preserving way (such as adding or reseting individual values), the task optionally performs *lightweight* direct analytics of the user data according to an application-specific function Δ and the certified FPs. This enables manipulations of the UH based on its current values, which may be required for scenarios such as fraud detection where the decision (to be recorded) whether a transaction is granted or not depends on a risk level stored in UH. The main part—authenticated manipulation of the UH—is done without leaking the authenticated and private data stored in the UH to the operator. The UH is updated using Update Information $\mathcal{UI} = (\alpha, \mathbf{s}, \mathbf{a})$. These contain three consecutive operations defined by three maps which are output by Δ during the computation. The first map α defines a *permutation* of the contents inside the UH. The second map $\mathbf{s}$ determines which values of the UH are set to new values directly, we call this *direct update* in the following. The final map $\mathbf{a}$ is defined by a vector of additive updates which will be applied to the values of UH, we call this vector *additive increment*. While the additive increment is hidden from the operator, the permutation and direct update are learned by both parties and hence subject to privacy considerations during auditing with the TSA to ensure that they leak no personal information on the user. Of course, if permutations and direct updates are not needed, the user can skip them in favor of a more efficient updates using only the additive increment. The chosen three-operation update mechanism is a tradeoff between what is usually required for (basic) bookkeeping applications and what can be efficiently implemented (using zero-knowledge proofs and homomorphic commitments).

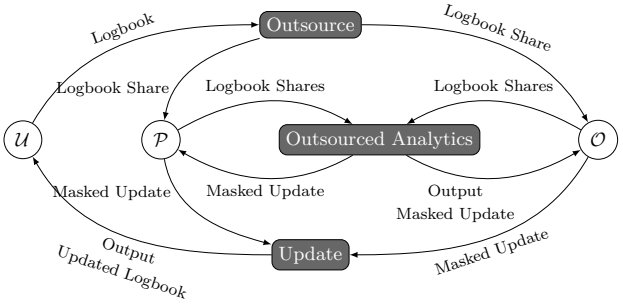

Fig. 3. Overview of tasks for outsourcing computations.

The UH remains authenticated because 1. the operator knows that the input was authenticated, and 2. the correctness properties ensure that the data was manipulated correctly.

2.4 Outsourcing Analytical Computations

We assume users to have relatively weak hardware incapable of securely computing analytical functions like neural networks efficiently. To still enable these costly computations we involve an additional party—the proxy—which provides the computational power and bandwidth required to perform such computations. As long as the proxy does not collude with the operator it learns no secrets or even analytical results from participation. A corrupted proxy also poses no risk for the operator's privacy. Involving a proxy enables *synchro-*

nization of analytical tasks which require private data from several users. The basic workflow for outsourcing analytical tasks is shown in Fig. 3; note that the three different tasks involved in this cannot be scheduled arbitrarily but have to be conducted top to bottom. The general workflow first lets the user *distribute* the logbook containing their current UH to a proxy of its choice and the operator using the Outsource task. These shares are then used by proxy and operator for computing the analytical function using the Outsourced Analytics task. This can potentially cause updates on the UH which follow the same three basic operations (permutation, direct update, additive increment) that were also used for the Bookkeeping task. Again, we want to hide the additive vector applied to the User History and thus use a similar mechanism as during the BK task, only that the values relevant for the user are temporarily stored by the proxy and hence masked with One-Time Pads. The update can later be applied by the user by means of the Update task which also re-enables the Outsource task for that user.

Outsource. To prepare the computation of an analytical task the Outsource (OS) task is conducted between user, proxy and operator as shown in Fig. 3: The user inserts their current logbook which contains the latest authenticated User History. A snapshot of this is shared between the operator and proxy for use in the computation while the user receives a new logbook which is identical to the old one except that it is now marked as having been outsourced.

Outsourced Analytics. Using the shares obtained from the OS tasks, proxy and operator can now conduct the actual computation. This is depicted in Fig. 3 as well. The function is computed on the private user data using any secret-sharing based MPC framework. The operator learns the desired analytical result directly. Information relevant for the user is masked and output to the proxy; the users fetch the data and reconstruct it at their convenience but the proxy does not learn the data. Computation again follows the function Δ and requires validated FPs from the operator.

Update. As mentioned before, the results of Outsourced Analytics can be used to update the user data. To apply these to the latest UH, PUBA provides the Update task (see Fig. 3). This also re-enables the Outsource task for their logbook. The analytical result destined for the user is forwarded to them. Both the operator and proxy input the share they obtained from the Outsourced Analytics task. These shares contain the updates for the UH, the computation results relevant for the user and additional information to detect tampering of the shares. The user inputs the latest UH and obtains a new authenticated UH which was updated using the same mechanism as used in the BK task.

The triplet from Fig. 3 is *non-blocking* regarding further Bookkeeping tasks: A user can outsource the latest User History in an Outsource task, running analytics in the "background", and then update the UH using the Bookkeeping task arbitrarily often *before* participating in the Update task. As mentioned, this is, e.g., desired in scenarios like targeted advertising. We stress, however, that the triplet is blocking with respect to further analytical tasks (amongst others, to hamper denial of service attacks on proxies and operator): After executing the Outsource task the user has to successfully perform the Update task before it can call Outsource again. This implies that the User History input to the Update task has potentially been changed via successive Bookkeeping tasks and significantly differs from the UH used during Outsource. Special care has to be taken during function design to ensure that the update is still meaningful even after any number of Bookkeeping tasks manipulating the UH in the meantime. This can, e.g., be done by defining a separate portion of the UH which can only be modified by Update.

2.5 Security Guarantees

This section discusses the security requirements in an informal fashion based on which we designed an ideal functionality in the UC framework (cf. Section 2.6).

The *operator* expects authenticated inputs and correct analytical results, thereby hiding potential trade secrets. In particular, we desire the following security requirements for the operator:

Owner Binding. Users can only use *their own* User History. Even corrupted users cannot efficiently steal an honest user's UH.

History Freshness. No user can use an outdated User History for any of the Bookkeeping, Outsource or Update task: The same UH can never be used twice without the operator noticing.

History Unforgeability. The User History can only be changed through task executions. It is computationally infeasible for a user to create a User History with arbitrary values that will be accepted by the operator. This hampers *Model Extraction Attacks* [48] in which the user uses targeted inputs to steal the FPs.

Uniqueness. Each user can have *at most* a single logbook. It is not efficiently possible for a user to own two valid logbooks at the same time.

Function Privacy. The Function Parameters input by the operator remain private: only the TSA is allowed to learn them. Other parties only learn the output of the function computed with these FPs.

Any *user* interacting with PUBA expects privacy of the collected data throughout different interactions:

Unlinkability. The leakage is limited to information that enables identifying the user during the User Registration task as well as coupling a consecutive tuple of Outsource, Outsourced Analytics and Update to the same anonymous user. Other than that, leakage is limited so as to not leak any information that enables the operator to link task executions to the same user. We discuss the amount of information leaked in each task in Appendix B.

Input Privacy. The user does not reveal anything about the UH that cannot be derived from the result of the computation.

Function Parameter Binding. Computations can *only* be performed on FPs which were previously certified by the TSA. It is not efficiently possible for the operator to use uncertified FPs.

2.6 The Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$

We introduce the UC functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ for PUBA in Appendix D. For space restrictions we only provide a brief intuition of the functionality in Fig. 4. It is designed such that it fulfills the requirements from Section 2.5. For an introduction to the UC framework, see Appendix C.1.

Our functionality internally manages all the User Histories. This ensures that the only way to change a UH is by interaction through the individual tasks. Inside the functionality the data is linked to the user via their pid which by UC conventions cannot be changed. If a user queries the functionality it fetches the latest UH that belongs to that user and uses it for the desired task. This models *Owner Binding*, *History Freshness* and *History Unforgeability* directly into the functionality.

We use the same idea to ensure the *Function Parameter Binding* property: The state of the functionality contains a list f_{FP} that contains all the FPs that were verified by the TSA to be used for a given task.

Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ (excerpt)

$\mathcal{F}_{PUBA}^{(\Delta)}$ facilitates Privacy-preserving User-data Bookkeeping & Analytics. The function to be computed is specified by the global parameter Δ. It is running with an operator $\mathcal{O}$, a signer $\mathcal{T}$, a polynomial number of proxies $\mathcal{P}$, and polynomial a number of users $\mathcal{U}$.

1. **State**:
   The functionality stores:
   A mapping $f_{\text{FP}}(task, n)$ which maps a tuple $task \in \{\texttt{UReg}, \{BK(k), OA(k)\}_{k=1}^{K}\}$ and $n \in \mathbb{N}$ to Function Parameters *fp* that can be used for the resp. task.
   And for each registered user, the functionality stores:
   - The pid *pid*.
   - The latest User History $\mathcal{UH}$.
   - A flag that is set iff that user is participating in an ongoing Outsourced Analytics task.
   - A (possibly empty) tuple of information for an ongoing Outsourced Analytics task:
     $\underline{pid_{\mathcal{P}}}$**:** the pid of the specified proxy.
     $\underline{k}$**:** specifies the function to be computed by Δ.
     $\underline{ssid}$**:** the Subsession Identifier of the Outsource call.
     $\underline{\mathcal{UH}}$**:** the snapshot of the User History when Outsource was queried.
     $\underline{in_{\mathcal{U}}}$**:** the additional input used by the user.
   - A (possibly empty) tuple containing the results of an Outsourced Analytics task that were not yet fetched during the Update task:
     $\underline{ssid}$**:** the Subsession Identifier of the Outsource call.
     $\underline{\mathcal{UI}}$**:** the update consisting of a permutation α, a direct update $\mathbf{s}$ and an increment $\mathbf{a}$ for the latest UH.
     $\underline{out_{\mathcal{U}}}$**:** the additional output relevant for the user.
2. **Sign Function Parameter**:
   (a) On input $(\texttt{SFP}, in_{\mathcal{T}})$ by $pid_{\mathcal{T}}$, and $(\texttt{SFP}, fp, task, in_{\mathcal{O}})$, abort if $\Delta(\texttt{SFP}, fp, task, in_{\mathcal{O}}, in_{\mathcal{T}}) = 0$ or if there is some ℓ for which $f_{\text{FP}}(task, \ell) = fp$.
   Otherwise, let ℓ be $\min_{\ell}$: $f_{\text{FP}}(task, \ell) = \perp$ be the first free slot of f_{FP}. Set $f_{\text{FP}}(task, \ell) := fp$ and leak $(task, \ell)$ to the adversary. Then output $(\texttt{ok})$ to $\mathcal{O}$ and $\mathcal{T}$.
3. **Bookkeeping**:
   (a) On input $(BK(k), fp, in_{\mathcal{O}})$ by $pid_{\mathcal{O}}$, abort if there is no $\ell \in \mathbb{N}$ for which $fp = f_{\text{FP}}(BK(k), \ell)$.
   Otherwise, leak ℓ to the adversary.
   (b) On input $(BK(k), in_{\mathcal{U}})$ by $pid_{\mathcal{U}}$, once the first operator message has already been handled, load the User History associated with $pid_{\mathcal{U}}$ and abort if $\mathcal{UH} = \perp$.
   Compute $\Delta(BK, fp, k, \mathcal{UH}, in_{\mathcal{U}}, in_{\mathcal{O}})$ to obtain outputs $(\mathcal{UI}, out_{\mathcal{U}}, out_{\mathcal{O}})$. Interpret $\mathcal{UI}$ as $(\alpha, \mathbf{s}, \mathbf{a})$.
   Set $\mathcal{UH}' := \alpha(\mathcal{UH})$, then for each entry $\mathbf{s}[i]$, if $\mathbf{s}[i] = \perp$ then set $\mathcal{UH}''[i] := \mathcal{UH}'[i]$, and otherwise set $\mathcal{UH}''[i] := \mathbf{s}[i]$. Finally, set $\mathcal{UH}^{\text{new}} := \mathcal{UH}'' + \mathbf{a}$ and update the stored for $pid_{\mathcal{U}}$ as $\mathcal{UH}^{\text{new}}$.
   Output $(\alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}})$ to $\mathcal{U}$ and $(\alpha, \mathbf{s}, out_{\mathcal{O}})$ to $\mathcal{O}$.

⋮

Fig. 4. An excerpt of the ideal functionality $\mathcal{F}_{PUBA}^{(\Delta)}$.

For any task where FPs are required the functionality aborts if the input is not in the list f_{FP}, and the list can only be expanded using the Sign Function Parameter task.

The functionality is parameterized by the function-specific Δ which specifies the computations for a given scenario. Its first parameter always specifies the task; UReg is for User Registration, SFP stands for Sign Function Parameter, BK for Bookkeeping, and OA for Outsourced Analytics. As there might be several ($K \in \mathbb{N}$) possible functions for BK and OA we specify by $k \leq K$ the function that is to be computed. The remaining parameters only depend on the first parameter that specifies the task.

We assume that a list of privacy-requirements that FPs have to fulfill for any task in order to be accepted is also contained in Δ. Those are checked in the SFP task by the functionality to ensure that any protocol realizing this functionality allows only FPs that fulfill the given requirements.

Modeling *Input Privacy* and *Function Privacy* is done by defining the functionality such that it does not leak any information that allows the adversary to reconstruct any of the inputs.

Not leaking the user's identity is a more challenging task. In UC the functionality generally reports to the adversary (and hence to the simulator in the ideal world) whenever an input has been received; the leak contains the task description and the pid of the calling party. To ensure *Unlinkability* our functionality does not leak the pid of the user to the adversary[1] (cf. Appendix B) but only that input from *any* user has been received. It is then up to the simulator to report messages correctly without getting the pid directly.

3 Instantiation

This section sketches our instantiation π_{PUBA} of PUBA, see Appendix E for the full version.

3.1 Cryptographic Building Blocks

We make use of the following building blocks. For formal definitions see Appendix C.

An IND-CPA secure symmetric encryption scheme. Parties exchange keys to set up a secure channel before starting a task. The resulting shared secret key is used as symmetric key by both parties to encrypt any message that belongs to this task. The parties encrypt those messages using an IND-CPA secure symmetric encryption scheme. AES [22] can be used for that purpose.

A structure-preserving EUF-CMA-secure signature scheme. To ensure the integrity of the logbook, we use an EUF-CMA-secure signature scheme which is compatible (structure-preserving) with our zero-knowledge proofs. A possible instantiation is [1].

An additively homomorphic, structure-preserving commitment scheme. Each user stores data inside their User History alongside additional information (see Section 3.2) to which it needs to commit. We require the scheme to be homomorphic, unconditionally hiding and computationally binding. These requirements are satisfied by [2].

A Non-Interactive Zero-Knowledge Proof of Knowledge scheme. This scheme is used by the user to prove that certain operations have been performed correctly. It needs to be extractable and zero-knowledge. Groth-Sahai proofs [34] satisfy our requirements.

A Robust Secret Scharing scheme. We use a robust secret sharing scheme, which lets a party create shares of a secret in such a way that (1) the recipients can verify the integrity of the received shares, and (2) tampering with the shares can be detected during reconstruction. Unlike *verifiable* secret sharing which only protects against a malicious dealer, *robust* secret sharing also protects against recipients trying to manipulate their shares in order to change the reconstructed output.

A pairing group. We make use of an asymmetric pairing group $\mathsf{gp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, e, p, g_1, g_2)$. The groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_\mathrm{T}$ of prime order p are cyclic groups with generators g_1 and g_2. Identification of a user relies on the hardness of the Co-CDH problem [12], which asks to compute g_2^x given g_1, g_2, g_1^x. The Co-DH assumption is implied by the SXDH assumption [34] we use to instantiate Groth-Sahai proofs.

3.2 The User Logbook

We denote by λ the logbook containing the data stored by a user:

$$\lambda = \left(\begin{smallmatrix} \mathcal{UH} \\ \mathtt{com}_{\mathcal{UH}} \\ \mathtt{unv}_{\mathcal{UH}} \end{smallmatrix}, \begin{smallmatrix} ser \\ \mathtt{com}_{ser} \\ \mathtt{unv}_{ser} \end{smallmatrix}, \begin{smallmatrix} lin \\ \mathtt{com}_{lin} \\ \mathtt{unv}_{lin} \end{smallmatrix}, \begin{smallmatrix} id \\ \mathtt{com}_{id} \\ \mathtt{unv}_{id} \end{smallmatrix}, \sigma \right) \tag{1}$$

1 The one exception here is User Registration which we consider to be the only identifying task for the user.

The logbook contains all the data required to maintain the User History and to anonymously interact with the operator.

The User History $\mathcal{UH}$. The key component of the logbook is the User History. It is a vector of $\mathbb{Z}_p$ elements that represents the authenticated data collected by the user.

The Serial Number ser. The serial number is a single $\mathbb{Z}_p$ element that uniquely determines a revision of a logbook. It is unique with overwhelming probability in that throughout the lifetime of PUBA there are no two logbooks (neither belonging to different users nor to the same user) that share the same serial number.

The Linking Number lin. We require linkability *inside* the triplet for outsourcing computations: consecutive tasks of Outsource, Outsourced Analytics and Update have to be linked to the same user as otherwise the data outsourced during the Outsource task cannot be used during Outsourced Analytics and the changes of the resulting update cannot be applied to the correct User History during the Update task. We thus use an additional $\mathbb{Z}_p$ element that links these executions for all three parties involved in outsourcing computations, which the user stores inside the logbook. If no data has been outsourced since the latest update then the linking number is 0.

The Identity id. The user has a fixed private identity which is represented as a secret $\mathbb{Z}_p$ element. It is randomly chosen during User Registration and is never directly revealed to anybody. To prove ownership of the logbook, knowledge of *id* has to be proven.

The Commitment Information com, unv *and Signature σ.* To ensure authenticity the operator generally signs all values inside the logbook. However, the signature is not on the values directly—as this would conflict with the users privacy requirement—but on *commitments* thereof. This is why the user not only stores the values inside the logbook but also the commitments that were used by the operator to compute the signature. As those are part of the witness to generate zero-knowledge proofs, the user also stores the corresponding unveil information.

The final value in the logbook is a signature by the operator that ensures integrity of the commitments on the UH, the serial number, the linking number and the identity.

3.3 General Principles

For every task involving the user, our protocol begins and ends with the same two mechanisms: The authenticated input mechanism ensures the user enters a fresh and authenticated logbook into the interaction while the updating mechanism provides the user with a new valid logbook when the task is finished.

Authenticated Input Mechanism. At the start of each task the user owns a valid logbook λ containing the data shown in Eq. (1). To prove validity of the logbook to the operator the user first *rerandomizes* all commitments $\mathtt{com}_*$ to commitments $\widetilde{\mathtt{com}_*}$ by homomorphically adding a commitment to 0. In the second step the user computes a Zero-Knowledge proof Π showing that they know (1) commitments $\mathtt{com}_*$ on the same values as the $\widetilde{\mathtt{com}_*}$ and (2) a signature σ that authenticates the original commitments under the verification key of the operator. The rerandomized commitments $\widetilde{\mathtt{com}_*}$ and proof Π are sent to the operator for validation. The above process is only conducted for values the operator is not supposed to learn, usually ($\mathcal{UH}$, *lin*, *id*). In case the user wants to fetch updates from an outsourced analytical computation or wants to start one and needs to show the logbook contains $lin = 0$, the hidden values are ($\mathcal{UH}$, *id*) only. The serial number *ser* is always revealed at the start of a task and checked by the operator (using a database lookup) to make sure the user does not try to use an outdated logbook for a new task.

Updating Mechanism. At the end of each task the user and operator jointly compute a new valid logbook λ^{new} to be used in the next task. To do so, the operator needs to reliably learn commitments ($\mathtt{com}^{\text{new}}_{\mathcal{UH}}$, $\mathtt{com}^{\text{new}}_{ser}$, $\mathtt{com}^{\text{new}}_{lin}$, $\mathtt{com}^{\text{new}}_{id}$) to all new values ($\mathcal{UH}^{\text{new}}$, lin^{new}, ser^{new}, id^{new}) and sign them for the user without learning the values themselves. We explain how commitments to each of the values are obtained by the operator.

The new UH is calculated in two steps: Permutations and value setting by the user and additive increments by the operator. If there are updates which are not compatible with the homomorphic property of the underlying commitment scheme—i.e., permutations and or data fields which should be set to entirely new values—the user performs those updates on their old UH, providing the operator with a new commitment value $\mathtt{com}'_{\mathcal{UH}}$ to the updated version and a zero-knowledge proof that they have done so correctly. For updates compatible with the commitment scheme—i.e., additive increments—the operator directly learns a commitment $\mathtt{com}_{\mathbf{a}}$ of the additive increment which they can add to

$\mathtt{com}'_{\mathcal{UH}}$ to obtain $\mathtt{com}^{\text{new}}_{\mathcal{UH}}$. In case there are only additive updates, the first step is skipped and the user directly adds $\mathtt{com}_{\mathbf{a}}$ to the commitment $\widetilde{\mathtt{com}_{\mathcal{UH}}}$ they learned at the start of the task.

A new serial number ser^{new} is coin-tossed similar to [11] by the operator and user. The modified protocol works in two rounds: (1) The user picks a random share $ser^{\text{new}}_{(\mathcal{U})}$ and sends a commitment $\mathtt{com}^{(\mathcal{U})}$ of that value to the operator. (2) The operator picks a random share $ser^{\text{new}}_{(\mathcal{O})}$, computes a commitment $\mathtt{com}^{(\mathcal{O})}$ to that value, calculates the commitment $\mathtt{com}^{\text{new}}_{ser} := \mathtt{com}^{(\mathcal{U})} + \mathtt{com}^{(\mathcal{O})}$ to the complete value $ser^{\text{new}} = ser^{\text{new}}_{(\mathcal{U})} + ser^{\text{new}}_{(\mathcal{O})}$ (without knowing $ser^{\text{new}}_{(\mathcal{U})}$) and sends their serial number share alongside the commitment and unveil information to the user.

The linking number *lin* of the logbook stays the same for most interactions. Whenever the linking number is changed (in Outsource and Update) the operator knows the new value lin^{new}, so computing a new commitment $\mathtt{com}^{\text{new}}_{lin}$ to sign for the user is no problem. If $lin^{\text{new}} = lin$, the operator can use the rerandomized commitment $\mathtt{com}^{\text{new}}_{lin} := \widetilde{\mathtt{com}_{lin}}$ they learned at the start of the task.

The ID *id* of course stays the same for one user throughout all tasks and hence the operator can use the rerandomized commitment $\mathtt{com}^{\text{new}}_{id} := \widetilde{\mathtt{com}_{id}}$ as well.

The operator signs all these commitments ($\mathtt{com}^{\text{new}}_{\mathcal{UH}}$, $\mathtt{com}^{\text{new}}_{ser}$, $\mathtt{com}^{\text{new}}_{lin}$, $\mathtt{com}^{\text{new}}_{id}$) to create a signature σ^{new} for the new logbook and sends it to the user. Finally, the user verifies the validity of the new logbook λ^{new}.

3.4 Individual Tasks

Let us explain how the central tasks of Bookkeeping, Outsource, Outsourced Analytics and Update are realized by π_{PUBA}.

Bookkeeping. The Bookkeeping task is almost completely covered by the authenticated input and updating mechanisms explained above. Between the two mechanisms, the user verifies that the operator uses correctly signed Function Parameters (c.p. Section 2.2) for the task and both of them jointly compute the update information and additional outputs according to Δ. We give more information on this MPC computation in Section 3.5.

Outsourcing Analytical Computations. An overview of how π_{PUBA} realizes outsourcing analytical computations can be found in Fig. 5.

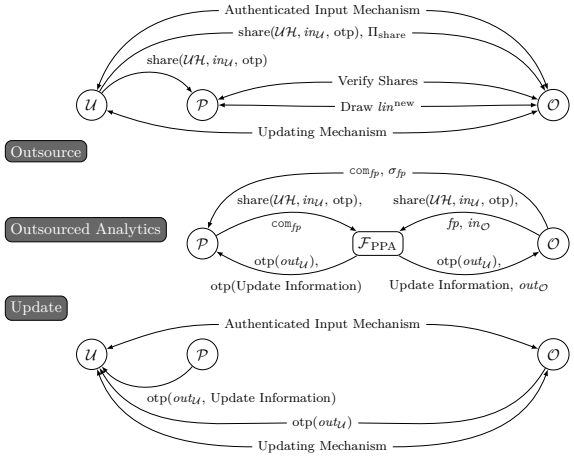

Fig. 5. Simplified depiction of π_{PUBA} for outsourcing computations.

For the Outsource task the user reveals the logbook's linking number *lin* to be zero when proving the validity of their input logbook. Additionally, the user provides additive robust secret shares of their UH $\mathcal{UH}$ and their auxiliary input $in_{\mathcal{U}}$ to the Outsourced Analytics, and of several one-time pads for the proxy and operator, respectively. The one-time pads will be used to let the user collect their update information and computation outputs in a privacy-preserving manner. The secret shares are also proven to be computed correctly. Proxy and operator check the values using the robust secret sharing scheme and, using the values obtained from the proxy in this process, the operator checks the zero-knowledge proof to ensure that the shares were created correctly. Proxy and operator then coin toss a new linking number lin^{new} to be used for this Outsource/Outsourced Analytics/Update triple. Again, as the last step the operator provides the user with the necessary information to obtain a new valid logbook, which only contains a new serial and linking number but the same UH as before.

As Outsourced Analytics does not include the user, the general principles from Section 3.3 do not apply to this task. Instead, the proxy verifies that the operator uses correctly signed FPs *fp* for the computation. To do so the operator sends a commitment $\mathtt{com}_{fp}$ as well as a signature σ_{fp} on the commitment to the proxy who checks that the signature verifies under the key of the TSA. Afterwards both jointly compute Δ via $\mathcal{F}_{\text{PPA}}$. Note that—apart from the update information the operator is supposed to learn—the update information and

outputs for the respective users are one-time padded to retain privacy from both proxy and operator.

In the Update task the user again proves their input logbook to be valid, revealing the linking number *lin* so the proxy and operator can hand out the correct update information and user output from Outsourced Analytics. The user unmasks the one-time padded information from the proxy and operator, respectively, checking them for consistency. The remainder of the task again consists of the user and operator conducting the logbook updating mechanism. The linking number lin^{new} is set to zero again in this process so the user will be able to outsource a new analytical task.

3.5 Wrapping the Computation of Δ

We constructed our protocol in a modular way. While our contribution lies in the construction of a privacy-preserving bookkeeping mechanism built *around* any existing MPC framework, we move the correct usage of the MPC framework itself into an own subfunctionality $\mathcal{F}_{\text{PPA}}$. We only provide a description of the subfunctionality that is sufficient to understand the main body of the paper.

The key reason why we cannot use a general MPC framework directly is because we require extra steps to ensure that both parties input the same data that was previously verified by our protocol. However, those extra steps can be achieved by using any protocol for secret-sharing-based MPC and expanding the computed function accordingly. Moreover, we stress that for a given instantiation of Δ, more efficient instantiations of $\mathcal{F}_{\text{PPA}}$ making non-black-box use of Δ (for example by letting the user prove in zero-knowledge that the function was computed correctly) are also possible.

We use the underlying computation mechanism to extend the following tasks:

User Registration. Here, user and operator use the actual MPC framework for computing the initial UH based on the (unauthenticated) inputs of both parties. Essentially, the added code verifies before the computation that the operator used the same FPs for the MPC that were also verified earlier. After the actual output of Δ has been computed we require additional steps to compute the initial commitment on the UH; from $\mathcal{F}_{\text{PPA}}$, the operator only learns the commitment on the UH and not the actual values.

Bookkeeping. Before starting the actual computation of Δ we again have to verify that the FPs match the ones that were verified before. This is the same trick used during UReg. We use a similar trick to ensure that the user uses the same UH that was verified before. Furthermore, the additive increment of the update should not be learned by the operator, thus, a commitment on this vector is output. The operator then only learns the commitment and the user learns the plain value and unveil information.

Outsourced Analytics. Again, the FPs have to be verified. Our protocol for outsourcing the latest UH is sketched in Fig. 5: During the Outsource task it lets the user create robust secret shares of their User History for the proxy and the operator and *prove* that those are indeed shares which sum up to an authenticated User History. The proxy and operator then only insert their respective shares and the wrapper ensures that 1. the shares are reconstructed correctly, and 2. computation only continues if no tampering is detected. More details can be found in Appendix E.

The function is designed to output the required maps for our three-stage update mechanism: the permutation α, the direct update $\mathbf{s}$, and the additive increment $\mathbf{a}$. Since the operator is again not supposed to learn $\mathbf{a}$, the wrapper adds code that computes an honest commitment and corresponding unveil information. As the user cannot fetch their relevant output directly those are routed through the proxy. To ensure that the proxy does not learn any of the values in the process, we let the user not only create robust shares of the UH but also of five One-Time Pads (OTPs). The first three OTPs hide α, $\mathbf{s}$, and $\mathbf{a}$, respectively. The fourth one hides the decommitment information on $\mathbf{a}$, and the final one is used to mask the function outputs relevant for the user. The additional code hence has to reconstruct and apply these masks to the function output and output all these masked values to the proxy.

3.6 Achieving Security Guarantees

In this section we argue informally why our protocol π_{PUBA} satisfies the security guarantees from Section 2.5 under the assumption that the underlying building blocks from Section 3.1 fulfill their respective properties.

Achieving Owner Binding. Generating a logbook on behalf of another user is infeasible, as this would require showing ownership of its public key which violates the co-CDH assumption. Also, the adversary cannot successfully steal another user's logbook as users never in-

teract with each other, and communication uses confidential channels. Pretending the adversary's logbook belongs to a different user is also prevented: Given the (perfect) extractability of the NIZKPoK this would result in a different witness, which in turn means that (1) the adversary used different unveil information for the commitment, breaking the binding property, or (2) forged a signature on a new commitment, breaking EU-F-CMA security.

Achieving History Freshness. History freshness follows from the online-check of serial numbers. A user trying to re-use an old UH has only three options: (1) Lie during the ZK proof that the rerandomization of $\mathtt{com}_{ser}$ was correctly performed, thus breaking the soundness-property, or (2) open their coin toss commitment $\mathtt{com}^{(\mathcal{U})}$ during the creation of the new serial number to a different value, which would break the binding property, or (3) compute a new signature σ on a changed commitment $\mathtt{com}'_{ser}$ that verifies under the operator's verification key, which would break the unforgeability of the signature scheme.

Achieving History Unforgeability. The logbook entries are only used as witnesses for ZK proofs, but each of them comes with a commitment signed by the operator. Thus history unforgeability intuitively holds for the following reasons: (1) The soundness property of the ZK scheme ensures that proofs containing forged entries will be rejected with overwhelming probability. (2) The binding property of the commitment scheme and the unforgeability of the signature scheme further disable attacks where the commitments on the entries are opened to different values or where the signature on a manipulated entry is forged.

Achieving Uniqueness. During the task for User Registration the operator fetches the public key of the user from $\mathcal{F}_{\text{Reg}}$, which models a Public-Key Infrastructure (PKI) in UC but with the difference that each user can only register *one* key. If the public key the user tries to use during User Registration has already been used the operator aborts. Thus uniqueness follows from the security properties of the PKI.

Function Privacy. Function Privacy during the actual computation follows from the security properties of the MPC framework. In the surrounding protocol the FPs are only ever sent as commitments; the hiding property of the commitment scheme hence ensures that this leaks no information about the actual FPs.

Achieving Unlinkability. We use the Zero-Knowledge property of the NIZKPoK scheme and the hiding and rerandomization property of the commitment scheme: Any data that could be used to link a user to a previous interaction is only used as witness for ZK proofs and the operator only sees commitments thereof. As the commitments are rerandomizable they do not leak information regarding their previous use.

Achieving Input Privacy. Computation of the function Δ inherits input privacy from the security of the functionality $\mathcal{F}_{\text{PPA}}$. In the surrounding protocol users only use UH in three settings: (1) Inside commitments, where the hiding property ensures that this leaks no information. (2) As part of the witness in ZKs proofs, where input privacy follows from the Zero-Knowledge property. (3) For secret sharing during the Outsource task, which is information-theoretically secure as long as the two recipients—the operator and the proxy—do not collude.

Function Parameter Binding. For using FPs which were not verified by the TSA a malicious operator would have to: (1) forge a signature σ_{fp} in the name of the TSA, which would break the unforgeability of the signature scheme, or (2) open a valid commitment $\mathtt{com}_{fp}$ to new (invalid) FPs fp' by breaking the binding property of the commitment scheme.

3.7 Formal Security Statement

As already mentioned in Section 2.6 we prove security of our protocol in the Universal Composability framework by showing indistinguishability against an ideal functionality $\mathcal{F}_{PUBA}^{(\Delta)}$. In this section we elaborate on the achieved security level.

Set-Up Assumptions. We formally conduct our investigation in the asynchronous UC framework with anonymous abort against static corruption of an arbitrary subset of parties that does not include both the proxy *and* the operator at the same time. Following strong impossibility results [18] regarding constructions in the UC-framework, most instantiations require set-up assumptions: Building blocks with a pre-defined behavior that are generally used in a black-box way. This means that those too can be interpreted as functionalities which are controlled entirely by the simulator—providing an advantage over real-world adversaries. The set up assumptions can be instantiated by any protocol realizing this functionality or even by using trusted hardware.

We require the following set-up assumptions:

$\mathcal{F}_{\text{CRS}}$: **Common Reference String (CRS)** A CRS is a string visible to all parties, sampled from a publicly known distribution.

$\mathcal{F}_{\text{auth}}$: **Authenticated Channels** Classical communication in the UC framework is unauthenticated; all communication goes through the adversary, who can read and arbitrarily change the messages. In contrast, authenticated channels enforce integrity of the communication between parties: While messages sent through this channel are still visible to the adversary they can neither be altered by a third party nor be sent in the name of a different party. To ensure user privacy we stress that only the operator, the proxy and the TSA send authenticated messages; users send messages over unauthenticated (hence anonymous) channels.

$\mathcal{F}_{\text{KE}}$: **Key Exchange (KE)** The functionality for secure KE allows two parties to negotiate a *shared secret key* over an open channel. The resulting key is known only to those two parties and can be used to set up confidential channels by using the shared key for symmetric encryption.

$\mathcal{F}_{\text{BB}}$: **Bulletin Board (BB)** The BB is a common abstraction to model a Public-Key Infrastructure (PKI). Parties can register their own public keys and fetch keys from other parties. Unlike real PKIs, BBs allow users to register a key *exactly once* and stored keys do not expire.

Security Theorem. We provide our ideal functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ in Appendix D and a proof of the following theorem in Appendix G:

Theorem 3.1. *Let $\mathcal{Z}$ be any PPT-environment that corrupts a subset of parties that does not contain the operator and the proxy at the same time and that does not contain the TSA, and let the building blocks be instantiated as described in Section 3.1. Then it holds that:*

$$\pi_{PUBA}^{(\mathcal{F}_{\text{CRS}},\mathcal{F}_{\text{auth}},\mathcal{F}_{\text{KE}},\mathcal{F}_{\text{BB}},\mathcal{F}_{\text{PPA}})} \geq_{UC} \mathcal{F}_{PUBA}^{(\Delta)}$$

This means that the protocol π_{PUBA} is at least as secure as $\mathcal{F}_{PUBA}^{(\Delta)}$. While $\mathcal{F}_{PUBA}^{(\Delta)}$ had to be deferred to the Appendix due to the page limit, we stress here that it fulfills all the requirements stated in Section 2.5. The UC proof implies that they maintain their validity *regardless* of which environment they are executed in.

4 Application: Fraud Detection

In the European Central Bank's latest report on card fraud [27] published in 2018, the total value of fraudulent transactions at points-of-sale in 2016 amounted to about 342 Million Euro. This number sounds high, yet it only amounts to 0.008% of the overall card transaction value. This small ratio is achieved mainly thanks to the use of "strong authentication" methods like Chip&Pin as well as fraud detection mechanisms as required by the 2nd European Payment Services Directive (PSD2).

We define a privacy-preserving mobile payment system including fraud detection capabilities. The operator in this system is the bank that offers the mobile payments service to its customers. These are the users in our system who interact using a smartphone App. Fraud detection consists in monitoring a customer's transactions for anomalies or typical fraud patterns. This can be done based on simple rules or sophisticated machine learning algorithms. Due to the real-time requirements, the combination of fraud detection with privacy for mobile payments is particularly challenging. To this end, we consider the following two-tier mechanism: We force the user to perform a more complex machine learning based fraud detection with the operator if some threshold of payment transactions has been reached, resulting in some risk level, and a simple but faster rule-based fraud detection during each payment at a point-of-sale. The latter takes the risk level into account and decides whether the current transaction is accepted or declined.

We assume that a transaction record consists of the following data: a bit indicating whether the transaction has been accepted or declined, a timestamp, the geographic location the transaction took place, the type of shop (e.g., grocery store, jewelry store, etc.), and the transaction value. The User History stores the latest T transactions along with the account's balance and risk level and some values that support the fraud detection mechanisms: a limit on the maximum value a single transaction can have and a value that indicates when the next tier-2-fraud detection mechanism should be executed. In the following we sketch the individual tasks of the system.

Registration. The user registers with the system and gets a new account with an empty balance.

Top-Up. The user can top-up the balance of his account, which is internally realized with a Bookkeeping task. Note that only an addition is needed to update the UH here.

Payment (with simple fraud detection). Before a payment is conducted, it is first checked whether the payment is allowed or the threshold of payment transactions has been reached and a risk calculation has to be performed first. Then, a light-weight fraud detection mechanism is executed (based on the risk level). Depending on the result the payment is either accepted or denied. Internally the Bookkeeping task is used.

Risk Calculation (with complex fraud detection). After a certain number of payment transaction have been executed, a more complex fraud detection algorithm has to be executed. We assume this complex fraud detection mechanism to be based on machine learning (e.g., logistic regression, as suggested by [15, 37]). Since this is a computationally expensive operation, this task is realized with the Outsourced Analytics task. The fraud detection mechanism takes all transactions in the User History as well as the current risk level into account and computes a new risk level.

A more detailed description of this application—including specifics of how the function Δ is instantiated—can be found in Appendix F.1. We estimate the application's performance in Section 5.4.

Table 1. Running times for operator/proxy in ms. $\sum$ denotes the total operator running time, **ZK** the portion of that spent verifying the zero-knowledge proof and $\mathcal{P}$ denotes the running time of the proxy during Outsource. Proxy running time during Update is <1 ms for all User History sizes.

$\lvert\mathcal{UH}\rvert$	Reg.		Bookkeeping		Outsource			Update	
	$\sum$	ZK	$\sum$	ZK	$\sum$	ZK	$\mathcal{P}$	$\sum$	ZK
10	6	3	22	16	20	10	4	20	12
100	6	3	57	51	30	10	15	54	44
200	6	3	96	90	42	10	26	93	80
400	6	3	176	170	60	10	44	172	152
600	7	3	254	248	79	10	65	251	225
800	7	3	332	327	97	10	84	331	299
1000	7	3	411	405	117	10	105	408	370

Table 2. Data exchanged in relation to User History size in kB.

$\lvert\mathcal{UH}\rvert$:	10	100	200	400	600	800	1000
Registration	1.6	1.6	1.6	1.6	1.6	1.6	1.6
Bookkeeping	5.8	11.8	18.4	31.6	44.8	58.0	71.2
(only add)	4.1	4.1	4.1	4.1	4.1	4.1	4.1
Outsource	8.9	34.1	62.0	118.0	174.0	229.9	285.9
Update	7.0	22.6	40.0	74.8	109.6	144.4	170.2

5 Implementation

We evaluated the practicality of PUBA by measuring execution times of a practical implementation. Since network communication depends on various external factors, we omitted communication times and only measured local computation.

5.1 Bookkeeping

Evaluation of the user side is done on a Nexus 5X smartphone released 2015 featuring a Snapdragon 808 with 2×1.8 GHz + 4×1.4 GHz running Android 8.1.0 (Phone 1) and a Galaxy S8 smartphone released 2017 featuring an Exynos 8895 with 4×2.3 GHz + 4×1.7 GHz running Android 9 (Phone 2). We executed the code for operator and proxies on much more powerful servers, equipped with an AMD Ryzen 9 3900X with 12×3.8 GHz. In all cases our implementation makes use of 6 threads to speed up cryptographic operations.

We implemented our protocol in C++17, employing the open-source library RELIC toolkit v0.5.0 [5] for group operations. The required building blocks were instantiated as suggested in Section 3.1: for *signatures* we use the scheme from [1], for *commitments* we implemented [2], and for the NIZKPoK scheme we use the method from [25, 34]. Our building blocks are instantiated over the pairing-friendly Barreto–Naehrig Curves Fp254BNb and Fp254n2BNb [6, 9].

We averaged over 50 executions of each protocol task for User History sizes of 10, 100, 200, 400, 600, 800 and 1000 in order to get representative results. For Bookkeeping we implemented both the three-stage update comprising a permutation of User History, setting values, and adding values, as well as a simplified version where no permutation and direct update are done. The concrete choice of permutation has no impact on performance, whereas performance improves slightly the more entries are set. Thus to provide a good lower bound, we performed a cyclic shift and then set a single entry. Table 1 shows the results using a server for operator and proxy and Table 3 using a smartphone for the user side. When only performing addition during Bookkeeping (and no permutation/setting of entries) it needs to communicate ≈4 kB regardless of the number of entries in the User History, and takes between ≈310 ms and ≈440 ms on Phone 1 and ≈460 ms on Phone 2, while taking 14 ms for the operator.

Overall, even for UH sizes where computation time on a smartphone exeeds 10 s, less than 300 kB of data need to be communicated. Thus, even when using mo-

Table 3. Running time on user devices in ms. $\sum$ denotes the total user running time, **O** denotes the online running time, **ZK** denotes the portion of that spent creating the zero-knowledge proofs, **Val** the time spent validating the new User History (not included in the online time) and **PC** precomputation time (also not included in the online time).

$\lvert\mathcal{UH}\rvert$	Phone	Registration				Bookkeeping				Outsource					Update			
		$\sum$	O	ZK	Val	$\sum$	O	ZK	Val	$\sum$	PC	O	ZK	Val	$\sum$	O	ZK	Val
10	1	154	97	78	57	616	556	494	60	551	189	298	193	64	528	461	400	67
	2	181	99	73	81	969	881	788	87	749	244	409	272	87	809	719	622	90
100	1	155	100	78	56	1559	1501	1432	58	1247	818	369	181	63	1541	1439	1319	107
	2	176	96	71	80	2200	2116	2033	84	1750	1180	477	240	89	2102	2008	1842	92
200	1	223	116	86	84	2725	2616	2542	87	2088	1540	499	204	92	2868	2723	2484	124
	2	175	96	70	79	3465	3382	3300	82	2959	2242	625	235	86	3353	3265	3011	90
400	1	241	122	91	118	5649	5530	5441	119	4309	3373	804	221	128	5633	5502	5015	130
	2	178	96	71	82	6302	6217	6136	85	5603	4508	986	249	88	6422	6331	5902	93
600	1	240	120	88	120	9226	9103	9003	121	7073	5773	1182	252	126	9210	9079	8406	129
	2	178	96	70	84	9695	9612	9519	84	8228	6801	1336	265	88	9978	9885	9269	92
800	1	242	121	90	122	11 899	11 775	11 672	125	9386	7763	1460	253	129	12 100	11 966	11 104	131
	2	186	99	74	87	13 130	13 046	12 950	85	10 778	9020	1656	265	90	13 402	13 311	12 462	93
1000	1	245	121	90	123	14 667	14 539	14 443	129	11 440	9594	1708	256	131	14 940	14 807	13 763	130
	2	189	100	74	92	16 383	16 290	16 213	91	13 447	11 377	1987	269	91	16 642	16 538	15 471	94

bile data, communication times will mostly depend on network latency and in general be relatively short.

5.2 Analytics Computation

We implemented logistic regression inference using MP-SPDZ [40], which allows for benchmarking across a range of security models and protocols. As the cleartext modulus of the used curve is not compatible with the implementation of homomorphic encryption in MP-SPDZ, we restrict ourselves to protocols based on oblivious transfer with malicious security. For this we use MASCOT [41]. As MP-SPDZ already implements logistic regression, we only had to choose the number of features and whether to approximate the sigmoid function for faster computation. For the former, we ran inference for 10, 20, 50, 100, and 1000 features, and for the latter benchmarked both the established sigmoid function and the three-piece approximation [44]. The latter has been found to deliver good results while being much simpler to compute in the context of secure computation. This is because the restrictions to three linear pieces only requires two comparisons and oblivious selections instead of exponentiation and logarithm. To define the computation domain, we used the order of the 254-bit prime field Weierstrass curve. This allows for a smooth integration with our zero-knowledge proof.

Table 4 shows our end-to-end timings when running on AWS c5.9xlarge and m4.large instances. The LAN times refers to the colocated setting. We simulate a WAN setting by adding a 100 ms roundtrip delay and a bandwidth restriction of 10 Mbit/s. We only use one thread and about 300 MB RAM for malicious security.

At the time of writing, the spot price in US East was USD 0.10 and 1.53 for m4.large and c5.9xlarge, respectively. This results in a cost per computation ranging from USD 1.6×10^{-5} to USD 0.0032.

5.3 Discussion

Our results show that, even on weak hardware compared to modern smartphones, for moderately sized User History our protocol runs fast enough for a smooth user experience: A Bookkeeping—which will be executed most frequently—runs in less than 3 s even including typical network latency for User Histories with 100 entries. When Bookkeeping can be performed without the need of permuting or setting User History entries, it runs in well under 1 s and can even support much larger User History sizes. Outsource needs to transmit more data but is also used less frequently, and a large part of the necessary computation is independent of the current User History state (i.e. creating commitments to shares of one-time pads and the random shares for one of the two parties) and can thus be done in the background in advance. This allows Outsource to have very short online running time even for large User History sizes. Update on the other hand takes about the same time as Bookkeeping. Thus, the main limiting factor for the size of the UH is the time needed for Bookkeeping.

Table 4. Time and communication for logistic regression with two parties. "Strong" refers to c5.9xlarge instances, otherwise we use m4.large.

No. features	Precise sigmoid				Approximate sigmoid			
	Strong LAN (s)	LAN (s)	WAN (s)	MB	Strong LAN (s)	LAN (s)	WAN (s)	MB
10	3.60	5.06	94.86	591.74	0.32	0.59	11.61	69.71
20	3.62	5.07	95.12	593.29	0.33	0.60	11.93	71.26
50	3.65	5.11	95.84	597.94	0.35	0.64	12.58	75.91
100	3.68	5.19	96.90	605.68	0.38	0.71	13.53	83.66
1000	4.28	6.47	116.10	745.10	0.97	1.87	32.82	223.07

This can be partially mitigated if the need of permuting/setting values arises only sparingly or if only parts of the UH are affected. Then the whole UH can be split into multiple parts where permuting/setting is done on some parts and a simple additive update is performed on the other parts. Our evaluation also shows that Phone 2 was consistently slower than Phone 1 even though the stronger processor would suggest otherwise. We are not certain about the cause of this. Possible explanations are the introduction of new battery saving mechanisms in Android 9, or the fact that Phone 1 has the stock google version of Android whereas Phone 2 has vendor-specific modifications.

5.4 Performance of Fraud Detection

We evaluated our fraud detection application from Section 4 with a User History containing 100 elements, corresponding to a scenario where the last $T = 19$ transactions are taken into account for analysis (5 entries per transaction plus previous risk level, number of remaining transactions, transaction limit and balance). We consider these numbers to be realistic and discuss them in Appendix A.3.

Since registration only needs to initialize the default risk level, number of remaining transactions and transaction limit, this is independent of the UH size and takes the same time and data volume as the registration task shown in Table 3 (i.e., 160 ms for Phone 1 and 190 ms for Phone 2 combined user plus operator computation time excluding communication time and 1.6 kB data). Similarly, the top-up task does not require any special computation and thus the results for bookkeeping with only addition holds for this task (i.e., 330 ms-450 ms for Phone 1 and ≈490 ms for Phone 2 combined user plus operator computation time excluding communication time and 4.1 kB data).

Fraud detection happens in two stages: A more complex machine learning based risk assessment is performed regularly, which results in a risk score, a limit for individual payments and a limit for the number of transactions before it has to be executed again. During each payment, simpler rules based on the result of the risk assessment are checked. In our implementation we verified the following two rules: 1. the transaction value is less or equal than the maximum allowed transaction amount 2. the number of payment transactions the user can perform before the complex fraud detection mechanism has to be run has not exceeded its limit. Additionally it is verified, that the current balance is sufficient for the transaction. To do so, we use bulletproof rangeproofs [13] to let the user prove that it evaluated the simple rules correctly. These additional proofs take ≈200 ms on Phone 1 and ≈185 ms on Phone 2, and 17 ms for the operator, independent of the UH size. Thus, for 100 entries (corresponding to the last $T = 19$ transactions), this task takes a total of 1.85 s/2.5 s (for Phone 1/2 resp.) on the user side and 80 ms on the operator side (excluding the communication time to transmit 22 kB of data) for 100 entries. Thus, when storing 19 previous transactions, a transaction can be performed in under 3 s even when taking the communication time into account. Increasing the number of previous transactions increases the computation time on average by ≈1.5 s/≈1.6 s for Phone 1/2 resp. and 40 ms for the operator per 20 additional transactions.

For the outsourced risk calculation we used logistic regression on 100 features using approximate sigmoid calculation and active security for the complex fraud detection mechanism. Our results suggest that the whole outsourced risk calculation process can be completed in well under 5 s: ≈0.7 s for outsourcing, ≈0.4 s for the logistic regression and ≈1.3 s for the update, plus communication time between the user and operator/proxy.

Acknowledgements

This work was supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs.

References

[1] M. Abe, J. Groth, K. Haralambiev, and M. Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 649–666. Springer, Heidelberg, Aug. 2011. 10.1007/978-3-642-22792-9_37.

[2] M. Abe, M. Kohlweiss, M. Ohkubo, and M. Tibouchi. Fully structure-preserving signatures and shrinking commitments. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 35–65. Springer, Heidelberg, Apr. 2015. 10.1007/978-3-662-46803-6_2.

[3] Aimia Coalition Loyalty UK Ltd. The Nectar loyalty program. Online Resource, 2020. https://www.nectar.com/.

[4] E. Androulaki and S. M. Bellovin. An anonymous credit card system. In S. Fischer-Hübner, C. Lambrinoudakis, and G. Pernul, editors, *TrustBus 2009*, volume 5695 of *LNCS*, pages 42–51. Springer, Heidelberg, Sept. 2009.

[5] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic, 2020.

[6] D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. C. López-Hernández. Faster explicit formulas for computing pairings over ordinary curves. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 48–68. Springer, Heidelberg, May 2011. 10.1007/978-3-642-20465-4_5.

[7] G. Asharov, S. Halevi, Y. Lindell, and T. Rabin. Privacy-preserving search of similar patients in genomic data. Cryptology ePrint Archive, Report 2017/144, 2017. https://eprint.iacr.org/2017/144.

[8] M. Backes, A. Kate, M. Maffei, and K. Pecina. ObliviAd: Provably secure and practical online behavioral advertising. In *2012 IEEE Symposium on Security and Privacy*, pages 257–271. IEEE Computer Society Press, May 2012. 10.1109/SP.2012.25.

[9] P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 319–331. Springer, Heidelberg, Aug. 2006. 10.1007/11693383_22.

[10] J. Blömer, J. Bobolz, D. Diemert, and F. Eidens. Updatable anonymous credentials and applications to incentive systems. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *ACM CCS 2019*, pages 1671–1685. ACM Press, Nov. 2019. 10.1145/3319535.3354223.

[11] M. Blum. Coin flipping by telephone. In A. Gersho, editor, *CRYPTO'81*, volume ECE Report 82-04, pages 11–15. U.C. Santa Barbara, Dept. of Elec. and Computer Eng., 1981.

[12] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, Sept. 2004. 10.1007/s00145-004-0314-9.

[13] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018. 10.1109/SP.2018.00020.

[14] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In B. S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 410–424. Springer, Heidelberg, Aug. 1997. 10.1007/BFb0052252.

[15] K. Campus. Credit card fraud detection using machine learning models and collating machine learning models. *International Journal of Pure and Applied Mathematics*, 118 (20):825–838, 2018.

[16] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. https://eprint.iacr.org/2000/067.

[17] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001. 10.1109/SFCS.2001.959888.

[18] R. Canetti and M. Fischlin. Universally composable commitments. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, Aug. 2001. 10.1007/3-540-44647-8_2.

[19] R. Canillas, R. Talbi, S. Bouchenak, O. Hasan, L. Brunie, and L. Sarrat. Exploratory study of privacy preserving fraud detection. In *19th International Middleware Conference 2018*, pages 25–31. ACM, 2018. 10.1145/3284028.3284032. URL https://doi.org/10.1145/3284028.3284032.

[20] G. S. Cetin, H. Chen, K. Laine, K. Lauter, P. Rindal, and Y. Xia. Private queries on encrypted genomic data. Cryptology ePrint Archive, Report 2017/207, 2017. https://eprint.iacr.org/2017/207.

[21] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted backdoor attacks on deep learning systems using data poisoning, 2017.

[22] J. Daemen and V. Rijmen. AES and the wide trail design strategy (invited talk). In L. R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 108–109. Springer, Heidelberg, Apr. / May 2002. 10.1007/3-540-46035-7_7.

[23] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, Aug. 2012. 10.1007/978-3-642-32009-5_38.

[24] T. M. Dugan and X. Zou. A survey of secure multiparty computation protocols for privacy preserving genetic tests. In *Proceedings of the First IEEE International Conference on Connected Health: Applications, Systems and Engineering Technologies, CHASE 2016, Washington, DC, USA, June 27-29, 2016*, pages 173–182. IEEE Computer Society, 2016. 10.1109/CHASE.2016.71. URL https://doi.org/10.1109/CHASE.2016.71.

[25] A. Escala and J. Groth. Fine-tuning Groth-Sahai proofs. In H. Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 630–649. Springer, Heidelberg, Mar. 2014.

10.1007/978-3-642-54631-0_36.

[26] European Central Bank (ECB). Average number of cash and card transactions per person per day in the euro area in 2016, by country. Website, 11 2017. https://www.statista.com/statistics/893459/average-number-of-transactions-per-person-per-day-by-method/, last visited 2021-11-18.

[27] European Central Bank (ECB). Fifth report on card fraud, september 2018. Website, 9 2018. https://www.ecb.europa.eu/pub/cardfraud/html/ecb.cardfraudreport201809.en.html, last visited 2020-04-27.

[28] Y. Gao, B. G. Doan, Z. Zhang, S. Ma, J. Zhang, A. Fu, S. Nepal, and H. Kim. Backdoor attacks and countermeasures on deep learning: A comprehensive review, 2020.

[29] O. Goldreich. *The Foundations of Cryptography - Volume 1: Basic Techniques*. Cambridge University Press, 2001. ISBN 0-521-79172-3. 10.1017/CBO9780511546891. URL http://www.wisdom.weizmann.ac.il/%7Eoded/foc-vol1.html.

[30] O. Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004. ISBN 0-521-83084-2. 10.1017/CBO9780511721656. URL http://www.wisdom.weizmann.ac.il/%7Eoded/foc-vol2.html.

[31] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In A. Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. 10.1145/28395.28420.

[32] S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th ACM STOC*, pages 365–377. ACM Press, May 1982. 10.1145/800070.802212.

[33] M. Green, W. Ladd, and I. Miers. A protocol for privately reporting ad impressions at scale. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 1591–1601. ACM Press, Oct. 2016. 10.1145/2976749.2978407.

[34] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In N. P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, Apr. 2008. 10.1007/978-3-540-78967-3_24.

[35] S. Guha, B. Cheng, and P. Francis. Privad: Practical privacy in online advertising. In D. G. Andersen and S. Ratnasamy, editors, *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011, Boston, MA, USA, March 30 - April 1, 2011*. USENIX Association, 2011. URL https://www.usenix.org/conference/nsdi11/privad-practical-privacy-online-advertising.

[36] G. Hartung, M. Hoffmann, M. Nagel, and A. Rupp. BBA+: Improving the security and applicability of privacy-preserving point collection. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 1925–1942. ACM Press, Oct. / Nov. 2017. 10.1145/3133956.3134071.

[37] F. Itoo, Meenakshi, and S. Singh. Comparison and analysis of logistic regression, naïve bayes and knn machine learning algorithms for credit card fraud detection. *International Journal of Information Technology*, pages 1–9, 2020. 10.1007/s41870-020-00430-yj. URL https://doi.org/10.1007/s41870-020-00430-y.

[38] T. Jager and A. Rupp. Black-box accumulation: Collecting incentives in a privacy-preserving way. *PoPETs*, 2016(3): 62–82, July 2016.

[39] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In W. Enck and A. P. Felt, editors, *USENIX Security 2018*, pages 1651–1669. USENIX Association, Aug. 2018.

[40] M. Keller. MP-SPDZ: A versatile framework for multi-party computation. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *ACM CCS 20*, pages 1575–1590. ACM Press, Nov. 2020. 10.1145/3372297.3417872.

[41] M. Keller, E. Orsini, and P. Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 830–842. ACM Press, Oct. 2016. 10.1145/2976749.2978357.

[42] V. Kolesnikov, R. Kumaresan, and A. Shikfa. Efficient verification of input consistency in server-assisted secure function evaluation. In J. Pieprzyk, A.-R. Sadeghi, and M. Manulis, editors, *CANS 12*, volume 7712 of *LNCS*, pages 201–217. Springer, Heidelberg, Dec. 2012. 10.1007/978-3-642-35404-5_16.

[43] Loblaw Companies. PC Optimum loyalty program. Online Resource, 2020. https://www.pcoptimum.ca.

[44] P. Mohassel and P. Rindal. ABY^3: A mixed protocol framework for machine learning. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 35–52. ACM Press, Oct. 2018. 10.1145/3243734.3243760.

[45] E. Parliament and Council. PSD2: 2nd payment services directive (2015/2366), 2015. URL http://data.europa.eu/eli/dir/2015/2366/2015-12-23. https://eur-lex.europa.eu.

[46] A. W. Senior, S. Pankanti, A. Hampapur, L. M. Brown, Y. Tian, A. Ekin, J. H. Connell, C. Shu, and M. Lu. Enabling video privacy through computer vision. *IEEE Secur. Priv.*, 3(3):50–57, 2005. 10.1109/MSP.2005.65. URL https://doi.org/10.1109/MSP.2005.65.

[47] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *NDSS 2010*. The Internet Society, Feb. / Mar. 2010.

[48] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction APIs. In T. Holz and S. Savage, editors, *USENIX Security 2016*, pages 601–618. USENIX Association, Aug. 2016.

[49] L. G. Valiant. Universal circuits (preliminary report). In A. K. Chandra, D. Wotschke, E. P. Friedman, and M. A. Harrison, editors, *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*, pages 196–203, Hershey, Pennsylvania, USA, May 1976. ACM. 10.1145/800113.803649.

[50] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 707–723. IEEE, 2019. 10.1109/SP.2019.00031. URL https://doi.org/10.1109/SP.2019.00031.

[51] T. Winkler and B. Rinner. A systematic approach towards user-centric privacy and security for smart camera networks. In M. Wolf, G. L. Foresti, and H. Bischof, editors, *2010 Fourth ACM/IEEE International Conference*

on Distributed Smart Cameras, Atlanta, GA, USA - August 31 - September 4, 2010, pages 133–141. ACM, 2010. 10.1145/1865987.1866009. URL https://doi.org/10.1145/1865987.1866009.

[52] H. Yu, J. Lim, K. Kim, and S.-B. Lee. Pinto: Enabling video privacy for commodity IoT cameras. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 1089–1101. ACM Press, Oct. 2018. 10.1145/3243734.3243830.

A Discussion: On the Limitations of Our Scheme

A.1 Verification by the TSA

A common problem of functions constructed using deep machine learning (such as neural networks) is a lack of transparency regarding their behavior. Our framework suffers from the same problem which even persists if we ignore function privacy for the operator; a user who has to compute a neural network on private data does not automatically know what the network computes and how the output is to be interpreted. A function that maps, say, purchases of a user to some abstract class of advertisements relevant for that user is hard to distinguish from one that maps purchases to an encoding that reflects the individual purchases on a fine-grained level.

In PUBA the problem is even harder as we additionally require function privacy for the operator; only a Trusted Signing Authority is there to ensure that the operator only uses valid Function Parameters which provide a sufficient level of privacy for the user. The TSA has the same problems mentioned above: While it is straightforward to check whether a given machine learning model indeed classifies as specified for randomly chosen inputs, a sufficiently complex model can be used to hide *backdoors* [21] in the form of special inputs provided by the operator which would break unlinkability and input privacy for any user. As it is highly unlikely that the TSA finds this backdoor by using random testing, the model behaves normally for all inputs chosen by the TSA with high probability and could even get certified. While we generally assume that the output of the function is a discrete set of much smaller size than the input space—as is the case for both applications we propose—we do not restrict PUBA to this behavior; using arbitrary output for the operator requires special attention during the versification step.

While it is possible to implant a backdoor into the model given a sufficiently large output space and a sufficiently complex model we stress that there are several different ways to detect—and even to remove, although at the cost of overall accuracy—a backdoor. A survey on the scenario itself alongside mechanisms to detect and remove a backdoor is given in [28].

The generality of PUBA lets the operator create a model with a backdoor and submit it for the Sign Function Parameter task, yet increasing progress in the field of backdoor detection [28, 50] given only the final model makes it unlikely that these Function Parameters will get a certificate. We hence require the TSA to perform a number of such tests in order for verification mechanism to be sufficiently daunting for an operator that tries to use backdoors.

A.2 The Case of Aborts

Aborts are a common problem in MPC: If a party looses connection during a computation or refuses to answer entirely then the computation can not be finished. This is also modeled into most security frameworks. For example, in the asynchronous UC model we model our protocol in the entire communication is managed by the adversary; parties can ask the adversary to transfer a given message to an other party but the adversary is free to change any part of the message. Using authenticated channels removes the adversaries capability ot *change* the message and secure channels additionally take the adversaries ability to *read* the message. Yet even with these precautions the adversary is still able to *drop* messages at will.

For normal computations an abort only means that the parties do not get any output. But for PUBA this means that the user is at worst left with no valid logbook if the abort happened after the old logbook has been invalidated at the start of a task but before the new one has been created and sent to the user. An additional case to consider is if the abort happens during an Outsourced Analytics task. This leaves the user incapable of ever outsourcing data again as the linking number will never be reset.

Yet we stress that dealing with aborts is straightforward and could easily be incorporated into the protocol, albeit at the cost of a longer functionality and protocol description and a much more complicated security proof. But for completeness reasons we sketch here how the protocol can be secured against aborts.

In total there are four tasks where the user is directly involved and where aborting in between means that there is no logbook that the user can use and one task where an abort implies that the user cannot outsource anymore. The four tasks where the user is directly involved in, namely User Registration, Bookkeeping, Outsource and Update, we have to ensure that the mechanism can not be abused to let a malicious user obtain two different logbooks. Thus we require that the same messages that were sent before the abort will be sent in the next interaction again to ensure that the reconstructed logbook will end up with the same serial number. So essentially the reconstruction only finishes the protocol using the state both parties had during the abort.

The situation only becomes complicated if an abort occurs during an Outsourced Analytics task. Without a reconstruction mechanism the user would be unable to outsource ever again, as resetting the linking number *lin* to **0** is only possible in the Update task which requires a completed Outsourced Analytics task. Yet again we stress that a slight modification suffices to deal with this case:

If an abort occurs during an Outsourced Analytics task then this abort only matters if no output has been provided to the operator as parties generally get notified of the abort.[2] Hence the operator is aware that the computation involving data from a given linking number *lin* has failed. The reconstruction task basically consists of the update task but with all three manipulation vectors corresponding to $\perp$. That is, the permutation α is the identity, the direct update $\mathbf{s}$ is $\perp$ everywhere, and the additive increment is $\mathbf{a} = 0$. This resets the linking number *lin* stored inside the users logbook to **0** and thus enables future Outsource tasks for that user.

Note that this reconstruction mechanism can be used to restore a broken logbook; yet until the reconstruction has been performed the user is essentially locked from any further interactions, with the exception of aborts during Outsourced Analytics where the user can still perform Bookkeeping tasks.

2 If the abort is happening in the real world, then we can assume standard techniques such as timeouts can be used to determine that the message will likely never arrive.

A.3 On the Expressiveness of Our Application Benchmark

We are no domain experts for fraud detection in mobile payments, and real-world parameters and implementation details for use cases like fraud detection are not easily obtainable. The simplified instantiations we benchmark in Section 5.4 should therefore be viewed as educated guess how real-world systems could be parameterized. Some explanation for our parameter choices: On average, we have 0.3-0.8 credit card transactions per day per person in Europe [26]. So 20 transactions per logbook (UH-size of 100) would cover about a month, while 200 transactions per logbook (UH-size of 1000) would cover at least 8 months. Older transactions are also implicitly taken into account by making the new risk level depend on the old risk level (and the new transactions). Moreover, Logistic Regression seems to be a reasonable method for credit card fraud detection (e.g., see [15, 37]), therefore we used that for our analysis. The simple fraud detection mechanism we implemented is most likely simpler that mechanisms used in practice. One could of course extend the simple fraud detection we implemented with additional rules. Some ideas on how the simple fraud detection mechanism could be extended are:

1. Look at the location and time difference between transactions and deny them if it is not physically possible to travel the distance in that amount of time.
2. If the transaction amount is a lot higher than the average transaction amount, the number of remaining transactions until the next calculation gets decreased by more than one.
3. Implement daily limits: a limit on the number of transactions that are allowed per day or a limit on the total transaction value that is allowed per day or both.

It would be nice to study how good our system models fraud detection systems used in practice, but that would be a research line of its own.

B Leakage of the Tasks

We claim in Section 2.5 that we ensure unlinkability: the operator can identify the user in the task for User Registration and link the triplet of consecutive execu-

Table 5. Leakage of each task involving a user.

User Registration	Bookkeeping	Outsource	Outsourced Analytics	Update
– Output $out_{\mathcal{O}}$ of Δ.	–Output $(\alpha, \mathbf{s}, out_{\mathcal{O}})$ of Δ.	– Nothing.	– Output $((\alpha_z, \mathbf{s}_z)_z, out_{\mathcal{O}})$ of the computation.	– Outputs $(\alpha, \mathbf{s})$ of the function Δ.
– pid of the user.	– Index of the used FP.		– Index of the used values from Outsource.	– Subsession Identifier of the Outsource task.
– Index of the used FP.			– Index of the used FP.	

tions of Outsource, Outsourced Analytics and Update to some anonymous user, but other than that any two tasks could have been performed by any two users inside the set of successfully registered users. We stress that in real applications the communication structure could still be used to link the same user to different tasks; yet other than that, the individual task executions leak no information to connect any two tasks to the same user outside the computation results. The leakage is listed in Table 5.

In *User Registration*, the user is de-anonymized. This is required to ensure that no user registers twice. In addition, the operator learns the output $out_{\mathcal{O}}$ of the function Δ alongside the used Function Parameters; this output is privacy preserving due to the requirements of the Sign Function Parameter task.

In *Bookkeeping*, all that a corrupted operator learns are the used Function Parameters and the parts of the output of Δ we consider to be relevant for the operator, namely everything except for the additive increment. Again, ensuring unlinkability comes down to the Sign Function Parameter task.

In *Outsource*, the operator learns no identifying information from the behavior of the task itself but learns (through the tasks name) for which function an Outsourced Analytics task has been scheduled.

In *Outsourced Analytics*, the task execution already leaks the function specifier k through the inputs. As the number of parties required for each function is public knowledge and since it is known in which order the calls are executed, the operator knows which Outsource calls are relevant and used for this task. Thus linking the used data to their corresponding Outsource call is trivially possible. Additionally, the operator once more obtains parts of the output from Δ which we require to be privacy-preserving.

In *Update*, the operator learns the Subsession Identifier of the Outsource call that caused the analytical computation. As this Outsource execution is already linked to an Outsourced Analytics execution, the operator can link the entire triplet to the same user. Other than that, the operator learns no new information as the outputs for that user were already learned during the Outsourced Analytics task execution.

C Preliminaries

C.1 The Universal Composability Framework

The Universal Composability (UC) framework [16, 17] models security through *indistinguishability* between worlds. It is a special case of simulation-based security [31, 32], the key idea of it can be sumarized as follows: Secure Multi-Party Computation (MPC) is relatively easy if we have a single entity that is trusted by all the participants and with whom the parties can communicate over secure and authenticated channels. The behavior of this party is referred to as the Ideal Functionality (IF) which defines how the inputs are used for computations, which outputs exist and which party obtains which output. The IF can be only accessed in a black-box way: each party sends its inputs to the trusted entity and obtains its output, without learning any other information in the process. This also makes the level of privacy apparent as anything that an adversary can learn is modelled explicitly by having the trusted party send this information to the adversary. The IF is easy to understand in general and enables easier analysis of the security guarantees.

As it is unrealistic to assume that such a trusted party exists in the real world we refer to the world in which the computations are performed by the trusted party as the *ideal world*.

The actual protocol is then executed in the *real world* where (mutually distrustful) parties interact with each other according to the protocol description. If the

real world can be shown to be *indistinguishable* to an interaction of all parties with the Ideal Functionality then the protocol inherits all the security guarantees of the IF. This requires a *simulator* that reports all protocol messages from the honest parties, as honest parties in the ideal world only forward their input to the functionality and do not send protocol messages. The simulator reports the messages without knowing the actual input and only learns the leaks provided by the IF. By proving that the view provided by the simulator in the ideal world is *indistinguishable* from the view of a protocol execution in the real world we get the guarantee that any attack on the protocol can also be launched against the IF. In particular, this means that no attacks on the real world exist that leak information which cannot be extracted in the ideal world.

While the standalone-setting requires the simulator to provide the entire transcript as a whole—which means that it can be arbitrarily rewritten as long as the result still looks like a valid protocol execution—the Universal Composability framework extends this scenario by requiring the simulator to provide the protocol messages *in time* and await the environments response. This models the idea that other protocols are running in parallel and hence ensures that security guarantees remain valid in arbitrary environments instead of requiring the protocol to be blocking as in the standalone case.

Each session in the UC-framework has a unique Session Identifier (sid). We additionally assume that each new task is assigned with a unique (inside this session) Subsession Identifier (ssid).

UC provides no restrictions on the scheduling of protocols which results in a rather strong security guarantee. However, this comes at the cost of feasibility. In fact, it was shown in [18] that not even weak (MPC-incomplete) building blocks such as commitments can be proven secure in this framework without set-up assumptions. This is why most constructions in the UC framework rely on set-up assumptions such as the Common Reference String (CRS) which is assumed to be a hybrid functionality that is set up by a trusted party. These can in reality either be instantiated by protocols based on computational hardness assumptions or by hardware assumptions.

C.2 Symmetric Encryption Schemes

Indistinguishability under Chosen Plaintext Attacks. Security for symmetric encryption schemes follows the indistinguishability-pardigm. The Indistinguishability under Chosen Plaintext Attacks (IND-CPA) security of symmetric schemes is defined as follows [30]:

Definition C.1. A symmetric encryption scheme $\mathsf{ENC} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is *IND-CPA-secure* if for every PPT-adversary $\mathcal{A}$ and every $x, y \in \{0,1\}^{poly(\kappa)}$ such that $|x| = |y|$, and where κ is the security parameter, the following equation is negligible in κ:

$$\left| \Pr\left[\mathcal{A}(\mathsf{ENC.Enc}_{\mathsf{k}}(x)) = 1 \;\middle|\; \mathsf{k} \leftarrow \mathsf{ENC.Gen}(1^\kappa) \right] \right.$$
$$\left. - \Pr\left[\mathcal{A}(\mathsf{ENC.Enc}_{\mathsf{k}}(y)) = 1 \;\middle|\; \mathsf{k} \leftarrow \mathsf{ENC.Gen}(1^\kappa) \right] \right|$$

C.3 Commitment Schemes

We provide descriptions for bit-commitment schemes which can be expanded cannonically to string-commitments.

Unconditionally Hiding Commitment Schemes. A commitment scheme is unconditionally hiding if the commitment message contains *absolutely* no information on the committed bit. More formally [29]:

Definition C.2. A bit commitment scheme $\mathsf{COM} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Unv})$ is *unconditionally hiding* if the following condition holds:

$$\{\mathsf{gp} \leftarrow \mathsf{COM.Gen}\colon \mathsf{COM.Com}_{\mathsf{gp}}(0)\} \approx_s$$
$$\{\mathsf{gp} \leftarrow \mathsf{COM.Gen}\colon \mathsf{COM.Com}_{\mathsf{gp}}(1)\}$$

Computationally Binding Commitment Schemes. The binding property implies that a commitment on some bit b can not be efficiently opened to a commitment on $1-b$. More formally [29]:

Definition C.3. A bit commitment scheme $\mathsf{COM} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Unv})$ is computationally binding if for every PPT-adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and security parameter κ it holds that the following equation is negligible in κ:

$$\Pr\left[\begin{array}{l|l} \mathsf{COM.Unv}(\mathtt{com}_b, \mathtt{unv}_b, b) \wedge & b \xleftarrow{r} \{0,1\}, \\ \mathsf{COM.Unv}(\mathtt{com}_b, \mathtt{unv}_{\overline{b}}, \overline{b}) & \mathsf{gp} \leftarrow \mathsf{COM.Gen}, \\ & (\mathtt{com}_b, \mathtt{unv}_b, s) \leftarrow \\ & \quad \mathcal{A}_1(\kappa, b), \\ & \mathtt{unv}_{\overline{b}} \leftarrow \mathcal{A}_2(\kappa, s) \end{array} \right]$$

Additively Homomorphic Commitment Schemes. A bit commitment scheme is *additively homomorphic* if com-

mitments expose a desired form of malleability such that arithmetic operations on the commitment yield a desired effect on the underlying values.

Definition C.4. A bit commitment scheme $\mathsf{COM} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Unv}, \mathsf{CAdd}, \mathsf{DAdd})$ is *additively homomorphic* if the following equation is overwhelming in the security parameter κ:

$$\Pr\left[\begin{array}{c|l} \mathsf{COM.Unv}(\mathtt{com}_{b\oplus b'}, & (b,b') \xleftarrow{r} \{0,1\}, \\ \mathtt{unv}_{b\oplus b'}, b\oplus b') & \mathsf{gp} \leftarrow \mathsf{Gen}, \\ & (\mathtt{com}_b, \mathtt{unv}_b) \leftarrow \\ & \quad \mathsf{COM.Com_{gp}}(b), \\ & (\mathtt{com}_{b'}, \mathtt{unv}_{b'}) \leftarrow \\ & \quad \mathsf{COM.Com_{gp}}(b'), \\ & \mathtt{com}_{b\oplus b'} \leftarrow \\ & \quad \mathsf{COM.CAdd}(\mathtt{com}_b, \mathtt{com}_{b'}), \\ & \mathtt{unv}_{b\oplus b'} \leftarrow \\ & \quad \mathsf{COM.DAdd}(\mathtt{unv}_b, \mathtt{unv}_{b'}) \end{array}\right]$$

Structure Preserving Commitment Schemes. If a commitment scheme is defined over some pairing group then an important property for enabling ZK proofs is that it preserves the structure. This means [2]

Definition C.5. A commitment scheme $\mathsf{COM} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Unv})$ is called structure-preserving with respect to a bilinear group generator g if the following conditions are all satisfied.

1) Common parameter gp consists of a group description gp generated by g and constants $a_{ij} \in \mathbb{Z}_p$.
2) Commitment and unveil messages consist of group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$.
3) Opening algorithm Unv consists only of evaluating membership in $\mathbb{G}_1$ and $\mathbb{G}_2$ and relations described by pairing product equations.

C.4 Signature Schemes

Signature schemes aim to provide authenticity of messages. *Existential Unforgeability under Chosen Message Attacks.* We follow the outline for describing Existential Unforgeability under Chosen Message Attacks (EUF-CMA) security set by [30]:

Definition C.6. A signature scheme $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sgn}, \mathsf{Vfy})$ is *EUF-CMA secure* if for every PPT-adversary $\mathcal{A}$ with access to a signing oracle $\mathsf{O_{SIG}}$ it holds that the following equation is negligible in the security parameter κ:

$$\Pr\left[\begin{array}{c|l} \mathsf{SIG.Vfy}(\sigma^*, \mathsf{vk}, m^*) & (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{SIG.Gen}(1^\kappa), \\ & (\sigma^*, m^*) \leftarrow \mathcal{A}^{\mathsf{O_{SIG_{sk}}}}(\mathsf{vk}, 1^\kappa) \end{array}\right]$$

where σ^* was never returned from $\mathsf{O_{SIG_{sk}}}$.

Structure Preserving Signature Schemes. The definition of structure preserving signature schemes is similar to that for commitments. We again use the definition from [2]:

Definition C.7. A digital signature scheme $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sgn}, \mathsf{Vfy})$ is called structure-preserving with respect to a bilinear group generator g if the following conditions are all satisfied.

1) Common parameter gk consists of a group description gp generated by g and constants $a_{ij} \in \mathbb{Z}_p$.
2) Verification key vk consists of group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$ other than gp.
3) Messages and signatures consist of group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$.
4) Verification algorithm Vfy consists only of evaluating membership in $\mathbb{G}_1$ and $\mathbb{G}_2$ and relations described by pairing product equations.

C.5 Zero-Knowledge Schemes

Let R be a witness relation for some NP-language $L = \{stmt | \exists wit : (stmt, wit) \in R\}$. A Zero-Knowledge Proof of Knowledge (ZKPoK) contains of two parties, a *Prover* P and a *Verifier* V. The prover tries to convince the verifier that for a given statement *stmt* it holds that $stmt \in L$ without leaking any other information. If this requires only a single message sent from P to V we call the scheme Non-Interactive Zero-Knowledge.

For efficiency reasons we work with *group-based* constructions, which generally follow the following definition:

Definition C.8. A Zero-Knowledge scheme $\mathsf{POK} = (\mathsf{GenPoK}, \mathsf{SetupPoK}, \mathsf{Prove}, \mathsf{Vfy})$ is *group*-based if each verifiable relation contains *triplets* $(\mathsf{gp}, stmt, wit)$ for a group parameter gp. Further, the algorithms behave as follows:

GenPoK takes as input a security parameter 1^κ and output public parameters gp.

$\mathsf{SetupPoK}$ takes as input gp and outputs a (public) Common Reference String *crs*.

Prove takes as input the Common Reference String *crs*, the group parameters gp, a statement *stmt* and a witness *wit* with $(\mathsf{gp}, stmt, wit) \in R$, and outputs a proof Π.

Vfy takes as input the Common Reference String *crs*, the group parameter gp, a statement *stmt*, and a proof Π, and outputs 0.

The following definitions are implicitly with respect to group-based NIZKs.

Perfect Completemness. A ZK scheme is perfectly complete if a proof regarding a true relation will *always* be accepted. More formally [34]

Definition C.9. A Zero-Knowledge scheme $\mathsf{POK} = (\mathsf{GenPoK}, \mathsf{SetupPoK}, \mathsf{Prove}, \mathsf{Vfy})$ is *perfectly complete* if for all adversary $\mathcal{A}$ we have

$$\Pr\left[\begin{array}{l|l} (gk, x, wit) \in R \implies & (gk, sk) \leftarrow \mathsf{POK.GenPoK}(1^\kappa), \\ \mathsf{POK.Vfy}(gk, crs, x, \Pi) & crs \leftarrow \mathsf{POK.SetupPoK}(gk, sk), \\ & (x, wit) \leftarrow \mathcal{A}(gk, crs), \\ & \Pi \leftarrow \mathsf{POK.Prove}(gk, crs, x, wit) \end{array}\right] = 1$$

Perfect Soundness. Perfect soundness is completeness in the other direction; it formally states that it should be *impossible* to convince a verifier of a false statement [34].

Definition C.10. A Zero-Knowledge scheme $\mathsf{POK} = (\mathsf{GenPoK}, \mathsf{SetupPoK}, \mathsf{Prove}, \mathsf{Vfy})$ is *perfectly sound* if for all adversaries $\mathcal{A}$ we have

$$\Pr\left[\begin{array}{l|l} x \notin L \implies & (gk, sk) \leftarrow \mathsf{POK.GenPoK}(1^\kappa), \\ \mathsf{POK.Vfy}(gk, crs, x, \Pi) & crs \leftarrow \mathsf{POK.SetupPoK}(gk, sk), \\ & (x, \Pi) \leftarrow \mathcal{A}(gk, crs) \end{array}\right] = 0$$

CRS Indistinguishability. CRS Indistinguishability intuitively means that there is some different way to compute the CRS such that (1) the new CRS can not be distinguished from one that was honestly created, and (2) the new CRS can be constructed with a backdoor. More formally:

Definition C.11. A Zero-Knowledge scheme $\mathsf{POK} = (\mathsf{GenPoK}, \mathsf{SetupEPoK}, \mathsf{SetupSPoK}, \mathsf{Prove}, \mathsf{Vfy}, \mathsf{ExtractWit}, \mathsf{SimProof})$ provides *computationally indistinguishable Common Reference Strings* if for every PPT-adversary $\mathcal{A}$ it holds that the following equation is negligible in the security parameter κ:

$$\left|\Pr\left[\begin{array}{l|l} \mathcal{A}(crs) = 1 & (gk, sk) \leftarrow \mathsf{POK.GenPoK}(1^\kappa), \\ & (crs, td_{\mathsf{ext}}) \leftarrow \\ & \quad \mathsf{POK.SetupEPoK}(gk, sk) \end{array}\right]\right.$$
$$\left.- \Pr\left[\begin{array}{l|l} \mathcal{A}(crs) = 1 & (gk, sk) \leftarrow \mathsf{POK.GenPoK}(1^\kappa), \\ & (crs, td_{\mathsf{sim}}) \leftarrow \\ & \quad \mathsf{POK.SetupSPoK}(gk, sk) \end{array}\right]\right|$$

Perfect F-Extractability. This property implies a (limited) way of extracting information from the witness. While it is not possible to extract the witness directly, its exponentiation can be extracted efficiently.

Definition C.12. A Zero-Knowledge scheme $\mathsf{POK} = (\mathsf{GenPoK}, \mathsf{SetupPoK}, \mathsf{Prove}, \mathsf{Vfy}, \mathsf{ExtractWit})$ is *perfectly F_{gp}-extractable* if POK follows CRS indistinguishability (Definition C.11) and for all adversaries $\mathcal{A}$ we have:

$$\Pr\left[\begin{array}{l|l} b = 1 \implies \exists_{wit} & \mathsf{gp} \leftarrow \mathsf{GenPoK}(1^\kappa), \\ F_{gp}(wit) = wit' & (crs, td_{\mathsf{ext}}) \leftarrow \mathsf{SetupPoK}(\mathsf{gp}), \\ & (stmt, \Pi) \leftarrow \mathcal{A}(crs, td_{\mathsf{ext}}), \\ & wit' \leftarrow \mathsf{ExtractWit}(crs, \\ & \quad td_{\mathsf{ext}}, stmt, \Pi), \\ & b \leftarrow \mathsf{Vfy}(crs, stmt, \Pi) \end{array}\right] = 1$$

Dual-Mode. Dual-Mode Zero-Knowledge generally depends on the Common Reference String *crs*. Such schemes provide two different distributions on how the CRS is created which follow CRS indistinguishability (Definition C.11).

We furthermore require *F_{gp}-extractability* in the mode that uses SetupEPoK to set up the CRS [34].

The second mode that uses SetupSPoK is supposed to provide *zero-knowledge*:

Definition C.13. A Zero-Knowledge scheme $\mathsf{POK} = (\mathsf{GenPoK}, \mathsf{SetupEPoK}, \mathsf{SetupSPoK}, \mathsf{Prove}, \mathsf{Vfy}, \mathsf{ExtractWit}, \mathsf{SimProof})$ provides *Statistical Simulatability* if for every adversary $\mathcal{A}$ it holds that the followinng equation is overwhelming in the security parameter κ:

$$\Pr\left[\begin{array}{l|l} \mathsf{POK.Vfy}(crs, x, \Pi) & (gk, sk) \leftarrow \mathsf{POK.GenPoK}(1^\kappa), \\ & (crs, td_{\mathsf{sim}}) \leftarrow \\ & \quad \mathsf{POK.SetupSPoK}(gk, sk), \\ & (x, wit) \leftarrow \mathcal{A}(gk, crs), \\ & \Pi \leftarrow \mathsf{POK.SetupSPoK}(x, crs, td_{\mathsf{sim}}) \end{array}\right]$$

We now have all the tools required to define a dual-mode Zero-Knowledge scheme.

Definition C.14. A Zero-Knowledge scheme $\mathsf{POK} = (\mathsf{GenPoK}, \mathsf{SetupEPoK}, \mathsf{SetupSPoK}, \mathsf{Prove}, \mathsf{Vfy}, \mathsf{ExtractWit}, \mathsf{SimProof})$ is called *Dual-Mode* Zero-Knowledge scheme if all of the following conditions are fulfilled:

1) POK has *CRS indistinguishability.*
2) POK when set up with $(crs, td_{\mathsf{ext}}) \leftarrow \mathsf{POK.SetupEPoK}(\cdot)$ has *perfect completeness.*
3) POK when set up with $(crs, td_{\mathsf{sim}}) \leftarrow \mathsf{POK.SetupSPoK}(\cdot)$ has *perfect completeness.*
4) POK when set up with $(crs, td_{\mathsf{ext}}) \leftarrow \mathsf{POK.SetupEPoK}(\cdot)$ has *perfect soundness.*
5) POK when set up with $(crs, td_{\mathsf{ext}}) \leftarrow \mathsf{POK.SetupEPoK}(\cdot)$ has *F_{gp}-extractability.*

6) POK when set up with $(crs, td_{\mathsf{sim}}) \leftarrow \mathsf{POK.SetupSPoK}(\cdot)$ has *statistical Zero-Knowledge*.

D The Ideal Functionality

Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$

This functionality facilitates user-centric Privacy-Preserving Analytics. The function to be computed is specified by the global parameter Δ.

State:
The Ideal Functionality stores:

- Set $\mathcal{P}_{\text{User}}$ of registered users.
- f_{UH}: $pid_{\mathcal{U}} \mapsto \mathcal{UH}$
- f_{OA}: $\mathcal{P}_{\text{User}} \to \{\texttt{true}, \texttt{false}\}$
- f_{OI}: $\{pid_{\mathcal{P}}\} \times \{1, \ldots, K\}$ where $(pid_{\mathcal{P}}, k)$ maps to a list $f_{\text{OI}}(pid_{\mathcal{P}}, k)$ of entries $(ssid, pid_{\mathcal{U}}, \mathcal{UH}, in_{\mathcal{U}})$.
- f_{FP}: $\{\text{UReg} \cup \{BK((K))\} \cup \{OA(K)\}\} \times \mathbb{N} \to \{fp\}^*$, where different tasks are mapped to a list of Function Parameters fp.
- Partial mapping f_{UP} on $\mathcal{P}_{\text{User}}$. $pid_{\mathcal{U}} \mapsto (ssid, \alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}})$.

Tasks:

- $\mathcal{F}_{PUBA}^{(\Delta)}$-*Init* (Fig. 7)
- $\mathcal{F}_{PUBA}^{(\Delta)}$-*Sign Function Parameter* (Fig. 8)
- $\mathcal{F}_{PUBA}^{(\Delta)}$-*User Registration* (Fig. 9)
- $\mathcal{F}_{PUBA}^{(\Delta)}$-*Bookkeeping* (Fig. 10)
- $\mathcal{F}_{PUBA}^{(\Delta)}$-*Outsource* (Fig. 11)
- $\mathcal{F}_{PUBA}^{(\Delta)}$-*Outsourced Analytics* (Fig. 12)
- $\mathcal{F}_{PUBA}^{(\Delta)}$-*Update* (Fig. 13)

Fig. 6. The basic functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ for user-centric Privacy-Preserving Analytics. The remaining tasks are described in Figs. 7 to 13.

In this section, we provide the full description of our Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$. We use the standard UC model [17], and assume that the simulator $\mathcal{S}$ is activated by $\mathcal{F}_{PUBA}^{(\Delta)}$, whenever any party provides any input.

Our functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ is defined such that all inputs have the form (Task name, List of secret inputs). The task name uniquely determines the task to be executed.

The notifications $\mathcal{S}$ obtains after $\mathcal{F}_{PUBA}^{(\Delta)}$ obtained input from any party depends on the respective party providing the input: On inputs from $\mathcal{T}$, $\mathcal{O}$, or $\mathcal{P}$, $\mathcal{F}_{PUBA}^{(\Delta)}$ activates $\mathcal{S}$ with input (Task name, pid), where pid is the Party Identifier of $\mathcal{T}$, $\mathcal{O}$ or $\mathcal{P}$, respectively.

For users, however, we ensure unlinkability in all tasks except for User Registration (for which we assume an out-of-band verification of the user's identity). During User Registration, the functionality explicitly leaks $pid_{\mathcal{U}}$ of the calling $\mathcal{U}$. For all other tasks, we model unlinkability by having the functionality only revealing the *role* of a user after a call, and *not* the pid; that is, the simulator only gets notified with (Task name, User) after a user sent input.

That way, with the exception of User Registration, the pid of any user is never revealed to anyone outside of $\mathcal{F}_{PUBA}^{(\Delta)}$. As such, the user can interact anonymously.

For our subtasks, we generally assume that additionally to what we explicitly write as in- and outputs, parties append the Subsession Identifier (ssid) of the current task.

The stateful functionality. Our Ideal Functionality is *stateful*, meaning that after interaction with any party, it updates its state. The state contains of several lists, which we present in Fig. 6. First of all, $\mathcal{F}_{PUBA}^{(\Delta)}$ stores lists $\mathcal{P}_{\text{User}}$ of registered users. This list contains the pids of all users which registered using $\mathcal{F}_{PUBA}^{(\Delta)}$-*User Registration*, and which hence have a valid UH. The UH itself is stored inside the functionality with f_{UH}, which ensures that the only way to change it is by using the provided tasks.

For Outsourced Analytics (OA) we only allow each user to only outsource one computation at a time. To that end, the Ideal Functionality remembers in f_{OA} if a given user—identified via $pid_{\mathcal{U}}$—has an outsourcing-triplet in progress or not. Furthermore, the data the user outsourced is stored in f_{OI}. The result of the outsourced computations are stored in f_{UP}, until they are fetched by a user using $\mathcal{F}_{PUBA}^{(\Delta)}$-*Update* from Fig. 13.

Finally, the functionality stores all certified Function Parameters for a given task *task* in f_{FP}. The only way to update this list is after positive response from the Trusted Signing Authority $\mathcal{T}$, which ensures that the operator can only perform computations of functions which were verified before.

$\mathcal{F}_{PUBA}^{(\Delta)}$-*Init*:

Input $\mathcal{O}$: (init).
Input $\mathcal{T}$: (init)

1. Respond to the other tasks.

Output $\mathcal{O}$: (ok).
Output $\mathcal{T}$: (ok).

Fig. 7. Task $\mathcal{F}_{PUBA}^{(\Delta)}$-*Init* for initializing the functionality.

$\mathcal{F}_{PUBA}^{(\Delta)}$-*Sign Function Parameter*:

Input $\mathcal{O}$: $(\texttt{SFP}, fp, task, in_{\mathcal{O}})$.
Input $\mathcal{T}$: $(\texttt{SFP}, in_{\mathcal{T}})$.

1. Check[a] that $\Delta(\texttt{SFP}, fp, task, in_{\mathcal{O}}, in_{\mathcal{T}}) = 1$.
2. Check[a] that fp have not been registered before for task $task$.
3. Let ℓ be $\min_\ell$: $f_{\text{FP}}(task, \ell) = \perp$.
4. Set $f_{\text{FP}}[task, \ell] = fp$.
5. Leak $(task, \ell)$ to the adversary.

Output $\mathcal{O}$: $(\texttt{ok})$.
Output $\mathcal{T}$: $(\texttt{ok})$.

Fig. 8. Task $\mathcal{F}_{PUBA}^{(\Delta)}$-*Sign Function Parameter* for signing Function Parameters.

$\mathcal{F}_{PUBA}^{(\Delta)}$-*User Registration*:

Input $\mathcal{O}$: $(\texttt{UReg}, fp, in_{\mathcal{O}})$.

1. Let[a] ℓ be the index for which $f_{\text{FP}}(\texttt{UReg}, \ell) = fp$.
2. Leak ℓ to the adversary.

Input $\mathcal{U}$: $(\texttt{UReg}, in_{\mathcal{U}})$.

1. Check[a] $pid_{\mathcal{U}} \notin \mathcal{P}_{\text{User}}$
2. Append $pid_{\mathcal{U}}$ to $\mathcal{P}_{\text{User}}$.
3. $(\mathcal{UH}, out_{\mathcal{U}}, out_{\mathcal{O}}) \leftarrow \Delta(UReg, fp, in_{\mathcal{U}}, in_{\mathcal{O}})$.
4. Append $(pid_{\mathcal{U}} \mapsto \mathcal{UH})$ to f_{UH}.
5. Append $(pid_{\mathcal{U}} \mapsto \texttt{false})$ to f_{OA}.

Output $\mathcal{U}$: $(\mathcal{UH}, out_{\mathcal{U}})$.
Output $\mathcal{O}$: $(out_{\mathcal{O}})$.

a If this fails, output $\perp$ and abort.

Fig. 9. Task $\mathcal{F}_{PUBA}^{(\Delta)}$-*User Registration* for registering a user.

The Init-task. The initializing task from Fig. 7 has to be called before anything else. The task only contains the operator and the TSA and essentially starts the whole process. Before calling init, all calls are ignored, after that, the functionality responds to calls of the forms given in Figs. 8 to 13.

Sign Function Parameters. With the task shown in Fig. 8, the operator can input Function Parameters fp to be used for a given task $task$. We assume that the application-specific function Δ contains some mechanism to verify that a given input fp is suitable for this task. If the function verifies fp for usage in $task$, then $\mathcal{F}_{PUBA}^{(\Delta)}$ adds these fp to $f_{\text{FP}}(task)$, which enables its future use. The functionality leaks the new amount of FPs it has for the task $task$ to the adversary; we require this information for our simulation.

User Registration. With the User Registration task from Fig. 9, a user can register for participation. This

$\{\mathcal{F}_{PUBA}^{(\Delta)}$-*Bookkeeping*$^{(k)}\}_{k=1}^{K}$:

Input $\mathcal{O}$: $(BK(k), fp, in_{\mathcal{O}})$.

1. Let[a] ℓ be $\min_\ell : f_{\text{FP}}(BK(k), \ell) = fp$.
2. Leak ℓ to the adversary.

Input $\mathcal{U}$: $(BK(k), in_{\mathcal{U}})$.

1. Set[a] $\mathcal{UH} \coloneqq f_{\text{UH}}(pid_{\mathcal{U}})$.
2. $(\alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}}, out_{\mathcal{O}}) \leftarrow \Delta(BK, fp, k, \mathcal{UH}, in_{\mathcal{U}}, in_{\mathcal{O}})$.
3. $\mathcal{UH}' \leftarrow \alpha(\mathcal{UH})$.
4. $\mathcal{UH}'' \coloneqq (uh''_0, \ldots, uh''_{m-1})$ with $uh''_i \coloneqq \begin{cases} \mathbf{s}[i] & \text{for } \mathbf{s}[i] \neq \perp \\ \mathcal{UH}'[i] & \text{for } \mathbf{s}[i] = \perp \end{cases}$.
5. $f_{\text{UH}}(pid_{\mathcal{U}}) \coloneqq \mathcal{UH}'' + \mathbf{a}$.
6. Set $f_{\text{UH}}(pid_{\mathcal{U}}) \coloneqq \mathcal{UH}^{\text{new}}$.

Output $\mathcal{U}$: $(\alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}})$.
Output $\mathcal{O}$: $(out_{\mathcal{O}}, \alpha, \mathbf{s})$

Fig. 10. Task $\mathcal{F}_{PUBA}^{(\Delta)}$-*Bookkeeping*$^{(k)}$ for direct computation between user and operator of the task defined by $k \in \{1, \ldots, K\}$.

task has to be executed before any further interaction with the user. This is ensured in the ideal world by giving the user their User History (UH) and storing it in f_{UH}. For any future interaction, the users current UH is always fetched from f_{UH}. This does not only ensure that a user cannot manipulate data on its own, it also enforces that any user wanting to perform a computation has undergone the registration process.

Specifics of the function that computes the initial UH are input as fp, a unique identifier—which does not leak any of the values in fp—is leaked to the adversary.

Bookkeeping. The update of the latest User History (possibly based on the data so far collected) by a validly registered user is shown in Fig. 10. The functionality fetches the current UH from f_{UH}, which ensures that the latest UH is used. The operator inputs FPs fp, which define the function to be computed; the identifier of which is leaked to the adversary. We generally assume that computations only leak the basic structure, such as a neural network or logistic regression, to the user and hide the trade secrets in fp. To ensure that only valid fp are used the functionality only continues if $fp \in f_{\text{FP}}(BK(k))$, and aborts if the operator wants to use uncertified input.

The computation of function k is defined in the application-specific function Δ. It yields a result $(\alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}}, out_{\mathcal{O}})$. $(\alpha, \mathbf{s}, \mathbf{a})$ is the part of the result that is used for the later update of the User History. $out_{\mathcal{U}}$ and $out_{\mathcal{O}}$ are respective analytical outputs for the user and the operator. For our security proof we further require that the outputs of corrupted users are leaked to the ad-

versary. Yet we stress that this is not in conflict with our confidentiality guarantees as we do not enforce privacy for corrupted users.

Afterwards, the User History gets updated through "permute", "set" and "add" operations. The variable α contains a permutation, which is applied to the User History to create a temporary User History as

$$\mathcal{UH}' \leftarrow \alpha(\mathcal{UH}).$$

This permutation changes the slots of some (or all) entries, while not changing the contents itself. For example, $\mathcal{UH} := (9, 8, 7)$ and $\alpha := \left(\begin{smallmatrix} 0 & 1 & 2 \\ 2 & 1 & 0 \end{smallmatrix}\right)^3$ would yield $\mathcal{UH}' := (7, 8, 9)$. After the permutation, a set operation is executed. The set vector $\mathbf{s}$ has length m and contains either $\mathbb{Z}_p$ elements or $\perp$. A new temporary User History is then created via

$$\mathcal{UH}'' := (uh''_0, \ldots, uh''_{m-1})$$
$$\text{with } uh''_i := \begin{cases} \mathbf{s}[i] & \text{for } \mathbf{s}[i] \neq \perp \\ \mathcal{UH}'[i] & \text{for } \mathbf{s}[i] = \perp \end{cases}.$$

To continue our small example, $\mathcal{UH}' := (7, 8, 9)$ and $\mathbf{s} := (1, 3, \perp)$ would yield $\mathcal{UH}'' := (1, 3, 9)$. Last, an add operation is performed by simply adding the vector $\mathbf{a}$ (which has also length m) to $\mathcal{UH}''$ to create the final new User History:

$$\mathcal{UH}^{\text{new}} \leftarrow \mathcal{UH}'' + \mathbf{a}.$$

Finishing the example, $\mathcal{UH}'' := (1, 3, 9)$ and $\mathbf{a} := (4, 0, 0)$ would yield $\mathcal{UH}^{\text{new}} := (5, 3, 9)$. This 3-step-process of updating the User History enables flexible updates, yet defines some rules for the update process so that later in the protocol (see Appendix E) the user can efficiently prove that he correctly updated the User History.

Outsource. The outsource-task is described in Fig. 11. It is a three-party task, consisting of the user $\mathcal{U}$ providing the data, the operator $\mathcal{O}$ providing the framework, and a proxy $\mathcal{P}$ providing computational power.

Outsourced tasks are intended to compute functions that either require more than one user, or are too complex to be efficiently computed on the users device.

In it, the functionality verifies that the user has no previous outsourced computation in progress, and remembers that one was started now. It stores the current UH alongside the remaining information for later use, and marks it with the pid of the proxy that should

$\{\mathcal{F}^{(\Delta)}_{PUBA}\text{-}Outsource^{(k)}\}^K_{k=1}$:

Input $\mathcal{U}$: $(OS(k), in_{\mathcal{U}})$.
Input $\mathcal{P}$: $(OS(k))$.
Input $\mathcal{O}$: $(OS(k))$.

1. Load current Subsession Identifier *ssid*.
2. If $\mathcal{U}$ and $\mathcal{O}$ are corrupted, append $(ssid, \perp, \perp, \perp)$ to $f_{\text{OI}}(pid_{\mathcal{P}}, k)$ and skip all following steps.
3. Check[a] $f_{\text{OA}}(pid_{\mathcal{U}}) \stackrel{?}{=} \texttt{false}$.
4. Set $f_{\text{OA}}(pid_{\mathcal{U}}) := \texttt{true}$.
5. Set[a] $\mathcal{UH} := f_{\text{UH}}(pid_{\mathcal{U}})$.
6. Append $(ssid, pid_{\mathcal{U}}, \mathcal{UH}, in_{\mathcal{U}})$ to $f_{\text{OI}}(pid_{\mathcal{P}}, k)$.

Output $\mathcal{U}$: (ok).
Output $\mathcal{P}$: (ok).
Output $\mathcal{O}$: (ok).

Fig. 11. Task $\mathcal{F}^{(\Delta)}_{PUBA}$-*Outsource*$^{(k)}$ for outsourcing user data required for outsourced computation of the task defined by $k \in \{1, \ldots, K\}$.

be used for the computation. To allow linkability with the next Outsourced Analytics and Update tasks, the functionality also stores the ssid of this task.

Outsourced Analytics. The actual analyical computation between operator and proxy is shown in Fig. 12. In any real protocol this task should only be possible if there was a previous call to Outsource. The functionality ensures this by fetching the values from its state, namely by taking the inputs of the users from $f_{\text{OI}}(pid_{\mathcal{P}}, \cdot)$.

Similar to a normal computation the proxy only learns a very basic structure, whereas the trade secrets involved in creating the function are hidden as input *fp* from $\mathcal{O}$ and only used if they were previously certified by $\mathcal{T}$ and hence stored in f_{FP}.

Despite the computation of k according to the application-specific function Δ and a subsequent storing of the results in f_{UP} for later use in $\mathcal{F}^{(\Delta)}_{PUBA}$-*Update* the functionality contains leaks in case of corruptions. For one, it is not efficiently possible for a simulator to extract the whole auxiliary input $in_{\mathcal{U}}$ during $\mathcal{F}^{(\Delta)}_{PUBA}$-*Outsource* so the functionality asks for new inputs for all corrupted users. Furthermore, we stress that a corrupted operator can create arbitrary users which means that for real protocols, it might be the case that some users who called $\mathcal{F}^{(\Delta)}_{PUBA}$-*Outsource* never registered. For those, we also input the data a bit later.

Update. While the premise of our framework is to (1) ensure that any update to the UH is applied by the user, and (2) reward users who gave their data for computation tasks, whereas relevant information was out-

3 Note that since our User History slots are numbered from 0 to $m - 1$, we also define the permutation that way.

$\{\mathcal{F}_{PUBA}^{(\Delta)}\text{-}Outsourced\ Analytics^{(k)}\}_{k=1}^{K}$:

Input $\mathcal{O}$: $(OA(k), fp, in_{\mathcal{O}})$.

1. Let[a] ℓ be $\min_\ell : f_{\text{FP}}(OA(k), \ell) = fp$.
2. Leak ℓ to the adversary.

Input $\mathcal{P}$: $(OA(k))$.

1. Load[a] and remove the first Z_k entries $\left\{\left(ssid_z, pid_z, \mathcal{UH}_z, in_z\right)\right\}_{z=1}^{Z_k}$ from $f_{\text{OI}}(pid_{\mathcal{P}}, k)$.
2. If $\mathcal{P}$ is corrupted, ask for updated inputs $\left\{in_z \,|\, 1 \le z \le Z_k, pid_z \in \mathcal{P}_{\text{corr}}\right\}$.
3. If $\mathcal{O}$ is corrupted, ask for updated information $(\mathcal{UH}_z, in_z)$ for each $z \in \{1, \ldots, Z_k\}$ with $pid_z \notin \mathcal{P}_{\text{User}}$.
4. $(\{(\alpha_z, \mathbf{s}_z, \mathbf{a}_z, out_z)\}_{z=1}^{Z_k}, out_{\mathcal{O}}) \leftarrow \Delta(OA, fp, k, \{(\mathcal{UH}_z, in_z)\}_{z=1}^{Z}, in_{\mathcal{O}})$.
5. Leak $\{(z, \alpha_z, \mathbf{s}_z, \mathbf{a}_z, out_z) \,|\, 1 \le z \le Z_k, pid_z \in \mathcal{P}_{\text{corr}}\}$ to the adversary.
6. If $\mathcal{O}$ is corrupted, leak $\{(z, \alpha_z, \mathbf{s}_z, \mathbf{a}_z, out_z) \,|\, 1 \le z \le Z_k, pid_z \notin \mathcal{P}_{\text{User}}\}$ to the adversary.
7. For every z from 1 to Z_k with $pid_z \in \mathcal{P}_{\text{User}}$ set $f_{\text{UP}}(pid_z) := (ssid_z, \alpha_z, \mathbf{s}_z, \mathbf{a}_z, out_z)$.

Output $\mathcal{P}$: (ok).
Output $\mathcal{O}$: $(\{\alpha_z, \mathbf{s}_z\}_{z=1}^{Z_k} out_{\mathcal{O}})$.

Fig. 12. Task $\mathcal{F}_{PUBA}^{(\Delta)}$-*Outsourced Analytics*$^{(k)}$ for outsourced computation between the operator and the proxy of the task defined by $k \in \{1, \ldots, K\}$.

$\mathcal{F}_{PUBA}^{(\Delta)}$-*Update*:

Input $\mathcal{U}$: (Upd).
Input $\mathcal{P}$: (Upd).
Input $\mathcal{O}$: (Upd).

1. If $\mathcal{U}$ and $\mathcal{O}$ are corrupted, set $(\mathcal{UH}, out_{\mathcal{U}}) = (\perp, \perp)$ and skip all following steps.
2. Set[a] $\mathcal{UH} := f_{\text{UH}}(pid_{\mathcal{U}})$.
3. Set[a] $(ssid, \alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}}) := f_{\text{UP}}(pid_{\mathcal{U}})$.
4. If $\mathcal{O}$ is corrupted, leak $ssid$ to the adversary.
5. If $\mathcal{U}$ is honest and $\mathcal{P}$ is corrupted, leak $ssid$ to the adversary.
6. Remove $pid_{\mathcal{U}} \mapsto (ssid, \alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}})$ from f_{UP}.
7. $\mathcal{UH}' \leftarrow \alpha(\mathcal{UH})$.
8. $\mathcal{UH}'' := (uh_0'', \ldots, uh_{m-1}'')$ with $uh_i'' := \begin{cases} \mathbf{s}[i] & \text{for } \mathbf{s}[i] \neq \perp \\ \mathcal{UH}'[i] & \text{for } \mathbf{s}[i] = \perp \end{cases}$.
9. Set $f_{\text{UH}}(pid_{\mathcal{U}}) := \mathcal{UH}'' + \mathbf{a}$.
10. Set $f_{\text{OA}}(pid_{\mathcal{U}}) := \texttt{false}$.

Output $\mathcal{U}$: $(\alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}})$.
Output $\mathcal{P}$: (ok).
Output $\mathcal{O}$: $(\alpha, \mathbf{s})$.

Fig. 13. Task $\mathcal{F}_{PUBA}^{(\Delta)}$-*Update* for letting the user update its User History and obtain the incentive based on the results of the outsourced computation.

put during $\mathcal{F}_{PUBA}^{(\Delta)}$-*Outsourced Analytics*, the final task of our functionality lets the user fetch the results from $\mathcal{F}_{PUBA}^{(\Delta)}$-*Outsourced Analytics* and obtain the incentive. Fig. 13 shows this step. Ignoring the leakage for now, the functionality fetches the latest UH (to model the fact that after Outsource, the user can perform further Bookkeeping tasks) and applies the three-stage update that was defined by the output from OA onto it. Furthermore, it marks the user as being allowed to outsource a computation again.

E Realization of the Functionality

E.1 Prerequisites

Our protocol requires certain pre- and post-processing steps, before the actual MPC-protocol can be executed. We model those additional steps, namely how the in- and outputs are handled, by an additional *subfunctionality* $\mathcal{F}_{\text{PPA}}$.

The stateless subfunctionality from Fig. 14 is required for executing the protocol for User Registration, Bookkeeping, and Outsourced Analytics.

Functionality $\mathcal{F}_{\text{PPA}}$

This functionality performs the computation specified by Δ in a way that provides *input consistency*. It is parameterized by the to-be-computed function Δ, a *homomorphic commitment scheme* COM which is *unconditionally hiding* and *computationally binding* and uses the subprotocol π_{Share} with the same commitment scheme COM, and a EUF-CMA-secure signature scheme SIG′. The following tasks are provided by $\mathcal{F}_{\text{PPA}}$:

- User Registration (Fig. 15):
  Handles the initial values of the user history during user registration.
- Bookkeeping (Fig. 16):
  Define different possible Bookkeepings between $\mathcal{U}$ and $\mathcal{O}$.
- Outsourced Analytics (Fig. 17):
  Define different possible Outsourced Analytics between $\mathcal{P}$ and $\mathcal{O}$.

Fig. 14. The functionality $\mathcal{F}_{\text{PPA}}$ we use to perform the computations.

User Registration. During User Registration shown in Fig. 15, $\mathcal{F}_{\text{PPA}}$ handles the function-specific part of defining how the initial User History $\mathcal{UH}$ looks like; there, $\mathcal{F}_{\text{PPA}}$ essentially is a wrapper for the actual function Δ, which additionally checks integrity of the used Function Parameters before using Δ and computes the commitment aftter its termination.

Input $\mathcal{U}$: $(\text{UReg}, \mathtt{com}_{fp}, in_{\mathcal{U}})$
Input $\mathcal{O}$: $(\text{UReg}, fp, \mathtt{unv}_{fp}, in_{\mathcal{O}})$

1. Check[a] $\mathsf{COM'.Unv}(\mathtt{com}_{fp}, \mathtt{unv}_{fp}, fp) = 1$.
2. Compute $(\mathcal{UH}, out_{\mathcal{U}}, out_{\mathcal{O}}) \leftarrow \Delta(\mathit{UReg}, fp, in_{\mathcal{U}}, in_{\mathcal{O}})$.
3. $(\mathtt{com}_{\mathcal{UH}}, \mathtt{unv}_{\mathcal{UH}}) \leftarrow \mathsf{COM.Com}(\mathcal{UH})$

Output $\mathcal{U}$: $\mathcal{UH}, \mathtt{unv}_{\mathcal{UH}}, out_{\mathcal{U}}$.
Output $\mathcal{O}$: $\mathtt{com}_{\mathcal{UH}}, out_{\mathcal{O}}$.

Fig. 15. The subfunctionality-task of $\mathcal{F}_{\text{PPA}}$ for performing user registration.

Bookkeeping. The case for the Bookkeeping is shown in Fig. 16. $\mathcal{F}_{\text{PPA}}$ verifies that both sides used the correct values as input; verification of the user inputs lets the user input the User History $\mathcal{UH}$ and unveil information $\mathtt{unv}_{\mathcal{UH}}$, which opens a commitment $\mathtt{com}_{\mathcal{UH}}$ input by $\mathcal{O}$. Likewise, the operator inputs Function Parameters fp and unveil information $\mathtt{unv}_{fp}$, which open the commitment $\mathtt{com}_{fp}$ input by the user. The check whether the given pairs of commitment and unveil messages for the given clear values are performed via MPC, hence the respective inputs of one party remain hidden from the respective other party. $\mathcal{F}_{\text{PPA}}$ only continues with computing the actual function Δ on the given inputs, if the commitments successfully opened to the specified values.

Input $\mathcal{U}$: $(BK(k), \mathcal{UH}, \mathtt{unv}_{\mathcal{UH}}, \mathtt{com}_{fp}, in_{\mathcal{U}})$
Input $\mathcal{O}$: $(BK(k), \mathtt{com}_{\mathcal{UH}}, fp, \mathtt{unv}_{fp}, in_{\mathcal{O}})$

1. Check[a] $\mathsf{COM.Unv}(\mathtt{com}_{\mathcal{UH}}, \mathtt{unv}_{\mathcal{UH}}, \mathcal{UH}) \stackrel{?}{=} 1$.
2. Check[a] $\mathsf{COM'.Unv}(\mathtt{com}_{fp}, \mathtt{unv}_{fp}, fp) = 1$.
3. Compute $(\alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}}, out_{\mathcal{O}}) \leftarrow \Delta(BK, fp, k, \mathcal{UH}, in_{\mathcal{U}}, in_{\mathcal{O}})$.
4. $(\mathtt{com}_{\mathbf{a}}, \mathtt{unv}_{\mathbf{a}}) \leftarrow \mathsf{COM.Com}(\mathbf{a})$.

Output $\mathcal{U}$: $\alpha, \mathbf{s}, \mathbf{a}, \mathtt{com}_{\mathbf{a}}, \mathtt{unv}_{\mathbf{a}}, out_{\mathcal{U}}$.
Output $\mathcal{O}$: $\alpha, \mathbf{s}, \mathtt{com}_{\mathbf{a}}, out_{\mathcal{O}}$.

a If this fails, output $\perp$ and abort.

Fig. 16. The subfunctionality-task of $\mathcal{F}_{\text{PPA}}$ for performing direct computations.

Outsourced Analytics. The behavior of $\mathcal{F}_{\text{PPA}}$ for analytical tasks is shown in Fig. 17. The operator input is verified in the same way as for Bookkeeping. After successful verification $\mathcal{F}_{\text{PPA}}$ internally reconstructs the shares for each user, computes the function, and *masks* the outputs. Since MPC leaks no intermediate results, this means that $\mathcal{F}_{\text{PPA}}$ only outputs *cryptographically protected* private outputs $out_{\mathcal{U}}$ for each participating user. $\mathcal{F}_{\text{PPA}}$ also provides $\mathcal{P}$ with additional information

Input $\mathcal{P}$: $(OA(k), \mathtt{com}_{fp}, \{(\mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}_z}, \mathrm{sh}^{(\mathcal{P})}_{in_z}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathtt{OUT}z}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathtt{UNV}z}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\alpha}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathbf{s}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathbf{a}}})\}_{z=1}^{Z_k})$.
Input $\mathcal{O}$: $(OA(k), fp, \mathtt{unv}_{fp}, \{(\mathrm{sh}^{(\mathcal{O})}_{\mathcal{UH}_z}, \mathrm{sh}^{(\mathcal{O})}_{in_z}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{OUT}z}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{UNV}z}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\alpha}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathbf{s}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathbf{a}}})\}_{z=1}^{Z_k}, in_{\mathcal{O}})$.

1. Check[a] $\mathsf{COM'.Unv}(\mathtt{com}_{fp}, \mathtt{unv}_{fp}, fp) = 1$.
2. For every z from 1 to Z_k:
   - Combine[a] $\mathcal{UH}_z \leftarrow \pi_{\text{Share}}\text{-}\mathit{Combine}(\mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}_z}, \mathrm{sh}^{(\mathcal{O})}_{\mathcal{UH}_z})$.
   - Combine[a] $in_{\mathcal{U}_z} \leftarrow \pi_{\text{Share}}\text{-}\mathit{Combine}(\mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}_z}}, \mathrm{sh}^{(\mathcal{O})}_{in_{\mathcal{U}_z}})$.
   - Combine[a] $\mathtt{o}_{\mathtt{OUT}z} \leftarrow \pi_{\text{Share}}\text{-}\mathit{Combine}(\mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathtt{OUT}z}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{OUT}z}})$.
   - Combine[a] $\mathtt{o}_{\mathtt{UNV}z} \leftarrow \pi_{\text{Share}}\text{-}\mathit{Combine}(\mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathtt{UNV}z}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{UNV}z}})$.
   - Combine[a] $\mathtt{o}_{\alpha z} \leftarrow \pi_{\text{Share}}\text{-}\mathit{Combine}(\mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\alpha z}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\alpha z}})$.
   - Combine[a] $\mathtt{o}_{\mathbf{s}z} \leftarrow \pi_{\text{Share}}\text{-}\mathit{Combine}(\mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathbf{s}z}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathbf{s}z}})$.
   - Combine[a] $\mathtt{o}_{\mathbf{a}z} \leftarrow \pi_{\text{Share}}\text{-}\mathit{Combine}(\mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathbf{a}z}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathbf{a}z}})$.
3. Compute $(\{(\alpha_z, \mathbf{s}_z, \mathbf{a}_z, out_{\mathcal{U}_z})\}_{z=1}^{Z_k}, out_{\mathcal{O}}) \leftarrow \Delta(OA, fp, k, \{(\mathcal{UH}_z, in_{\mathcal{U}_z})\}_{z=1}^{Z}, in_{\mathcal{O}})$.
4. For every z from 1 to Z_k:
   - $(\mathtt{com}_{\mathbf{a}_z}, \mathtt{unv}_{\mathbf{a}_z}) \leftarrow \mathsf{COM.Com}(\mathbf{a}_z)$
   - $\mathtt{c}_{\mathtt{unv}_{\mathbf{a}_z}} :\!\!- \mathtt{unv}_{\mathbf{a}_z} + \mathtt{o}_{\mathtt{UNV}z}$.
   - $\mathtt{c}_{out_{\mathcal{U}_z}} :\!\!- out_{\mathcal{U}_z} + \mathtt{o}_{\mathtt{OUT}z}$.
   - $\mathtt{c}_{\alpha_z} :\!\!- \alpha_z + \mathtt{o}_{\alpha z}$.
   - $\mathtt{c}_{\mathbf{s}_z} :\!\!- \mathbf{s}_z + \mathtt{o}_{\mathbf{s}z}$.
   - $\mathtt{c}_{\mathbf{a}_z} :\!\!- \mathbf{a}_z + \mathtt{o}_{\mathbf{a}z}$.

Output $\mathcal{P}$: $\{(\mathtt{c}_{\alpha_z}, \mathtt{c}_{\mathbf{s}_z}, \mathtt{c}_{\mathbf{a}_z}, \mathtt{c}_{\mathtt{unv}_{\mathbf{a}_z}}, \mathtt{c}_{out_{\mathcal{U}_z}})\}_{z=1}^{Z_k}$.
Output $\mathcal{O}$: $\{(\alpha, \mathbf{s}, \mathtt{com}_{\mathbf{a}_z}, \mathtt{c}_{out_{\mathcal{U}_z}})\}_{z=1}^{Z_k}, out_{\mathcal{O}}$.

Fig. 17. The subfunctionality-task of $\mathcal{F}_{\text{PPA}}$ for performing Outsourced Analytics.

The update itself consists of three parts, namely a permutation α, a list of updates that are applied directly $\mathbf{s}$, and an additional increment $\mathbf{a}$. We stress that while α and $\mathbf{s}$ are public values, the addition vector $\mathbf{a}$ is only to be known by the user. Hence it is not output to either of the proxy or operator directly. To increase the Users privacy guarantees, while still allowing guarantees for $\mathcal{O}$ that $\mathcal{UH}$ has been updated correctly, $\mathcal{F}_{\text{PPA}}$ computes a *commitment* $\mathtt{com}_{\mathbf{a}}$ with corresponding unveil information $\mathtt{unv}_{\mathbf{a}}$ on the addition vector $\mathbf{a}$, and outputs $\mathtt{com}_{\mathbf{a}}$ to $\mathcal{O}$. The remaining values, that is, $\mathbf{a}$ and $\mathtt{unv}_{\mathbf{a}}$, are masked with $\mathtt{o}_{\mathbf{a}}$ and $\mathtt{o}_{\mathtt{UNV}}$, respectively, and output to $\mathcal{P}$. This hides all information from $\mathcal{P}$, as only the User knows the One-Time Pads.

A Robust Secret Sharing-protocol. Both our subfunctionality and our protocol make use of a *sharing* protocol and its corresponding *combine* protocol. Those are required so that the user can *share* information with $\mathcal{P}$ and $\mathcal{O}$ in such a way, that no party —neither $\mathcal{P}$ nor $\mathcal{O}$ —can change the shares unnoticed. The protocols are shown in Fig. 18.

Essentially, this comes down to *additive secret sharing*, where the dealer does not only send the additive share of a value to each party, but also adds a *commit-*

Protocol π_{Share}

This protocol facilitates Robust Secret Sharing. It is parameterized by an *unconditionally hiding* and *computationally binding homomorphic commitment scheme* COM, for which inverse elements in the image group of COM are efficiently computable.

π_{Share}-*Share*:

Input: Value x.

- Sample uniformly random $x^{(0)}$.
- $x^{(1)} \coloneqq x - x^{(0)}$.
- $\left(\mathtt{com}_x^{(0)}, \mathtt{unv}_x^{(0)}\right) \leftarrow \mathsf{COM.Com}\left(x^{(0)}\right)$.
- $\left(\mathtt{com}_x^{(1)}, \mathtt{unv}_x^{(1)}\right) \leftarrow \mathsf{COM.Com}\left(x^{(1)}\right)$.
- $\mathtt{sh}_x^{(0)} \coloneqq \left(x^{(0)}, \mathtt{com}_x^{(1)}, \mathtt{unv}_x^{(0)}\right)$.
- $\mathtt{sh}_x^{(1)} \coloneqq \left(x^{(1)}, \mathtt{com}_x^{(0)}, \mathtt{unv}_x^{(1)}\right)$.

Output: Shares $\left(\mathtt{sh}_x^{(0)}, \mathtt{sh}_x^{(1)}\right)$.

π_{Share}-*Combine*:

Input: Shares $\left(\mathtt{sh}_x^{(0)}, \mathtt{sh}_x^{(1)}\right)$.

- $\text{Parse}^{\text{a}}\left(x^{(0)}, \mathtt{com}_x^{(1)}, \mathtt{unv}_x^{(0)}\right) \coloneqq \mathtt{sh}_x^{(0)}$.
- $\text{Parse}^{\text{a}}\left(x^{(1)}, \mathtt{com}_x^{(0)}, \mathtt{unv}_x^{(1)}\right) \coloneqq \mathtt{sh}_x^{(1)}$.
- $\text{Check}^{\text{a}}\ \mathsf{COM.Unv}\left(\mathtt{com}_x^{(0)}, \mathtt{unv}_x^{(0)}, x^{(0)}\right) \stackrel{?}{=} 1$.
- $\text{Check}^{\text{a}}\ \mathsf{COM.Unv}\left(\mathtt{com}_x^{(1)}, \mathtt{unv}_x^{(1)}, x^{(1)}\right) \stackrel{?}{=} 1$.
- $x \coloneqq x^{(0)} + x^{(1)}$.

Output: Value x.

a If this fails, output $\perp$ and abort.

Fig. 18. Protocol for Robust Secret Sharing (RSS).

Protocol π_{Verify}

This protocol facilitates sharing and verification of logbooks. It is parameterized by an *unconditionally hiding* and *computationally binding homomorphic commitment scheme* COM, for which inverse elements in the image group of COM are efficiently computable, and an EUF-CMA-secure, structure-preserving signature scheme SIG.

Input: Message $\mathtt{VfyL}$, lb $\lambda \coloneqq (\mathcal{UH}, \mathtt{com}_{\mathcal{UH}}, \mathtt{unv}_{\mathcal{UH}}, ser, \mathtt{com}_{ser}, \mathtt{unv}_{ser}, lin, \mathtt{com}_{lin}, \mathtt{unv}_{lin}, id, \mathtt{com}_{id}, \mathtt{unv}_{id}, \sigma)$.

- $\text{Check}^{\text{a}}\ \mathsf{COM.Unv}(\mathtt{com}_{\mathcal{UH}}, \mathtt{unv}_{\mathcal{UH}}, \mathcal{UH}) \stackrel{?}{=} 1$
- $\text{Check}^{\text{a}}\ \mathsf{COM.Unv}(\mathtt{com}_{ser}^{(\mathcal{O})}, \mathtt{unv}_{ser}^{(\mathcal{O})}, ser^{(\mathcal{O})}) \stackrel{?}{=} 1$
- $\text{Check}^{\text{a}}\ \mathsf{COM.Unv}(\mathtt{com}_{lin}, \mathtt{unv}_{lin}, lin) \stackrel{?}{=} 1$
- $\text{Check}^{\text{a}}\ \mathsf{COM.Unv}(\mathtt{com}_{id}, \mathtt{unv}_{id}, id) \stackrel{?}{=} 1$
- $\text{Check}^{\text{a}}\ \mathsf{SIG.Vfy}(\mathsf{vk}_{\mathcal{O}}, \sigma, \mathtt{com}_{\mathcal{UH}} \| \mathtt{com}_{ser} \| \mathtt{com}_{lin} \| \mathtt{com}_{id}) \stackrel{?}{=} 1$

Output: λ

Input: Message $\mathtt{ShareL}$, lb $\lambda \coloneqq (\mathcal{UH}, \mathtt{com}_{\mathcal{UH}}, \mathtt{unv}_{\mathcal{UH}}, ser, \mathtt{com}_{ser}, \mathtt{unv}_{ser}, lin, \mathtt{com}_{lin}, \mathtt{unv}_{lin}, id, \mathtt{com}_{id}, \mathtt{unv}_{id}, \sigma)$.

- $\mathtt{o}_{\mathtt{OUT}} \sim \mathcal{IN}$
- $\mathtt{o}_{\mathtt{UNV}} \xleftarrow{r} \mathbb{Z}_p$
- $\mathtt{o}_{\alpha} \xleftarrow{r} \mathbb{Z}_p^m$
- $\mathtt{o}_{\mathbf{s}} \xleftarrow{r} \mathbb{Z}_p^m$
- $\mathtt{o}_{\mathbf{a}} \xleftarrow{r} \mathbb{Z}_p^m$
- $(\mathtt{sh}_{\mathcal{UH}}^{(\mathcal{P})}, \mathtt{sh}_{\mathcal{UH}}^{(\mathcal{O})}) \leftarrow \pi_{\text{Share}}\text{-}\mathit{Share}(\mathcal{UH})$
- $(\mathtt{sh}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \mathtt{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}) \leftarrow \pi_{\text{Share}}\text{-}\mathit{Share}(in_{\mathcal{U}})$
- $(\mathtt{sh}_{\mathtt{o}_{\mathtt{OUT}}}^{(\mathcal{P})}, \mathtt{sh}_{\mathtt{o}_{\mathtt{OUT}}}^{(\mathcal{O})}) \leftarrow \pi_{\text{Share}}\text{-}\mathit{Share}(\mathtt{o}_{\mathtt{OUT}})$
- $(\mathtt{sh}_{\mathtt{o}_{\mathtt{UNV}}}^{(\mathcal{P})}, \mathtt{sh}_{\mathtt{o}_{\mathtt{UNV}}}^{(\mathcal{O})}) \leftarrow \pi_{\text{Share}}\text{-}\mathit{Share}(\mathtt{o}_{\mathtt{UNV}})$
- $(\mathtt{sh}_{\mathtt{o}_{\alpha}}^{(\mathcal{P})}, \mathtt{sh}_{\mathtt{o}_{\alpha}}^{(\mathcal{O})}) \leftarrow \pi_{\text{Share}}\text{-}\mathit{Share}(\mathtt{o}_{\alpha})$
- $(\mathtt{sh}_{\mathtt{o}_{\mathbf{s}}}^{(\mathcal{P})}, \mathtt{sh}_{\mathtt{o}_{\mathbf{s}}}^{(\mathcal{O})}) \leftarrow \pi_{\text{Share}}\text{-}\mathit{Share}(\mathtt{o}_{\mathbf{s}})$
- $(\mathtt{sh}_{\mathtt{o}_{\mathbf{a}}}^{(\mathcal{P})}, \mathtt{sh}_{\mathtt{o}_{\mathbf{a}}}^{(\mathcal{O})}) \leftarrow \pi_{\text{Share}}\text{-}\mathit{Share}(\mathtt{o}_{\mathbf{a}})$

Output: $(\mathtt{o}_{\mathtt{OUT}}, \mathtt{o}_{\mathtt{UNV}}, \mathtt{o}_{\alpha}, \mathtt{o}_{\mathbf{s}}, \mathtt{o}_{\mathbf{a}}, \{\mathtt{sh}_{\mathcal{UH}}^{(x)}, \mathtt{sh}_{in_{\mathcal{U}}}^{(x)}, \mathtt{sh}_{\mathtt{o}_{\mathtt{OUT}}}^{(x)}, \mathtt{sh}_{\mathtt{o}_{\mathtt{UNV}}}^{(x)}, \mathtt{sh}_{\mathtt{o}_{\alpha}}^{(x)}, \mathtt{sh}_{\mathtt{o}_{\mathbf{s}}}^{(x)}, \mathtt{sh}_{\mathtt{o}_{\mathbf{a}}}^{(x)}\}_{x \in \{\mathcal{P}, \mathcal{O}\}})$

Input: Message $\mathtt{VfyS}$, inputs $\mathtt{com}_{\mathcal{UH}}, \mathtt{unv}_{\mathcal{UH}}, \mathcal{UH}, \mathtt{com}_{in_{\mathcal{U}}}, \mathtt{unv}_{in_{\mathcal{U}}}, in_{\mathcal{U}}, \mathtt{com}_{\mathtt{o}_{\mathtt{OUT}}}, \mathtt{unv}_{\mathtt{o}_{\mathtt{OUT}}}, \mathtt{o}_{\mathtt{OUT}}, \mathtt{com}_{\mathtt{o}_{\mathtt{UNV}}}, \mathtt{unv}_{\mathtt{o}_{\mathtt{UNV}}}, \mathtt{o}_{\mathtt{UNV}}, \mathtt{com}_{\mathtt{o}_{\alpha}}, \mathtt{unv}_{\mathtt{o}_{\alpha}}, \alpha, \mathtt{com}_{\mathtt{o}_{\mathbf{s}}}, \mathtt{unv}_{\mathtt{o}_{\mathbf{s}}}, \mathbf{s}, \mathtt{com}_{\mathtt{o}_{\mathbf{a}}}, \mathtt{unv}_{\mathtt{o}_{\mathbf{a}}}, \mathbf{a}$

- $\text{Check}^{\text{a}}\ \mathsf{Unv}(\mathtt{com}_{\mathcal{UH}}, \mathtt{unv}_{\mathcal{UH}}, \mathcal{UH}) = 1$
- $\text{Check}^{\text{a}}\ \mathsf{Unv}(\mathtt{com}_{in_{\mathcal{U}}}, \mathtt{unv}_{in_{\mathcal{U}}}, in_{\mathcal{U}}) = 1$
- $\text{Check}^{\text{a}}\ \mathsf{Unv}(\mathtt{com}_{\mathtt{o}_{\mathtt{OUT}}}, \mathtt{unv}_{\mathtt{o}_{\mathtt{OUT}}}, \mathtt{o}_{\mathtt{OUT}}) = 1$
- $\text{Check}^{\text{a}}\ \mathsf{Unv}(\mathtt{com}_{\mathtt{o}_{\mathtt{UNV}}}, \mathtt{unv}_{\mathtt{o}_{\mathtt{UNV}}}, \mathtt{o}_{\mathtt{UNV}}) = 1$
- $\text{Check}^{\text{a}}\ \mathsf{Unv}(\mathtt{com}_{\mathtt{o}_{\alpha}}, \mathtt{unv}_{\mathtt{o}_{\alpha}}, \mathtt{o}_{\alpha}) = 1$
- $\text{Check}^{\text{a}}\ \mathsf{Unv}(\mathtt{com}_{\mathtt{o}_{\mathbf{s}}}, \mathtt{unv}_{\mathtt{o}_{\mathbf{s}}}, \mathtt{o}_{\mathbf{s}}) = 1$
- $\text{Check}^{\text{a}}\ \mathsf{Unv}(\mathtt{com}_{\mathtt{o}_{\mathbf{a}}}, \mathtt{unv}_{\mathtt{o}_{\mathbf{a}}}, \mathtt{o}_{\mathbf{a}}) = 1$

Output: ok

a If this fails, output $\perp$ and abort.

Fig. 19. Protocol for User logbook verification.

ment on the respective *other* parties share. Additionally, each party obtains unveil information on its own commitment.

The combine protocol then lets each party send their own value and their unveil information; the other party only accepts, if the unveil information matches the commitment it received during the sharing-phase.

Additionally, since this task is used after each user interaction during any task, we have a *verification* protocol π_{Verify}, with which a user can check that its logbook λ has been created correctly.

The protocol basically verifies that all the commitments are on the correct values and that the signature is correct.

E.2 Our Protocol

With those building blocks we can now provide our full protocol. The protocol runs in the

Protocol π_{PUBA}

This protocol facilitates user-centric Privacy-Preserving Analytics.

π*-Shared State*:

Each user $\mathcal{U}$ stores:

- User logbook λ
- User public key $\mathsf{pk}_{\mathcal{U}}$
- One-Time Pads o_{α}, $\mathsf{o}_{\mathbf{s}}$, $\mathsf{o}_{\mathbf{a}}$, $\mathsf{o}_{\mathtt{OUT}}$ and $\mathsf{o}_{\mathtt{UNV}}$
- Verification key $\mathsf{vk}_{\mathcal{O}}$ of the operator
- Verification key $\mathsf{vk}_{\mathcal{T}}$ of the Trusted Signing Authority

Each proxy $\mathcal{P}$ stores:

- Verification key $\mathsf{vk}_{\mathcal{O}}$ of the operator
- Verification key $\mathsf{vk}_{\mathcal{T}}$ of the Trusted Signing Authority
- Mapping $f_{\mathrm{OI}}^{(\mathcal{P})}$ on $\{1,\ldots,K\}$ that maps k to a list $f_{\mathrm{OI}}^{(\mathcal{P})}(k)$ of entries $(lin, \mathsf{sh}_{\mathcal{UH}}^{(\mathcal{P})}, \mathsf{sh}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \mathsf{sh}_{\mathsf{o}_{\mathtt{OUT}}}^{(\mathcal{P})}, \mathsf{sh}_{\mathsf{o}_{\alpha}}^{(\mathcal{P})}, \mathsf{sh}_{\mathsf{o}_{\mathbf{s}}}^{(\mathcal{P})}, \mathsf{sh}_{\mathsf{o}_{\mathbf{a}}}^{(\mathcal{P})})$
- Partial mapping $f_{\mathrm{UP}}^{(\mathcal{P})}$ on $\{lin\}\colon lin \mapsto (\mathsf{c}_{\alpha}, \mathsf{c}_{\mathbf{s}}, \mathsf{c}_{\mathbf{a}}, \mathsf{c}_{out_{\mathcal{U}}})$

The operator $\mathcal{O}$ stores:

- Signature key pair $(\mathsf{vk}_{\mathcal{O}}, \mathsf{sk}_{\mathcal{O}})$
- List $\mathsf{L}_{\mathsf{SER}}$ of observed serial numbers
- Mapping $f_{\mathrm{OI}}^{(\mathcal{O})}$ on $\{pid_{\mathcal{P}}\} \times \{1,\ldots,K\}$ that maps $(pid_{\mathcal{P}}, k)$ to a list $f_{\mathrm{OI}}^{(\mathcal{O})}(pid_{\mathcal{P}}, k)$ of entries $(lin, \mathsf{sh}_{\mathcal{UH}}^{(\mathcal{O})}, \mathsf{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}, \mathsf{sh}_{\mathsf{o}_{\mathtt{OUT}}}^{(\mathcal{O})}, \mathsf{sh}_{\mathsf{o}_{\alpha}}^{(\mathcal{O})}, \mathsf{sh}_{\mathsf{o}_{\mathbf{s}}}^{(\mathcal{O})}, \mathsf{sh}_{\mathsf{o}_{\mathbf{a}}}^{(\mathcal{O})})$
- Partial mapping $f_{\mathrm{UP}}^{(\mathcal{O})}$ on $\{lin\}\colon lin \mapsto (\alpha, \mathbf{s}, \mathsf{com}_{\mathbf{a}}, \mathsf{c}_{out_{\mathcal{U}}})$
- Mapping f_{FP} on fp that maps Function Parameters fp to a list $f_{\mathrm{FP}}(fp)$ of tuples $(\mathtt{com}_{fp}, \mathtt{unv}_{fp}, \sigma_{fp})$.

The Trusted Signing Authority $\mathcal{T}$ stores:

- Signature key pair $(\mathsf{vk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$

Tasks:

- π*-Init* (Fig. 21)
- π*-Sign Function Parameter* (Fig. 22)
- π*-User Registration* (Fig. 24)
- π*-Bookkeeping* (Fig. 27)
- π*-Outsource* (Fig. 31)
- π*-Outsourced Analytics* (Fig. 34)
- π*-Update* (Fig. 36)

Fig. 20. Shared state of the protocol π_{PUBA}.

$\{\mathcal{F}_{\mathrm{PPA}}, \mathcal{F}_{\mathrm{BB}}, \mathcal{F}_{\mathrm{KE}}, \mathcal{F}_{\mathrm{CRS}}\}$-hybrid model. Those are used for the following purpose:

$\mathcal{F}_{\mathrm{PPA}}$ has been explained earlier in this chapter.

$\mathcal{F}_{\mathrm{BB}}$ is used during User Registration to ensure that no user creates more than one account and to verify that the user really is who it claims to be.

$\mathcal{F}_{\mathrm{KE}}$ is used to set up private channels which are not subject to eavesdropping. We implicitly assume that prior to each protocol execution each pair of participating parties calls an instance of $\mathcal{F}_{\mathrm{KE}}$ to create a secure *session key* which encrypts each message exchanged.

$\mathcal{F}_{\mathrm{CRS}}$ is required for our pairing-based instantiations of the Zero-Knowledge protocol.

The full protocol is parameterized by:

- an Ideal Functionality $\mathcal{F}_{\mathrm{PPA}}$ that handles problem-specific tasks of the computation, which works on inputs defined by $\mathcal{IN}$.
- an Ideal Functionality $\mathcal{F}_{\mathrm{BB}}$ that provides a bulletin board functionality.
- an Ideal Functionality $\mathcal{F}_{\mathrm{KE}}$ that provides a secure key exchange functionality. We implicitly assume that all parties call this to obtain a pair-wise shared secret key k which both of them use throughout the subsession.
- a IND-CPA-secure symmetric encryption scheme $\mathsf{ENC} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ which parties implicitly use for all messages using their shared key k.
- a common reference string *crs*.
- group parameters gp.
- an *unconditionally hiding* and *computationally binding homomorphic commitment scheme* COM for which inverse elements in the image group of COM are efficiently computable. This is instantiated as the same scheme used in π_{Share}.
- an *unconditionally hiding* and *computationally binding* commitment scheme COM'.
- an EUF-CMA-secure, structure-preserving *signature scheme* SIG. Again, this scheme is the same as the one used for π_{Share}.
- a trapdoor dual-mode zero-knowledge proof-of-knowledge scheme POK based on *crs* where an honestly chosen *crs* yields overwhelming completeness and negligible soundness property whereas one mode offers F-extractability and the other mode offers simulatability.
- An EUF-CMA-secure signature scheme SIG'.

The protocol from Fig. 20 makes heavy use of the languages defined in Figs. 26, 29, 33 and 38. The intuition behind each task is as follows:

Init. Before any other task can be executed the operator and the TSA have to run the Init task. While we modeled it as a two-party task the two parties never interact with one another and only create their signing keys and register them at $\mathcal{F}_{\text{BB}}$.

π-*Init*:
Input $\mathcal{O}$: (init)
Input $\mathcal{T}$: (init)

$\mathcal{O}$: Generate signature key pair:
- $(\mathsf{vk}_{\mathcal{O}}, \mathsf{sk}_{\mathcal{O}}) \leftarrow \mathsf{SIG.Gen}(\mathsf{gp})$.
- Call$^{\text{a}}$ $\mathcal{F}_{\text{BB}}$ instance $sid_{\mathcal{F}_{\text{BB}}}$ with input $(\texttt{Register}, pid_{\mathcal{O}}, \mathsf{vk}_{\mathcal{O}})$.

$\mathcal{T}$: Generate signature key pair:
- $(\mathsf{vk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}) \leftarrow \mathsf{SIG}'.\mathsf{Gen}(\mathsf{gp})$.
- Call$^{\text{a}}$ $\mathcal{F}_{\text{BB}}$ instance $sid_{\mathcal{F}_{\text{BB}}}$ with input $(\texttt{Register}, pid_{\mathcal{T}}, \mathsf{vk}_{\mathcal{T}})$.

Output $\mathcal{O}$: (ok)
Output $\mathcal{T}$: (ok)

Fig. 21. The protocol π-*Init* for the Init-task (Fig. 7).

Sign Function Parameter. Our framework is designed such that it hides the function details from the user. While we believe this to be extremely important on one hand as the operator potentially spent a lot of time and money in the creation of the model, we stress that fully hiding the function *and* the result from the user allows for a trivial attack on the users anonymity if the operator inputs a non-privacy-preserving function such as the identity.

π-*Sign Function Parameter*:
Input $\mathcal{O}$: Message SFP, FPs *fp*, task *task*, aux. input $in_{\mathcal{O}}$
Input $\mathcal{T}$: Message SFP, aux. input $in_{\mathcal{T}}$.

$\mathcal{O}$: $(\mathsf{com}_{fp}, \mathsf{unv}_{fp}) \leftarrow \mathsf{COM}'.\mathsf{Com}(fp)$.
$\mathcal{O} \nrightarrow \mathcal{T}$: $(fp, \mathsf{com}_{fp}, \mathsf{unv}_{fp}, task, in_{\mathcal{O}})$
$\mathcal{T}$: Check$^{\text{a}}$ that $\Delta(\text{SFP}, fp, task, in_{Operator}, in_{\mathcal{T}}) = 1$.
$\mathcal{T}$: Check$^{\text{a}}$ that $\mathsf{COM}'.\mathsf{Unv}(\mathsf{com}_{fp}, \mathsf{unv}_{fp}, fp) = 1$.
$\mathcal{T}$: $\sigma_{fp} \leftarrow \mathsf{SIG}'.\mathsf{Sgn}(\mathsf{sk}_{\mathcal{T}}, (task, \mathsf{com}_{fp}))$.
$\mathcal{O} \nleftarrow \mathcal{T}$: (σ_{fp})
$\mathcal{O}$: $f_{\text{FP}}(fp) := (\mathsf{com}_{fp}, \mathsf{unv}_{fp}, \sigma_{fp})$.

Output $\mathcal{T}$: Message ok
Output $\mathcal{O}$: Message ok

Fig. 22. The protocol π-*Sign Function Parameter* for the Sign Function Parameter-task (Fig. 8).

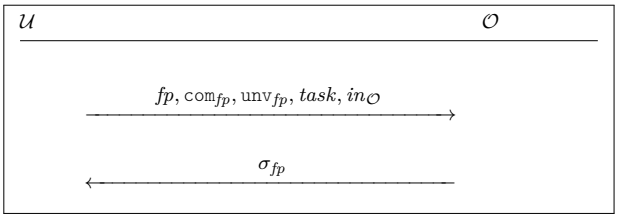

Fig. 23. Message overview for π-*Sign Function Parameter*.

As a compromise we suggest a third party which *certifies* that operator input meets certain privacy criteria: the Trusted Signing Authority (TSA) $\mathcal{T}$. The key idea is that the function to-be-computed only has a very generic design—say in the form of general logistic regression or a neural network—but the function specifics are stored in Function Parameters (FPs) *fp*. In order to circumvent the attack sketched above and ensure that the operator only ever uses *valid* inputs the tasks User Registration, Bookkeeping and Outsourced Analytics require *signed* FPs. To that end, the task Sign Function Parameter lets the operator input some FPs *fp* which are to be used for computation of some task $task \in \{\text{UReg}, BK(k), OA(k)\}$ alongside a commitment com_{fp} and corresponding unveil information unv_{fp}. The TSA verifies them using the application-dependent Δ to ensure that they match the required privacy standards. We assume those privacy standards to be public knowledge.

If a given set of FPs verifies the operator obtains a signature σ_{fp} on $(task, \mathsf{com}_{fp})$ and uses com_{fp} as future certificate for *fp*: Before using the FPs *fp* the $\mathcal{O}$ sends com_{fp} and σ_{fp} to the respective other party (the user or the proxy, respectively who then input com_{fp} into $\mathcal{F}_{\text{PPA}}$. The operator $\mathcal{O}$ inputs *fp* and unv_{fp}. Before starting the computation defined by Δ the commitment is verified. The binding property of COM' ensures that the operator can only use FPs which were signed by $\mathcal{T}$.

User Registration. The protocol for User Registration is shown in Figs. 24 and 25. We want to ensure that each user has at most a single logbook. To ensure that no user can create multiple accounts we make use of the *Bulletin Board* functionality $\mathcal{F}_{\text{BB}}$. This lets the user register a key *exactly once*. The user has to publish a fresh *public key* to the bulletin board which is taken from a pairing group $e(\cdot, \cdot)$. The user can only register if it knows the corresponding secret key and hence proves knowledge of it using the ZK proof from Fig. 26. This proves knowledge of both unv_{id} and *id* such that the pairing equation is fulfilled and such that the commitment com_{id} unveils to $\mathsf{pk}_{\mathcal{U}}$. The operator $\mathcal{O}$ then only accepts if this proof is valid and if the public key has

π-*User Registration*:
Input $\mathcal{U}$: Message UReg, aux. input $in_{\mathcal{U}}$
Input $\mathcal{O}$: Message UReg, Function Parameters fp, aux. input $in_{\mathcal{O}}$

$\mathcal{U}$: Fetch verification keys:
- Call[a] $\mathcal{F}_{\text{BB}}$ instance $sid_{\mathcal{F}_{\text{BB}}}$ with input $(\texttt{Retrieve}, pid_{\mathcal{O}})$ and store output $\mathsf{vk}_{\mathcal{O}}$.
- Call[a] $\mathcal{F}_{\text{BB}}$ instance $sid_{\mathcal{F}_{\text{BB}}}$ with input $(\texttt{Retrieve}, pid_{\mathcal{T}})$ and store output $\mathsf{vk}_{\mathcal{T}}$.

Draw User secret key:
- Draw random User ID $id \xleftarrow{r} \mathbb{Z}_p$.
- Compute User public key $\mathsf{pk}_{\mathcal{U}} := id \cdot g_1$.
- Call $\mathcal{F}_{\text{BB}}$ instance $sid_{\mathcal{F}_{\text{BB}}}$ with input $(\texttt{Register}, pid_{\mathcal{U}}, \mathsf{pk}_{\mathcal{U}})$.
- $(\mathtt{com}_{id}, \mathtt{unv}_{id}) \leftarrow \mathsf{COM.Com}(id)$

Calculate proof of user secret key knowledge:
- $stmt := (\mathsf{pk}_{\mathcal{U}}, \mathtt{com}_{id})$
- $wit := (\mathtt{unv}_{id}, id' := id \cdot g_2)$
- $\Pi \leftarrow \mathsf{Prove}(stmt, wit, L_{\text{R}})$, using L_{R} from Fig. 26.

Draw share of new serial number:
- $(ser^{\text{new}})^{(\mathcal{U})} \xleftarrow{r} \mathbb{Z}_p$
- $(\mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}}, \mathtt{unv}^{(\mathcal{U})}_{ser^{\text{new}}}) \leftarrow \mathsf{COM.Com}((ser^{\text{new}})^{(\mathcal{U})})$

$\mathcal{U} \nrightarrow \mathcal{O}$: $(\Pi, \mathtt{com}_{id}, \mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}})$

$\mathcal{O}$: Fetch FP information:
- Set[a] $(\mathtt{com}_{fp}, \mathtt{unv}_{fp}, \sigma_{fp}) := f_{\text{FP}}(fp)$.

$\mathcal{O}$: Fetch user public key:
- Call $\mathcal{F}_{\text{BB}}$ instance $sid_{\mathcal{F}_{\text{BB}}}$ with input $(\texttt{Retrieve}, pid_{\mathcal{U}})$ and receive output $\mathsf{pk}_{\mathcal{U}}$.

Check[a] that no user with public key $\mathsf{pk}_{\mathcal{U}}$ has registered yet.
Check proof:
- $stmt := (\mathsf{pk}_{\mathcal{U}}, \mathtt{com}_{id})$
- Check[a] $\mathsf{Vfy}(\Pi, stmt, L_{\text{R}}) = 1$

$\mathcal{U} \leftarrow \mathcal{O}$: $(\mathtt{com}_{fp}, \sigma_{fp})$

$\mathcal{U}$: Check[a] that $\mathsf{SIG}'.\mathsf{Vfy}((\texttt{UReg}, \mathtt{com}_{fp}), \sigma_{fp}, \mathsf{vk}_{\mathcal{T}}) = 1$.

$\mathcal{U} \nleftrightarrow \mathcal{O}$: Compute initial user history:

$\mathcal{U} \rightarrow \mathcal{F}_{\text{PPA}}$: $(\texttt{UReg}, \mathtt{com}_{fp}, in_{\mathcal{U}})$

$\mathcal{O} \rightarrow \mathcal{F}_{\text{PPA}}$: $(\texttt{UReg}, fp, \mathtt{unv}_{fp}, in_{\mathcal{O}})$

$\mathcal{U} \leftarrow \mathcal{F}_{\text{PPA}}$: $(\mathcal{UH}, \mathtt{unv}_{\mathcal{UH}}, out_{\mathcal{U}})$

$\mathcal{O} \leftarrow \mathcal{F}_{\text{PPA}}$: $(\mathtt{com}_{\mathcal{UH}}, out_{\mathcal{O}})$

$\mathcal{O}$: Draw share of new serial number:
- $(ser^{\text{new}})^{(\mathcal{O})} \xleftarrow{r} \mathbb{Z}_p$
- $(\mathtt{com}^{(\mathcal{O})}_{ser^{\text{new}}}, \mathtt{unv}^{(\mathcal{O})}_{ser^{\text{new}}}) \leftarrow \mathsf{COM.Com}((ser^{\text{new}})^{(\mathcal{O})})$

Compute commitments and signature for initial User History:
- $\mathtt{com}_{ser^{\text{new}}} := \mathtt{com}^{(\mathcal{O})}_{ser^{\text{new}}} \oplus \mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}}$
- $(\mathtt{com}_{lin}, \mathtt{unv}_{lin}) \leftarrow \mathsf{COM.Com}(0)$
- $\sigma \leftarrow \mathsf{SIG.Sgn}(\mathsf{sk}_{\mathcal{O}}, \mathtt{com}_{\mathcal{UH}} \| \mathtt{com}_{ser^{\text{new}}} \| \mathtt{com}_{lin} \| \mathtt{com}_{id})$

$\mathcal{U} \nleftarrow \mathcal{O}$: $(\mathtt{com}_{\mathcal{UH}}, (ser^{\text{new}})^{(\mathcal{O})}, \mathtt{com}^{(\mathcal{O})}_{ser^{\text{new}}}, \mathtt{unv}^{(\mathcal{O})}_{ser^{\text{new}}}, \mathtt{com}_{lin}, \mathtt{unv}_{lin}, \sigma)$

$\mathcal{U}$: Set[a] and store $\lambda^{\text{new}} := \pi_{\text{Verify}}(\texttt{VfyL}, (\mathcal{UH}, \mathtt{com}_{\mathcal{UH}}, \mathtt{unv}_{\mathcal{UH}}, ((ser^{\text{new}})^{(\mathcal{U})} + (ser^{\text{new}})^{(\mathcal{O})}), (\mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}} \oplus \mathtt{com}^{(\mathcal{O})}_{ser^{\text{new}}}), (\mathtt{unv}^{(\mathcal{U})}_{ser^{\text{new}}} \oplus \mathtt{unv}^{(\mathcal{O})}_{ser^{\text{new}}}), 0, \mathtt{com}_{lin}, \mathtt{unv}_{lin}, id, \mathtt{com}_{id}, \mathtt{unv}_{id}, \sigma))$

Output $\mathcal{U}$: Initial user history $\mathcal{UH}$, aux. output $out_{\mathcal{U}}$
Output $\mathcal{O}$: Aux. output $out_{\mathcal{O}}$

[a] If this fails, output $\perp$ and abort.

Fig. 24. The protocol π-*User Registration* for User Registration (Fig. 9).

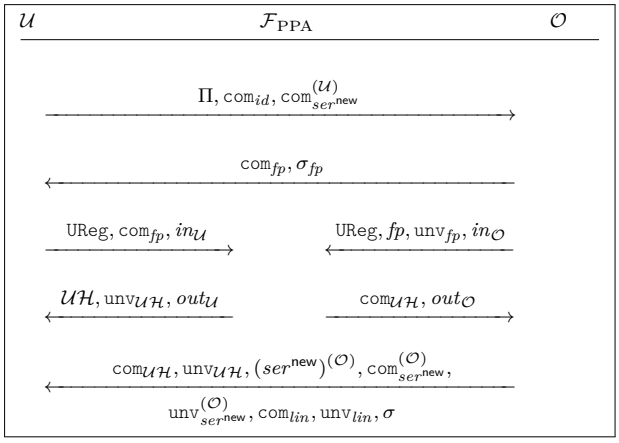

Fig. 25. Message overview for π-*User Registration*.

$$L_{\text{R}}[(\mathtt{unv}_{id}, id') \colon e(\mathsf{pk}_{\mathcal{U}}, g_2) = e(g_1, id') \wedge \mathsf{COM.Unv}(\mathtt{com}_{id}, \mathtt{unv}_{id}, \mathsf{pk}_{\mathcal{U}}) = 1]$$

Fig. 26. Language L_{R} from π-*User Registration* following the notation of [14].

not been used before. If this succeeds the two parties engage in the creation of the initial logbook.

The initial contents of the User History are computed via the MPC-framework from $\mathcal{F}_{\text{PPA}}$ according to Δ which can depend on additional secret FPs which are input by the operator. These are verified before the computation such that only FPs which were signed by the TSA $\mathcal{T}$ can be used for the computation.

After obtaining the initial User History this way the two parties create the initial logbook together. The principle is fairly similar in this task to all the following tasks: To ensure history freshness the user has to carry a serial number *ser* in its logbook which is invalidated during the next interaction with the operator. However, the serial number would allow tracking of the same user through different tasks—which we want to avoid. Hence, we decided to take inspiration from the Blum coin flipping protocol [11] over $\mathbb{Z}_p$ where only the user learns the outcome of the coin toss and the operator only learns that the outcome is randomly distributed over $\mathbb{Z}_p$. This is to ensure that no party picks a malicious serial number such as one that contains tracking information. Using the homomorphism of the commitment scheme COM the operator thus can create a commitment on the *actual* serial number based only on the commitment of the users share and its own commitment.

Using this information alongside the previously created commitments $\mathtt{com}_{lin}$ on the linking number (which is initially 0 as is only set if outsourcing-triplets have been started) and $\mathtt{com}_{\mathcal{UH}}$ and $\mathtt{com}_{id}$ which were obtained by the user the operator creates a *signature*.

This signature is only over commitments. Our instantiation of the signature scheme preserves the structure of the message ensures that this can later be used to prove in ZK that a signature on a commitment of a given value is known. Furthermore, since the commitment itself could theoretically be used for linking when seeing it again the operator only sees *rerandomized* versions of this commitment in future interactions alongside a ZK proof that this rerandomization is really on a commitment for which a signature made by the operator is known.

Note that all the communication happening here is *identifying* on *both* sides—even for the user. The *id* is uniquely identifiable along different tasks and hence *hidden* by the user from this point onwards. It is only used in the witness of Zero-Knowledge proofs in future interactions.

Bookkeeping. The protocol for directly updating the User History is given in Figs. 27 and 28. Essentially it is a wrapper around $\mathcal{F}_{\mathrm{PPA}}$: Before accessing the hybrid functionality the user proves to the operator that the latest input is used and the operator proves to the user that it will input valid Function Parameters that were signed by the TSA $\mathcal{T}$. The language L_{B}^{Val} for proving the former can be found in Fig. 29: The user proves correct rerandomization of the User History and that it knows serial, id and linking number for which it knows a signature from $\mathcal{O}$. For the latter—namely letting the operator prove that the FPs used for the computation are validly signed—we use an interactive protocol: $\mathcal{O}$ sends the commitment and the signature on the current task and the commitment to the user. Note that we do not hide from the user *which* FPs will be used—meaning that the user learns whether the same FPs have been used in previous computations—but only what those FPs *are*. Hence we do not require any form of rerandomization. The operator sends $\mathtt{com}_{fp}$ alongside the signature σ_{fp} directly to the user. The user then verifies the signature using the verification key $\mathsf{vk}_{\mathcal{T}}$ of $\mathcal{T}$ and only continues if this signature is valid. In that case the user inputs the commitment to $\mathcal{F}_{\mathrm{PPA}}$.

The operator inputs the corresponding opening information $\mathtt{unv}_{fp}$ and the clear values of *fp*. The definition of $\mathcal{F}_{\mathrm{PPA}}$ ensures that computation only happens if $\mathtt{unv}_{fp}$ (which is an input by the operator) successfully opens

$\{\pi\text{-}Bookkeeping^{(k)}\}_{k=1}^{K}$:
Input $\mathcal{U}$: Message $BK(k)$, aux. input $in_{\mathcal{U}}$
Input $\mathcal{O}$: Message $BK(k)$, FPs *fp*, aux. input $in_{\mathcal{O}}$
$\mathcal{U}$: Calculate lb-validity proof and draw share of new serial:
- $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, \widetilde{\mathtt{unv}_{\mathcal{UH}}}) \leftarrow \mathsf{COM.Rrnd}(\mathtt{com}_{\mathcal{UH}}, \mathtt{unv}_{\mathcal{UH}})$
- $(\widetilde{\mathtt{com}_{lin}}, \widetilde{\mathtt{unv}_{lin}}) \leftarrow \mathsf{COM.Rrnd}(\mathtt{com}_{lin}, \mathtt{unv}_{lin})$
- $(\widetilde{\mathtt{com}_{id}}, \widetilde{\mathtt{unv}_{id}}) \leftarrow \mathsf{COM.Rrnd}(\mathtt{com}_{id}, \mathtt{unv}_{id})$
- $stmt^{Val} := (\widetilde{\mathtt{com}_{\mathcal{UH}}}, ser, \widetilde{\mathtt{com}_{lin}}, \widetilde{\mathtt{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
- $wit^{Val} := (\mathtt{com}_{\mathcal{UH}}, \mathtt{unv}_{\mathcal{UH}}, \widetilde{\mathtt{unv}_{\mathcal{UH}}}, \mathtt{com}_{ser}, \mathtt{unv}_{ser}, \mathtt{com}_{lin}, \mathtt{unv}_{lin}, \widetilde{\mathtt{unv}_{lin}}, \mathsf{pk}_{\mathcal{U}}, \mathtt{com}_{id}, \mathtt{unv}_{id}, \widetilde{\mathtt{unv}_{id}}, \sigma)$
- $\Pi_{Val} \leftarrow \mathsf{POK.Prove}(stmt^{Val}, wit^{Val}, L_{\mathrm{B}}^{Val})$, using L_{B}^{Val} from Fig. 29.

Draw share of new serial number:
- $(ser^{\mathrm{new}})^{(\mathcal{U})} \xleftarrow{r} \mathbb{Z}_p$
- $(\mathtt{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})}, \mathtt{unv}_{ser^{\mathrm{new}}}^{(\mathcal{U})}) \leftarrow \mathsf{COM.Com}((ser^{\mathrm{new}})^{(\mathcal{U})})$

$\mathcal{U}\rightarrow\mathcal{O}$: $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, ser, \widetilde{\mathtt{com}_{lin}}, \widetilde{\mathtt{com}_{id}}, \Pi_{Val}, \mathtt{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})})$
$\mathcal{O}$: Check proof and serial number and fetch FP information:
- $stmt^{Val} := (\widetilde{\mathtt{com}_{\mathcal{UH}}}, ser, \widetilde{\mathtt{com}_{lin}}, \widetilde{\mathtt{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
- Check$^{\mathrm{a}}$ $\mathsf{POK.Vfy}(\Pi_{Val}, stmt^{Val}, L_{\mathrm{B}}^{Val}) = 1$
- Check$^{\mathrm{a}}$ $ser \notin \mathsf{L_{SER}}$
- $\mathsf{L_{SER}} := \mathsf{L_{SER}} \cup \{ser\}$
- Set$^{\mathrm{a}}$ $(\mathtt{com}_{fp}, \mathtt{unv}_{fp}, \sigma_{fp}) := f_{\mathrm{FP}}(fp)$.

$\mathcal{U}\leftarrow\mathcal{O}$: $(\mathtt{com}_{fp}, \sigma_{fp})$
$\mathcal{U}$: Check$^{\mathrm{a}}$ that $\mathsf{SIG}'.\mathsf{Vfy}((BK(k), \mathtt{com}_{fp}), \sigma_{fp}, \mathsf{vk}_{\mathcal{T}}) = 1$.
Communicate with $\mathcal{F}_{\mathrm{PPA}}$:
$\mathcal{U}\rightarrow\mathcal{F}_{\mathbf{PPA}}$: $(BK(k), \mathcal{UH}, \widetilde{\mathtt{unv}_{\mathcal{UH}}}, \mathtt{com}_{fp}, in_{\mathcal{U}})$
$\mathcal{O}\rightarrow\mathcal{F}_{\mathbf{PPA}}$: $(BK(k), \widetilde{\mathtt{com}_{\mathcal{UH}}}, fp, \mathtt{unv}_{fp}, in_{\mathcal{O}})$
$\mathcal{U}\leftarrow\mathcal{F}_{\mathbf{PPA}}$: $(\alpha, \mathbf{s}, \mathbf{a}, \mathtt{com}_{\mathbf{a}}, \mathtt{unv}_{\mathbf{a}}, out_{\mathcal{U}})$
$\mathcal{O}\leftarrow\mathcal{F}_{\mathbf{PPA}}$: $(\alpha, \mathbf{s}, \mathtt{com}_{\mathbf{a}}, out_{\mathcal{O}})$
$\mathcal{U}$: Calculate new User History and prove correctness:
- **if** $\alpha \neq \perp \vee \mathbf{s} \neq \perp$ **then**
  - Apply permutation: $\mathcal{UH}' \leftarrow \alpha(\mathcal{UH})$.
  - For i from 0 to $|\mathcal{UH}|-1$, if $\mathbf{s}[i] \neq \perp$, then set $\mathcal{UH}''[i] := \mathbf{s}[i]$, else copy $\mathcal{UH}''[i] := \mathcal{UH}'[i]$.
  - Apply addition: $\mathcal{UH}^{\mathrm{new}} \leftarrow \mathcal{UH}'' + \mathbf{a}$.
  - $(\mathtt{com}'_{\mathcal{UH}}, \mathtt{unv}'_{\mathcal{UH}}) \leftarrow \mathsf{COM.Com}(\mathcal{UH}')$.
  - $(\mathtt{com}''_{\mathcal{UH}}, \mathtt{unv}''_{\mathcal{UH}}) \leftarrow \mathsf{COM.Com}(\mathcal{UH}'')$.
  - $(\mathtt{com}^{\mathrm{new}}_{\mathcal{UH}}, \mathtt{unv}^{\mathrm{new}}_{\mathcal{UH}}) \leftarrow (\mathtt{com}''_{\mathcal{UH}}, \mathtt{unv}''_{\mathcal{UH}}) \oplus (\mathtt{com}_{\mathbf{a}}, \mathtt{unv}_{\mathbf{a}})$.
  - Parse $(uh_0, \ldots, uh_{m-1}) =: \mathcal{UH}$, $(uh'_0, \ldots, uh'_{m-1}) =: \mathcal{UH}'$, and $(uh''_0, \ldots, uh''_{m-1}) =: \mathcal{UH}''$.
  - $stmt^{Tr} := (\widetilde{\mathtt{com}_{\mathcal{UH}}}, \mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \alpha, \mathbf{s})$
  - $wit^{Tr} := (\widetilde{\mathtt{unv}_{\mathcal{UH}}}, \mathtt{unv}'_{\mathcal{UH}}, \mathtt{unv}''_{\mathcal{UH}}, uh_0, \ldots, uh_{m-1}, uh'_0, \ldots, uh'_{m-1}, uh''_0, \ldots, uh''_{m-1})$
  - $\Pi_{Tr} \leftarrow \mathsf{POK.Prove}(stmt^{Tr}, wit^{Tr}, L_{\mathrm{B}}^{Tr})$, using L_{B}^{Tr} from Fig. 30.

  $\mathcal{U}\rightarrow\mathcal{O}$: $(\mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \Pi_{Tr})$
- **else** $(\mathtt{com}^{\mathrm{new}}_{\mathcal{UH}}, \mathtt{unv}^{\mathrm{new}}_{\mathcal{UH}}) \leftarrow (\mathtt{com}''_{\mathcal{UH}}, \mathtt{unv}''_{\mathcal{UH}}) \oplus (\mathtt{com}_{\mathbf{a}}, \mathtt{unv}_{\mathbf{a}})$.

$\mathcal{O}$: Draw share of new serial number:
- $(ser^{\mathrm{new}})^{(\mathcal{O})} \xleftarrow{r} \mathbb{Z}_p$
- $(\mathtt{com}^{(\mathcal{O})}_{ser^{\mathrm{new}}}, \mathtt{unv}^{(\mathcal{O})}_{ser^{\mathrm{new}}}) \leftarrow \mathsf{COM.Com}((ser^{\mathrm{new}})^{(\mathcal{O})})$

Verify proof, compute signature for the new UH.
- **if** $\alpha \neq \perp \vee \mathbf{s} \neq \perp$ **then**
  - $stmt^{Tr} := (\widetilde{\mathtt{com}_{\mathcal{UH}}}, \mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \alpha, \mathbf{s})$
  - Check$^{\mathrm{a}}$ $\mathsf{POK.Vfy}(\Pi_{Tr}, stmt^{Tr}, L_{\mathrm{B}}^{Tr}) = 1$.
  - $\mathtt{com}^{\mathrm{new}}_{\mathcal{UH}} := \mathtt{com}''_{\mathcal{UH}} \oplus \mathtt{com}_{\mathbf{a}}$.
- **else** $\mathtt{com}^{\mathrm{new}}_{\mathcal{UH}} := \widetilde{\mathtt{com}_{\mathcal{UH}}} \oplus \mathtt{com}_{\mathbf{a}}$.
- $\mathtt{com}^{\mathrm{new}}_{ser} := \mathtt{com}^{(\mathcal{O})}_{ser^{\mathrm{new}}} \oplus \mathtt{com}^{(\mathcal{U})}_{ser^{\mathrm{new}}}$
- $\sigma^{\mathrm{new}} \leftarrow \mathsf{SIG.Sgn}(\mathsf{sk}_{\mathcal{O}}, \mathtt{com}^{\mathrm{new}}_{\mathcal{UH}} \| \mathtt{com}^{\mathrm{new}}_{ser} \| \widetilde{\mathtt{com}_{lin}} \| \widetilde{\mathtt{com}_{id}})$

$\mathcal{U}\leftarrow\mathcal{O}$: $(\mathtt{com}^{\mathrm{new}}_{\mathcal{UH}}, (ser^{\mathrm{new}})^{(\mathcal{O})}, \mathtt{com}^{(\mathcal{O})}_{ser^{\mathrm{new}}}, \mathtt{unv}^{(\mathcal{O})}_{ser^{\mathrm{new}}}, \sigma^{\mathrm{new}})$
$\mathcal{U}$: Set$^{\mathrm{a}}$ and store $\lambda^{\mathrm{new}} := \pi_{\mathrm{Verify}}(\mathtt{VfyL}, (\mathcal{UH}^{\mathrm{new}}, \mathtt{com}^{\mathrm{new}}_{\mathcal{UH}}, \mathtt{unv}^{\mathrm{new}}_{\mathcal{UH}}, ((ser^{\mathrm{new}})^{(\mathcal{U})} \oplus (ser^{\mathrm{new}})^{(\mathcal{O})}), (\mathtt{com}^{(\mathcal{U})}_{ser^{\mathrm{new}}} \oplus \mathtt{com}^{(\mathcal{O})}_{ser^{\mathrm{new}}}), (\mathtt{unv}^{(\mathcal{U})}_{ser^{\mathrm{new}}} \oplus \mathtt{unv}^{(\mathcal{O})}_{ser^{\mathrm{new}}}), lin, \widetilde{\mathtt{com}_{lin}}, \widetilde{\mathtt{unv}_{lin}}, id, \widetilde{\mathtt{com}_{id}}, \widetilde{\mathtt{unv}_{id}}, \sigma^{\mathrm{new}}))$.
Output $\mathcal{U}$: User History $\mathcal{UH}^{\mathrm{new}}$, aux. output $out_{\mathcal{U}}$.
Output $\mathcal{O}$: Aux. output $out_{\mathcal{O}}$, permutation α, set vector $\mathbf{s}$.

Fig. 27. The protocol π-*Bookkeeping* for the Bookkeeping-task (Fig. 10).

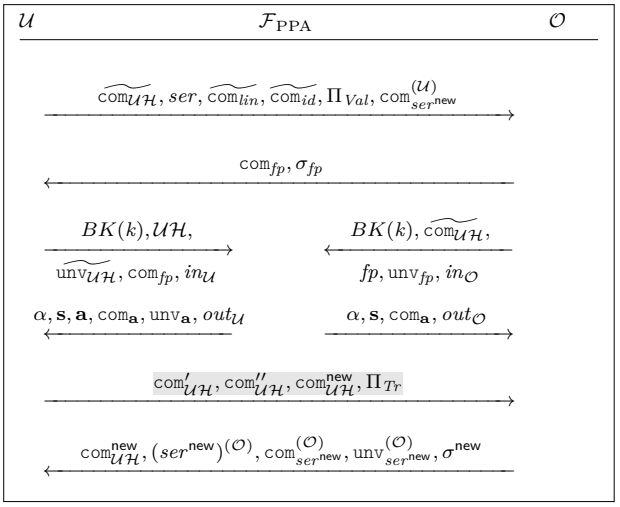

Fig. 28. Message overview for π-*Bookkeeping*$^{(k)}$. The highlighted message is not always sent.

$$
\begin{aligned}
&L_{\text{B}}^{Val}[(\text{com}_{\mathcal{UH}}, \text{unv}_{\mathcal{UH}}, \widetilde{\text{unv}_{\mathcal{UH}}}, \text{com}_{ser}, \text{unv}_{ser}, \text{com}_{lin},\\
&\quad \text{unv}_{lin}, \widetilde{\text{unv}_{lin}}, \text{pk}_{\mathcal{U}}, \text{com}_{id}, \text{unv}_{id}, \widetilde{\text{unv}_{id}}, \sigma):\\
&\quad \text{COM.Unv}((\text{com}_{\mathcal{UH}} \ominus \widetilde{\text{com}_{\mathcal{UH}}}), (\text{unv}_{\mathcal{UH}} \ominus \widetilde{\text{unv}_{\mathcal{UH}}}), \mathbf{0}) = 1 \wedge\\
&\quad \text{COM.Unv}(\text{com}_{ser}, \text{unv}_{ser}, ser) = 1 \wedge\\
&\quad \text{COM.Unv}((\text{com}_{lin} \ominus \widetilde{\text{com}_{lin}}), (\text{unv}_{lin} \ominus \widetilde{\text{unv}_{lin}}), 0) = 1 \wedge\\
&\quad \text{COM.Unv}(\text{com}_{id}, \text{unv}_{id}, \text{pk}_{\mathcal{U}}) = 1 \wedge\\
&\quad \text{COM.Unv}(\widetilde{\text{com}_{id}}, \widetilde{\text{unv}_{id}}, \text{pk}_{\mathcal{U}}) = 1 \wedge\\
&\quad \text{SIG.Vfy}(\text{vk}_{\mathcal{O}}, \sigma, \text{com}_{\mathcal{UH}} \| \text{com}_{ser} \| \text{com}_{lin} \| \text{com}_{id}) = 1]
\end{aligned}
$$

Fig. 29. Language L_{B}^{Val} from π-*Bookkeeping* following the notation of [14].

$$
\begin{aligned}
&L_{\text{B}}^{Tr}[(\widetilde{\text{unv}_{\mathcal{UH}}}, \text{unv}'_{\mathcal{UH}}, \text{unv}''_{\mathcal{UH}}, uh_0, \ldots, uh_{m-1}, uh'_0, \ldots, uh'_{m-1},\\
&\quad uh''_0, \ldots, uh''_{m-1}):\\
&\quad \text{COM.Unv}(\widetilde{\text{com}_{\mathcal{UH}}}, \widetilde{\text{unv}_{\mathcal{UH}}}, (uh_0, \ldots, uh_{m-1})) = 1 \wedge\\
&\quad \text{COM.Unv}(\text{com}'_{\mathcal{UH}}, \text{unv}'_{\mathcal{UH}}, (uh'_0, \ldots, uh'_{m-1})) = 1 \wedge\\
&\quad \text{COM.Unv}(\text{com}''_{\mathcal{UH}}, \text{unv}''_{\mathcal{UH}}, (uh''_0, \ldots, uh''_{m-1})) = 1 \wedge\\
&\quad \text{For } i \text{ from } 0 \text{ to } m-1:\\
&\qquad uh'_i = uh_{\alpha(i)}\\
&\qquad uh''_i := \begin{cases} \mathbf{s}[i] & \text{for } \mathbf{s}[i] \neq \perp \\ uh'_i & \text{for } \mathbf{s}[i] = \perp \end{cases}\\
&\quad]
\end{aligned}
$$

Fig. 30. Language L_{B}^{Tr} used in π-*Bookkeeping* and π-*Update* following the notation of [14].

com_{fp} (which is input by the user) to *fp* (which again is input by the operator). Assuming both unforgeability of signatures and binding of the commitment scheme this ensures that no malicious operator can input uncertified Function Parameters; it either would have to forge a signature on a new commitment without knowing the signing key of $\mathcal{T}$ or use an existing signature but find some opening information and clear text values which open the (previously signed) com_{fp} to a different value.

After computation via $\mathcal{F}_{\text{PPA}}$ the user updates its UH. We consider this tasks use-case to be primarily for bookkeeping and for updating the User History which is why we allow more complex transformations here than in Outsourced Analytics. The user obtains the triple (α, $\mathbf{s}, \mathbf{a}$) from $\mathcal{F}_{\text{PPA}}$, the vectors α and $\mathbf{s}$ are also learned by the operator. This triple contains a permutation α that arbitrarily permutes contents of the User History and can be the identity if no permutation is required, a set vector $\mathbf{s}$ which defines which slots of the User History are set to new values directly and which can contain elements indicating that the old values should be used, and a private add vector $\mathbf{a}$ which defines the additive increment for each value of the User History which can be the neutral element if this entry should not be changed. The first two maps are applied to the elements in the User History directly by the user: First the contents are permuted according to α and then those slots for which an entry in $\mathbf{s}$ exists are updated to their corresponding new values. The user then proves to the operator that it updated its history correctly. For that purpose the language L_{B}^{Tr} from Fig. 29 is used. Note that the operator never learns the actual contents of the new User History. Instead, the operator only learns a commitment on it and the permutation α and the directly updated values $\mathbf{s}$. The proof alongside the commitments on the new User History are sent to the operator. We stress that this step is optional in our protocol and can be skipped if the output of the function only contained trivial a permutation α and set vector $\mathbf{s}$ (that is, one that does not manipulate the User History). In this case both parties directly engage in the computation of the new User History instead of the user proving and the operator verifying that the updates were applied correctly.

The operator homomorphically computes the commitment of the final User History using the commitment on the permuted and updated UH from the user and the commitment on the addition vector $\mathbf{a}$ obtained from $\mathcal{F}_{\text{PPA}}$. The same technique is used to update the serial number homomorphically. This commitment is incorporated into the signature computed by the operator. The operator then sends the information required by the user to create the new logbook to the user. The user updates its logbook as in the User Registration task.

Outsource. The goal of the Outsource task is to *distribute* the data of the user between the operator and a proxy in such a way that (1) both parties know afterwards that the values were shared correctly, and (2) assuming $\mathcal{P}$ and $\mathcal{O}$ do not work together no information on the user data is leaked. Those two guarantees are met by the Robust Secret Sharing (RSS) protocol from Fig. 18 which creates shares of a given user-input such that each party obtains its own additive share and a *commitment* of the *other parties* share. Additionally, each party gets its own *unveil* information for later verification.

The user uses the RSS-protocol π_{Share}-*Share* to create robust shares of the User History $\mathcal{UH}$ which are later used for the computation, the auxiliary input $in_{\mathcal{U}}$, and five One-Time Pads o_α, $\mathsf{o}_\mathbf{s}$, $\mathsf{o}_\mathbf{a}$, o_{OUT} and o_{UNV}. Those are also used as input for the Outsourced Analytics as they mask the following outputs: The first three pads (o_α, $\mathsf{o}_\mathbf{s}$ and $\mathsf{o}_\mathbf{a}$) mask the three outputs relevant for updating the User History (namely the permutation α, the set vector $\mathbf{s}$ and the addition vector $\mathbf{a}$) later from the proxy such that given the output of $\mathcal{F}_{\text{PPA}}$ to $\mathcal{P}$ only the user can reconstruct them. The fourth pad (o_{OUT}) masks the auxiliary output $out_{\mathcal{U}}$.

During Outsourced Analytics the operator only learns a commitment $\mathsf{com}_\mathbf{a}$ of the addition vector $\mathbf{a}$. For an update that works analogous to the Bookkeeping task the user requires the decommitment information $\mathsf{unv}_\mathbf{a}$. Since there are commitment schemes in which access to the unveil information only suffices to reconstruct the input of the commitment,[4] $\mathcal{F}_{\text{PPA}}$ *masks* the unveil information with the final OTP, o_{UNV}.

As in every interaction that requires double-spending detection the user computes a Zero-Knowledge-proof that the User History is valid. The language L_{O} from Fig. 33 used for this task additionally ensures that the secret shares have been created correctly: The first line of the proof ensures that the commitment on the User History for which the user knows the signature from the operator can be homomorphically split into the two values one of which is sent to the operator directly and one of which is sent to the proxy who then sends the commitment to the operator. Both are then used in the statement of L_{O}. The hiding property of the

$\{\pi\text{-}Outsource^{(k)}\}_{k=1}^{K}$:
Input $\mathcal{U}$: Message $OS(k)$, aux. input for the computation $in_{\mathcal{U}}$.
Input $\mathcal{P}$ and $\mathcal{O}$: Message $OS(k)$

$\mathcal{U}$: Create robust secret shares:
- $(\mathsf{o}_{\text{OUT}}, \mathsf{o}_{\text{UNV}}, \mathsf{o}_\alpha, \mathsf{o}_\mathbf{s}, \mathsf{o}_\mathbf{a}, \{\mathsf{sh}^{(x)}_{\mathcal{UH}}, \mathsf{sh}^{(x)}_{in_{\mathcal{U}}}, \mathsf{sh}^{(x)}_{\mathsf{o}_{\text{OUT}}}, \mathsf{sh}^{(x)}_{\mathsf{o}_{\text{UNV}}}, \mathsf{sh}^{(x)}_{\mathsf{o}_\alpha}, \mathsf{sh}^{(x)}_{\mathsf{o}_\mathbf{s}}, \mathsf{sh}^{(x)}_{\mathsf{o}_\mathbf{a}}\}_{x\in\{\mathcal{P},\mathcal{O}\}}) \leftarrow \pi_{\text{Verify}}(\text{ShareL}, \lambda)$

Calculate proof and draw serial number share:
- $(\widetilde{\mathsf{com}_{id}}, \widetilde{\mathsf{unv}_{id}}) \leftarrow \mathsf{COM.Rrnd}(\mathsf{com}_{id}, \mathsf{unv}_{id})$
- $stmt := (\mathsf{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathsf{com}^{(\mathcal{O})}_{\mathcal{UH}}, ser, \widetilde{\mathsf{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
- $wit := (\mathsf{com}_{\mathcal{UH}}, \mathsf{unv}_{\mathcal{UH}}, \mathsf{unv}^{(\mathcal{P})}_{\mathcal{UH}}, \mathsf{com}_{ser}, \mathsf{unv}_{ser}, \mathsf{com}_{lin}, \mathsf{unv}_{lin}, \mathsf{pk}_{\mathcal{U}}, \mathsf{com}_{id}, \mathsf{unv}_{id}, \widetilde{\mathsf{unv}_{id}}, \sigma)$
- $\Pi \leftarrow \mathsf{POK.Prove}(stmt, wit, L_{\text{O}})$, using L_{O} from Fig. 33.
- $(ser^{\text{new}})^{(\mathcal{U})} \xleftarrow{\text{r}} \mathbb{Z}_p$.
- $(\mathsf{com}^{(\mathcal{U})}_{ser^{\text{new}}}, \mathsf{unv}^{(\mathcal{U})}_{ser^{\text{new}}}) \leftarrow \mathsf{COM.Com}((ser^{\text{new}})^{(\mathcal{U})})$

$\mathcal{U}\rightarrow\mathcal{O}$: $(\mathsf{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathsf{sh}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathsf{sh}^{(\mathcal{O})}_{\mathsf{o}_{\text{OUT}}}, \mathsf{sh}^{(\mathcal{O})}_{\mathsf{o}_{\text{UNV}}}, \mathsf{sh}^{(\mathcal{O})}_{\mathsf{o}_\alpha}, \mathsf{sh}^{(\mathcal{O})}_{\mathsf{o}_\mathbf{s}}, \mathsf{sh}^{(\mathcal{O})}_{\mathsf{o}_\mathbf{a}}, ser, \widetilde{\mathsf{com}_{id}}, \Pi, \mathsf{com}^{(\mathcal{U})}_{ser^{\text{new}}})$

$\mathcal{U}\rightarrow\mathcal{P}$: $(\mathsf{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathsf{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathsf{sh}^{(\mathcal{P})}_{\mathsf{o}_{\text{OUT}}}, \mathsf{sh}^{(\mathcal{P})}_{\mathsf{o}_{\text{UNV}}}, \mathsf{sh}^{(\mathcal{P})}_{\mathsf{o}_\alpha}, \mathsf{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{s}}, \mathsf{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{a}})$

$\mathcal{O}$: Check[a] serial number $ser \notin \mathsf{L_{SER}}$ and add ser to $\mathsf{L_{SER}}$.

Prepare linking number: $(lin^{\text{new}})^{(\mathcal{O})} \xleftarrow{\text{r}} \mathbb{Z}_p$
$(\mathsf{com}^{(\mathcal{O})}_{lin^{\text{new}}}, \mathsf{unv}^{(\mathcal{O})}_{lin^{\text{new}}}) \leftarrow \mathsf{Com}((lin^{\text{new}})^{(\mathcal{O})})$

$\mathcal{O}\twoheadrightarrow\mathcal{P}$: $(\mathsf{com}^{(\mathcal{O})}_{lin^{\text{new}}}, \mathsf{com}^{(\mathcal{P})}_{\mathcal{UH}}, \mathsf{com}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathsf{com}^{(\mathcal{P})}_{\mathsf{o}_{\text{OUT}}}, \mathsf{com}^{(\mathcal{P})}_{\mathsf{o}_{\text{UNV}}}, \mathsf{com}^{(\mathcal{P})}_{\mathsf{o}_\alpha}, \mathsf{com}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{s}}, \mathsf{com}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{a}})$

$\mathcal{P}$: Check[a] $\pi_{\text{Verify}}(\texttt{VfyS}, \mathsf{com}^{(\mathcal{P})}_{\mathcal{UH}}, \mathsf{unv}^{(\mathcal{P})}_{\mathcal{UH}}, \mathcal{UH}^{(\mathcal{P})}, \mathsf{com}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathsf{unv}^{\mathcal{P}}_{in_{\mathcal{U}}}, in^{\mathcal{P}}_{\mathcal{U}}, \mathsf{com}^{(\mathcal{P})}_{\mathsf{o}_{\text{OUT}}}, \mathsf{unv}^{\mathcal{P}}_{\mathsf{o}_{\text{OUT}}}, {\mathsf{o}_{\text{OUT}}}^{\mathcal{P}}, \mathsf{com}^{(\mathcal{P})}_{\mathsf{o}_{\text{UNV}}}, \mathsf{unv}^{\mathcal{P}}_{\mathsf{o}_{\text{UNV}}}, {\mathsf{o}_{\text{UNV}}}^{\mathcal{P}}, \mathsf{com}^{(\mathcal{P})}_{\mathsf{o}_\alpha}, \mathsf{unv}^{(\mathcal{P})}_{\mathsf{o}_\alpha}, {\mathsf{o}_\alpha}^{(\mathcal{P})}, \mathsf{com}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{s}}, \mathsf{unv}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{s}}, {\mathsf{o}_\mathbf{s}}^{(\mathcal{P})}, \mathsf{com}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{a}}, \mathsf{unv}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{a}}, {\mathsf{o}_\mathbf{a}}^{(\mathcal{P})}) = 1$

Draw $(lin^{\text{new}})^{(\mathcal{P})} \xleftarrow{\text{r}} \mathbb{Z}_p$

$\mathcal{P}\twoheadrightarrow\mathcal{O}$: $((lin^{\text{new}})^{(\mathcal{P})}, \mathsf{com}^{(\mathcal{O})}_{\mathcal{UH}}, \mathsf{com}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathsf{com}^{(\mathcal{O})}_{\mathsf{o}_{\text{OUT}}}, \mathsf{com}^{(\mathcal{O})}_{\mathsf{o}_{\text{UNV}}}, \mathsf{com}^{(\mathcal{O})}_{\mathsf{o}_\alpha}, \mathsf{com}^{(\mathcal{O})}_{\mathsf{o}_\mathbf{s}}, \mathsf{com}^{(\mathcal{O})}_{\mathsf{o}_\mathbf{a}})$

$\mathcal{O}$: Check proof and sharings
- $stmt := (\mathsf{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathsf{com}^{(\mathcal{O})}_{\mathcal{UH}}, ser, \widetilde{\mathsf{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
- Check[a] $\mathsf{POK.Vfy}(\Pi, stmt, L_{\text{O}}) = 1$
- Check[a] $\pi_{\text{Verify}}(\texttt{VfyS}, \mathsf{com}^{(\mathcal{O})}_{\mathcal{UH}}, \mathsf{unv}^{(\mathcal{O})}_{\mathcal{UH}}, \mathcal{UH}^{(\mathcal{O})}, \mathsf{com}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathsf{unv}^{\mathcal{O}}_{in_{\mathcal{U}}}, in^{\mathcal{O}}_{\mathcal{U}}, \mathsf{com}^{(\mathcal{O})}_{\mathsf{o}_{\text{OUT}}}, \mathsf{unv}^{\mathcal{O}}_{\mathsf{o}_{\text{OUT}}}, {\mathsf{o}_{\text{OUT}}}^{\mathcal{O}}, \mathsf{com}^{(\mathcal{O})}_{\mathsf{o}_{\text{UNV}}}, \mathsf{unv}^{\mathcal{O}}_{\mathsf{o}_{\text{UNV}}}, {\mathsf{o}_{\text{UNV}}}^{\mathcal{O}}, \mathsf{com}^{(\mathcal{O})}_{\mathsf{o}_\alpha}, \mathsf{unv}^{(\mathcal{O})}_{\mathsf{o}_\alpha}, {\mathsf{o}_\alpha}^{(\mathcal{O})}, \mathsf{com}^{(\mathcal{O})}_{\mathsf{o}_\mathbf{s}}, \mathsf{unv}^{(\mathcal{O})}_{\mathsf{o}_\mathbf{s}}, {\mathsf{o}_\mathbf{s}}^{(\mathcal{O})}, \mathsf{com}^{(\mathcal{O})}_{\mathsf{o}_\mathbf{a}}, \mathsf{unv}^{(\mathcal{O})}_{\mathsf{o}_\mathbf{a}}, {\mathsf{o}_\mathbf{a}}^{(\mathcal{O})}) = 1$

Store outsource information and draw share of new serial:
- $lin^{\text{new}} := (lin^{\text{new}})^{(\mathcal{O})} + (lin^{\text{new}})^{(\mathcal{P})}$
- Add $(lin^{\text{new}}, \mathsf{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathsf{sh}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathsf{sh}^{(\mathcal{O})}_{\mathsf{o}_{\text{OUT}}}, \mathsf{sh}^{(\mathcal{O})}_{\mathsf{o}_{\text{UNV}}}, \mathsf{sh}^{(\mathcal{O})}_{\mathsf{o}_\alpha}, \mathsf{sh}^{(\mathcal{O})}_{\mathsf{o}_\mathbf{s}}, \mathsf{sh}^{(\mathcal{O})}_{\mathsf{o}_\mathbf{a}})$ to $f^{(\mathcal{O})}_{\text{OI}}(pid_{\mathcal{P}}, k)$
- $(ser^{\text{new}})^{(\mathcal{O})} \xleftarrow{\text{r}} \mathbb{Z}_p$
- $(\mathsf{com}^{(\mathcal{O})}_{ser^{\text{new}}}, \mathsf{unv}^{(\mathcal{O})}_{ser^{\text{new}}}) \leftarrow \mathsf{COM.Com}((ser^{\text{new}})^{(\mathcal{O})})$

Compute commitments and signature for updated UH:
- $\mathsf{com}^{\text{new}}_{\mathcal{UH}} := \mathsf{com}^{(\mathcal{P})}_{\mathcal{UH}} \oplus \mathsf{com}^{(\mathcal{O})}_{\mathcal{UH}}$
- $\mathsf{com}^{\text{new}}_{ser} := \mathsf{com}^{(\mathcal{U})}_{ser^{\text{new}}} \oplus \mathsf{com}^{(\mathcal{O})}_{ser^{\text{new}}}$
- $(\mathsf{com}^{\text{new}}_{lin}, \mathsf{unv}^{\text{new}}_{lin}) \leftarrow \mathsf{COM.Com}(lin^{\text{new}})$
- $\sigma^{\text{new}} \leftarrow \mathsf{SIG.Sgn}(\mathsf{sk}_{\mathcal{O}}, \mathsf{com}^{\text{new}}_{\mathcal{UH}} \| \mathsf{com}^{\text{new}}_{ser} \| \mathsf{com}^{\text{new}}_{lin} \| \widetilde{\mathsf{com}_{id}})$

$\mathcal{U}\leftarrow\mathcal{O}$: $((ser^{\text{new}})^{(\mathcal{O})}, \mathsf{com}^{(\mathcal{O})}_{ser^{\text{new}}}, \mathsf{unv}^{(\mathcal{O})}_{ser^{\text{new}}}, \mathsf{com}^{\text{new}}_{lin}, \mathsf{unv}^{\text{new}}_{lin}, \sigma^{\text{new}})$

$\mathcal{P}\twoheadleftarrow\mathcal{O}$: $((lin^{\text{new}})^{(\mathcal{O})}, \mathsf{unv}^{(\mathcal{O})}_{lin^{\text{new}}})$

$\mathcal{P}$: Store outsource information:
- Check[a] $\mathsf{Unv}(\mathsf{com}^{(\mathcal{O})}_{lin^{\text{new}}}, \mathsf{unv}^{(\mathcal{O})}_{lin^{\text{new}}}, (lin^{\text{new}})^{(\mathcal{O})}) = 1$
- $lin^{\text{new}} := (lin^{\text{new}})^{(\mathcal{O})} + (lin^{\text{new}})^{(\mathcal{P})}$
- Add $(lin^{\text{new}}, \mathsf{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathsf{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathsf{sh}^{(\mathcal{P})}_{\mathsf{o}_{\text{OUT}}}, \mathsf{sh}^{(\mathcal{P})}_{\mathsf{o}_{\text{UNV}}}, \mathsf{sh}^{(\mathcal{P})}_{\mathsf{o}_\alpha}, \mathsf{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{s}}, \mathsf{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{a}})$ to $f^{(\mathcal{P})}_{\text{OI}}(k)$

$\mathcal{U}\leftarrow\mathcal{P}$: (lin^{new})

$\mathcal{U}$: Set[a] and store $\lambda^{\text{new}} := \pi_{\text{Verify}}(\texttt{VfyL}, (\mathcal{UH}, (\mathsf{com}^{(\mathcal{P})}_{\mathcal{UH}} \oplus \mathsf{com}^{(\mathcal{O})}_{\mathcal{UH}}), (\mathsf{unv}^{(\mathcal{P})}_{\mathcal{UH}} \oplus \mathsf{unv}^{(\mathcal{O})}_{\mathcal{UH}}), ((ser^{\text{new}})^{(\mathcal{U})} + (ser^{\text{new}})^{(\mathcal{O})}), (\mathsf{com}^{(\mathcal{U})}_{ser^{\text{new}}} \oplus \mathsf{com}^{(\mathcal{O})}_{ser^{\text{new}}}), (\mathsf{unv}^{(\mathcal{U})}_{ser^{\text{new}}} \oplus \mathsf{unv}^{(\mathcal{O})}_{ser^{\text{new}}}), lin^{\text{new}}, \mathsf{com}^{\text{new}}_{lin}, \mathsf{unv}^{\text{new}}_{lin}, id, \widetilde{\mathsf{com}_{id}}, \widetilde{\mathsf{unv}_{id}}, \sigma^{\text{new}}))$.

Output $\mathcal{U}$, $\mathcal{P}$ and $\mathcal{O}$: Confirmation ok.

Fig. 31. The protocol π-*Outsource* for the Outsource-task (Fig. 11).

4 Technically, the security definition of any commitment scheme makes no restriction on the amount of information regarding x stored in unv_x. So given any commitment scheme COM where given unv_x only, it is hard to determine x, we can create a commitment scheme COM′ which is equivalent to COM only that COM.Unv sends a tuple (unv_x, x). The new protocol would be as secure as the actual commitment protocol.

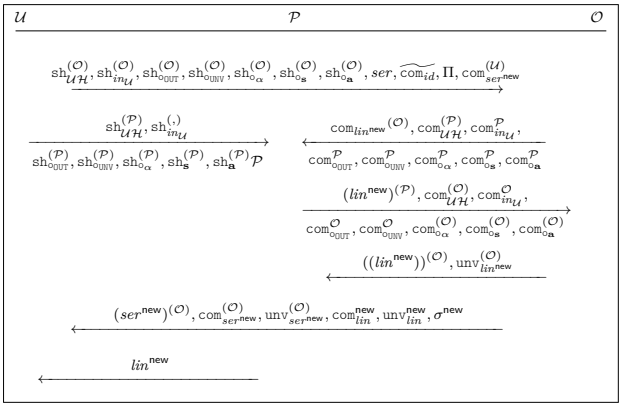

Fig. 32. Message overview for π-*Outsource*.

$$
\begin{aligned}
L_{\mathrm{O}}[&(\mathrm{com}_{\mathcal{UH}}, \mathrm{unv}_{\mathcal{UH}}, \mathrm{unv}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{com}_{ser}, \mathrm{unv}_{ser}, \mathrm{com}_{lin}, \mathrm{unv}_{lin},\\
&\mathsf{pk}_{\mathcal{U}}, \mathrm{com}_{id}, \mathrm{unv}_{id}, \widetilde{\mathrm{unv}_{id}}, \sigma):\\
&\mathsf{COM.Unv}(\mathrm{com}_{\mathcal{UH}} \ominus (\mathrm{com}^{(\mathcal{O})}_{\mathcal{UH}} \oplus \mathrm{com}^{(\mathcal{P})}_{\mathcal{UH}}),\\
&\qquad \mathrm{unv}_{\mathcal{UH}} \ominus (\mathrm{unv}^{(\mathcal{O})}_{\mathcal{UH}} \oplus \mathrm{unv}^{(\mathcal{P})}_{\mathcal{UH}}), \mathbf{0}) = 1 \wedge\\
&\mathsf{COM.Unv}(\mathrm{com}_{ser}, \mathrm{unv}_{ser}, ser) = 1 \wedge\\
&\mathsf{COM.Unv}(\mathrm{com}_{lin}, \mathrm{unv}_{lin}, 0) = 1 \wedge\\
&\mathsf{COM.Unv}(\mathrm{com}_{id}, \mathrm{unv}_{id}, \mathsf{pk}_{\mathcal{U}}) = 1 \wedge\\
&\mathsf{COM.Unv}(\widetilde{\mathrm{com}_{id}}, \widetilde{\mathrm{unv}_{id}}, \mathsf{pk}_{\mathcal{U}}) = 1 \wedge\\
&\mathsf{SIG.Vfy}(\mathsf{vk}_{\mathcal{O}}, \sigma, \mathrm{com}_{\mathcal{UH}} \| \mathrm{com}_{ser} \| \mathrm{com}_{lin} \| \mathrm{com}_{id}) = 1]
\end{aligned}
$$

Fig. 33. Language L_{O} from π-*Outsource* following the notation of [14].

commitment scheme ensures that the operator does not learn the share of the proxy while the fact that the proxy sent the commitment to the operator ensures that this really is the commitment that the user sent; otherwise the user could send some commitment to the operator claiming that it is the proxy-server's share and send a different commitment to the proxy such that the reconstructed User History contains different values than the ones stored in the user's logbook.

Since the user had to unveil the latest serial number of its logbook in order to convince the operator that indeed the latest User History was used the protocol also includes the creation of a new logbook. Note that this logbook then has a non-zero linking number which at the same time prohibits the user from starting a second outsourcing-triplet before finishing the first one and stores information that can be used later by the user to fetch the results during the Update task. Other than that the logbook generation is the same as the one used during the Bookkeeping task.

Outsourced Analytics. The protocol for performing analytical tasks lets both parties fetch the values stored earlier and input them into $\mathcal{F}_{\mathrm{PPA}}$. Again, function specifics are input by the operator as *fp* and the input is verified beforehand using the same basic technique as in the task for Bookkeeping—but this time against the proxy.

After receiving output from $\mathcal{F}_{\mathrm{PPA}}$ both parties store the results for later use.

$\{\pi$-*Outsourced Analytics*$^{(k)}\}_{k=1}^{K}$:
Input $\mathcal{P}$: Message $OA(k)$.
Input $\mathcal{O}$: Message $OA(k)$, FPs *fp*, aux. input $in_{\mathcal{O}}$.

$\mathcal{P}$: Load outsource information:
- Load$^{\mathrm{a}}$ and remove the first Z_k entries $\{(lin, \mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathrm{o_{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathrm{o_{OUNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathrm{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathbf{s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathbf{a}})_z\}_{z=1}^{Z_k}$ from $f^{(\mathcal{P})}_{\mathrm{OI}}(k)$.

$\mathcal{O}$: Load outsource information:
- Load$^{\mathrm{a}}$ and remove the first Z_k entries $\{(lin, \mathrm{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathrm{o_{OUT}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathrm{o_{OUNV}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathrm{o}_\alpha}, \mathrm{sh}^{(\mathcal{O})}_{\mathbf{s}}, \mathrm{sh}^{(\mathcal{O})}_{\mathbf{a}})_z\}_{z=1}^{Z_k}$ from $f^{(\mathcal{O})}_{\mathrm{OI}}(pid_{\mathcal{P}}, k)$.

$\mathcal{O}$: Fetch FP information:
- Set$^{\mathrm{a}}$ $(\mathrm{com}_{fp}, \mathrm{unv}_{fp}, \sigma_{fp}) \coloneqq f_{\mathrm{FP}}(fp)$.

$\mathcal{O} \rightarrow \mathcal{P}$: $(\mathrm{com}_{fp}, \sigma_{fp})$

$\mathcal{P}$: Check$^{\mathrm{a}}$ that $\mathsf{SIG}'.\mathsf{Vfy}((OA(k), \mathrm{com}_{fp}), \sigma_{fp}, \mathsf{vk}_{\mathcal{T}}) = 1$.

$\mathcal{P} \leftrightarrow \mathcal{O}$: Compute function:

$\mathcal{P} \rightarrow \mathcal{F}_{\mathbf{PPA}}$: $(OA(k), \mathrm{com}_{fp}, \{(\mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathrm{o_{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathrm{o_{OUNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathrm{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathbf{s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathbf{a}})_z\}_{z=1}^{Z_k})$

$\mathcal{O} \rightarrow \mathcal{F}_{\mathbf{PPA}}$: $(OA(k), fp, \mathrm{unv}_{fp}, \{(\mathrm{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathrm{o_{OUT}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathrm{o_{OUNV}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathrm{o}_\alpha}, \mathrm{sh}^{(\mathcal{O})}_{\mathbf{s}}, \mathrm{sh}^{(\mathcal{O})}_{\mathbf{a}})_z\}_{z=1}^{Z_k}, in_{\mathcal{O}})$

$\mathcal{P} \leftarrow \mathcal{F}_{\mathbf{PPA}}$: $\{(\mathrm{c}_{\alpha_z}, \mathrm{c}_{\mathbf{s}_z}, \mathrm{c}_{\mathbf{a}_z}, \mathrm{c}_{\mathrm{unv}_{\mathbf{a}_z}}, \mathrm{c}_{out_{\mathcal{U}_z}})\}_{z=1}^{Z_k}$

$\mathcal{O} \leftarrow \mathcal{F}_{\mathbf{PPA}}$: $\{(\alpha, \mathbf{s}, \mathrm{com}_{\mathbf{a}_z}, \mathrm{c}_{out_{\mathcal{U}_z}})\}_{z=1}^{Z_k}, out_{\mathcal{O}}$

$\mathcal{O}$: Store update information:
For all $z \in (1, \ldots, Z_k)$: set $f^{(\mathcal{O})}_{\mathrm{UP}}(lin_z) \coloneqq (\alpha, \mathbf{s}, \mathrm{com}_{\mathbf{a}}, \mathrm{c}_{out_{\mathcal{U}}})_z$.

$\mathcal{P}$: Store update information:
For all $z \in (1, \ldots, Z_k)$: set $f^{(\mathcal{P})}_{\mathrm{UP}}(lin_z) \coloneqq (\mathrm{c}_\alpha, \mathrm{c}_{\mathbf{s}}, \mathrm{c}_{\mathbf{a}}, \mathrm{c}_{\mathrm{unv}_{\mathbf{a}}}, \mathrm{c}_{out_{\mathcal{U}}})_z$

Output $\mathcal{P}$: Confirmation ok.
Output $\mathcal{O}$: Aux. output $out_{\mathcal{O}}$.

Fig. 34. The protocol π-*Outsourced Analytics* for the Outsourced Analytics-task (Fig. 12).

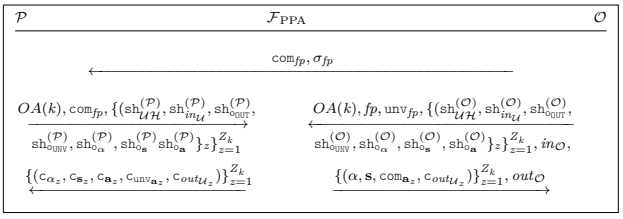

Fig. 35. Message overview for π-*Outsourced Analytics*$^{(k)}$.

Update. The Update protocol contains two steps which do not necessarily have to be executed at once. In the first step the user only requests the data $\mathcal{P}$ has stored by sending the linking number. In the second step the user requests the data from $\mathcal{O}$ and additionally computes a new logbook and proves that the data in there has been applied correctly. This proof employs the language from Fig. 38 which is reminiscent of the proof used during π-*Bookkeeping*, but also proves that the *linking number* is correct. Additionally, in case of a non-trivial permutation or direct update the user uses the language L_{B}^{Tr} in order to prove that the permutation was applied correctly.

π-*Update*:
Input $\mathcal{U}$: Upd
Input $\mathcal{P}$: Upd
Input $\mathcal{O}$: Upd
$\mathcal{U}\rightarrow\mathcal{P}$: (lin)
  $\mathcal{P}$: Set$^{\mathrm{a}}$ and remove $(\mathsf{c}_\alpha, \mathsf{c}_{\mathbf{s}}, \mathsf{c}_{\mathbf{a}}, \mathsf{c}_{\mathsf{unv}_{\mathbf{a}}}, \mathsf{c}_{out_{\mathcal{U}}}) := f_{\mathrm{UP}}^{(\mathcal{P})}(lin)$
$\mathcal{U}\leftarrow\mathcal{P}$: $(\mathsf{c}_\alpha, \mathsf{c}_{\mathbf{s}}, \mathsf{c}_{\mathbf{a}}, \mathsf{c}_{\mathsf{unv}_{\mathbf{a}}}, \mathsf{c}_{out_{\mathcal{U}}})$
  $\mathcal{U}$: Prove lb validity and correct updates and draw share of serial:
    – $(\widetilde{\mathsf{com}_{\mathcal{UH}}}, \widetilde{\mathsf{unv}_{\mathcal{UH}}}) \leftarrow \mathsf{COM.Rrnd}(\mathsf{com}_{\mathcal{UH}}, \mathsf{unv}_{\mathcal{UH}})$
    – Set $\alpha := \mathsf{c}_\alpha - \mathsf{o}_\alpha$, $\mathbf{s} := \mathsf{c}_{\mathbf{s}} - \mathsf{o}_{\mathbf{s}}$ and $\mathbf{a} := \mathsf{c}_{\mathbf{a}} - \mathsf{o}_{\mathbf{a}}$.
    – Set $\mathsf{unv}_{\mathbf{a}} := \mathsf{c}_{\mathsf{unv}_{\mathbf{a}}} - \mathsf{o}_{\mathsf{UNV}}$ and $out_{\mathcal{U}} := \mathsf{c}_{out_{\mathcal{U}}} - \mathsf{o}_{\mathsf{OUT}}$
    – $(\widetilde{\mathsf{com}_{id}}, \widetilde{\mathsf{unv}_{id}}) \leftarrow \mathsf{COM.Rrnd}(\mathsf{com}_{id}, \mathsf{unv}_{id})$
    – $stmt := (\widetilde{\mathsf{com}_{\mathcal{UH}}}, ser, lin, \widetilde{\mathsf{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
    – $wit := (\mathsf{com}_{\mathcal{UH}}, \mathsf{unv}_{\mathcal{UH}}, \widetilde{\mathsf{unv}_{\mathcal{UH}}}, \mathsf{com}_{ser}, \mathsf{unv}_{ser}, \mathsf{com}_{lin}, \mathsf{unv}_{lin}, \mathsf{pk}_{\mathcal{U}}, \mathsf{com}_{id}, \mathsf{unv}_{id}, \widetilde{\mathsf{unv}_{id}}, \sigma)$
    – $\Pi \leftarrow \mathsf{POK.Prove}(stmt, wit, L_{\mathrm{U}})$, using L_{U} from Fig. 38.
    – $(ser^{\mathrm{new}})^{(\mathcal{U})} \xleftarrow{\mathrm{r}} \mathbb{Z}_p$
    – $(\mathsf{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})}, \mathsf{unv}_{ser^{\mathrm{new}}}^{(\mathcal{U})}) \leftarrow \mathsf{COM.Com}((ser^{\mathrm{new}})^{(\mathcal{U})})$
    – **if** $\alpha \neq \perp \wedge \mathbf{s} \neq \perp$ **then**
      * Apply permutation: $\mathcal{UH}' \leftarrow \alpha(\widetilde{\mathcal{UH}})$.
      * For i from 0 to $|\mathcal{UH}| - 1$, if $\mathbf{s}[i] \neq \perp$, then set value $\mathcal{UH}''[i] := \mathbf{s}[i]$, else copy $\mathcal{UH}''[i] := \mathcal{UH}'[i]$.
      * Apply addition: $\mathcal{UH}^{\mathrm{new}} \leftarrow \mathcal{UH}'' + \mathbf{a}$.
      * $(\mathsf{com}'_{\mathcal{UH}}, \mathsf{unv}'_{\mathcal{UH}}) \leftarrow \mathsf{COM.Com}(\mathcal{UH}')$.
      * $(\mathsf{com}''_{\mathcal{UH}}, \mathsf{unv}''_{\mathcal{UH}}) \leftarrow \mathsf{COM.Com}(\mathcal{UH}'')$.
      * $stmt^{Tr} := (\widetilde{\mathsf{com}_{\mathcal{UH}}}, \mathsf{com}'_{\mathcal{UH}}, \mathsf{com}''_{\mathcal{UH}}, \alpha, \mathbf{s})$
      * $wit^{Tr} := (\widetilde{\mathsf{unv}_{\mathcal{UH}}}, \mathsf{unv}'_{\mathcal{UH}}, \mathsf{unv}''_{\mathcal{UH}}, uh_0, \ldots, uh_{m-1}, uh'_0, \ldots, uh'_{m-1}, uh''_0, \ldots, uh''_{m-1})$
      * $\Pi_{Tr} \leftarrow \mathsf{POK.Prove}(stmt^{Tr}, wit^{Tr}, L_{\mathrm{B}}^{Tr})$, using L_{B}^{Tr} from Fig. 30.
      $\mathcal{U}\rightarrow\mathcal{O}$: $(\widetilde{\mathsf{com}_{\mathcal{UH}}}, \mathsf{com}'_{\mathcal{UH}}, \mathsf{com}''_{\mathcal{UH}}, \widetilde{\mathsf{com}_{id}}, ser, lin, \Pi, \Pi_{Tr}, \mathsf{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})})$
    – **else**
      * Apply addition: $\mathcal{UH}^{\mathrm{new}} \leftarrow \mathcal{UH} + \mathbf{a}$.
      $\mathcal{U}\rightarrow\mathcal{O}$: $(\widetilde{\mathsf{com}_{\mathcal{UH}}}, \widetilde{\mathsf{com}_{id}}, ser, lin, \Pi, \mathsf{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})})$
  $\mathcal{O}$: Load UI and check proof and serial:
    – Set$^{\mathrm{a}}$ $(\alpha, \mathbf{s}, \mathsf{com}_{\mathbf{a}}, (\mathsf{c}_{out_{\mathcal{U}}})') := f_{\mathrm{UP}}^{(\mathcal{O})}(lin)$
    – Remove $lin \mapsto (\alpha, \mathbf{s}, \mathsf{com}_{\mathbf{a}}, (\mathsf{c}_{out_{\mathcal{U}}})')$ from $f_{\mathrm{UP}}^{(\mathcal{O})}$
    – $stmt := (\widetilde{\mathsf{com}_{\mathcal{UH}}}, ser, lin, \widetilde{\mathsf{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
    – Check$^{\mathrm{a}}$ $\mathsf{POK.Vfy}(\Pi, stmt, L_{\mathrm{U}}) \stackrel{?}{=} 1$
    – **if** $\alpha \neq \perp \vee \mathbf{s} \neq \perp$ **then**
      * $stmt^{Tr} := (\widetilde{\mathsf{com}_{\mathcal{UH}}}, \mathsf{com}'_{\mathcal{UH}}, \mathsf{com}''_{\mathcal{UH}}, \alpha, \mathbf{s})$
      * Check$^{\mathrm{a}}$ $\mathsf{POK.Vfy}(\Pi_{Tr}, stmt^{Tr}, L_{\mathrm{B}}^{Tr}) \stackrel{?}{=} 1$
      * $\mathsf{com}_{\mathcal{UH}}^{\mathrm{new}} := \mathsf{com}''_{\mathcal{UH}} \oplus \mathsf{com}_{\mathbf{a}}$
    – **else**
      * $\mathsf{com}_{\mathcal{UH}}^{\mathrm{new}} := \widetilde{\mathsf{com}_{\mathcal{UH}}} \oplus \mathsf{com}_{\mathbf{a}}$
    – Check$^{\mathrm{a}}$ $ser \notin \mathsf{L_{SER}}$
    – $\mathsf{L_{SER}} := \mathsf{L_{SER}} \cup \{ser\}$
    Prepare the new UH:
    – $(ser^{\mathrm{new}})^{(\mathcal{O})} \xleftarrow{\mathrm{r}} \mathbb{Z}_p$
    – $(\mathsf{com}_{ser^{\mathrm{new}}}^{(\mathcal{O})}, \mathsf{unv}_{ser^{\mathrm{new}}}^{(\mathcal{O})}) \leftarrow \mathsf{COM.Com}((ser^{\mathrm{new}})^{(\mathcal{O})})$
    – $\mathsf{com}_{ser}^{\mathrm{new}} := \mathsf{com}_{ser^{\mathrm{new}}}^{(\mathcal{O})} \oplus \mathsf{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})}$
    – $(\mathsf{com}_{lin}^{\mathrm{new}}, \mathsf{unv}_{lin}^{\mathrm{new}}) \leftarrow \mathsf{COM.Com}(0)$
    – $\sigma^{\mathrm{new}} \leftarrow \mathsf{SIG.Sgn}(\mathsf{sk}_{\mathcal{O}}, \mathsf{com}_{\mathcal{UH}}^{\mathrm{new}} \| \mathsf{com}_{ser}^{\mathrm{new}} \| \mathsf{com}_{lin}^{\mathrm{new}} \| \widetilde{\mathsf{com}_{id}})$
$\mathcal{U}\leftarrow\mathcal{O}$: $(\mathsf{com}_{\mathbf{a}}, \mathsf{com}_{\mathcal{UH}}^{\mathrm{new}}, (ser^{\mathrm{new}})^{(\mathcal{O})}, \mathsf{com}_{ser^{\mathrm{new}}}^{(\mathcal{O})}, \mathsf{unv}_{ser^{\mathrm{new}}}^{(\mathcal{O})}, \mathsf{com}_{lin}^{\mathrm{new}}, \mathsf{unv}_{lin}^{\mathrm{new}}, \sigma^{\mathrm{new}}, \mathsf{c}_{out_{\mathcal{U}}})$
  $\mathcal{U}$: Check$^{\mathrm{a}}$ $\mathsf{Unv}(\mathsf{com}_{\mathbf{a}}, \mathsf{unv}_{\mathbf{a}}, \mathbf{a}) \stackrel{?}{=} 1$ and$^{\mathrm{a}}$ $\mathsf{c}_{out_{\mathcal{U}}}{}' \stackrel{?}{=} \mathsf{c}_{out_{\mathcal{U}}}$ Set$^{\mathrm{a}}$ and store $\lambda^{\mathrm{new}} := \pi_{\mathrm{Verify}}(\mathtt{VfyL}, (\mathcal{UH}^{\mathrm{new}}, \mathsf{com}_{\mathcal{UH}}^{\mathrm{new}}, (\widetilde{\mathsf{unv}_{\mathcal{UH}}} \oplus \mathsf{unv}_{\mathbf{a}}), ((ser^{\mathrm{new}})^{(\mathcal{U})} + (ser^{\mathrm{new}})^{(\mathcal{O})}), (\mathsf{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})} \oplus \mathsf{com}_{ser^{\mathrm{new}}}^{(\mathcal{O})}), (\mathsf{unv}_{ser^{\mathrm{new}}}^{(\mathcal{U})} \oplus \mathsf{unv}_{ser^{\mathrm{new}}}^{(\mathcal{O})}), 0, \mathsf{com}_{lin}^{\mathrm{new}}, \mathsf{unv}_{lin}^{\mathrm{new}}, id, \widetilde{\mathsf{com}_{id}}, \widetilde{\mathsf{unv}_{id}}, \sigma^{\mathrm{new}}))$
Output $\mathcal{U}$: Permutation α, set vector $\mathbf{s}$, add vector $\mathbf{a}$, aux. $out_{\mathcal{U}}$.
Output $\mathcal{P}$: Confirmation ok.
Output $\mathcal{O}$: Permutation α, set vector $\mathbf{s}$.

Fig. 36. The protocol π-*Update* for the Update-task (Fig. 13).

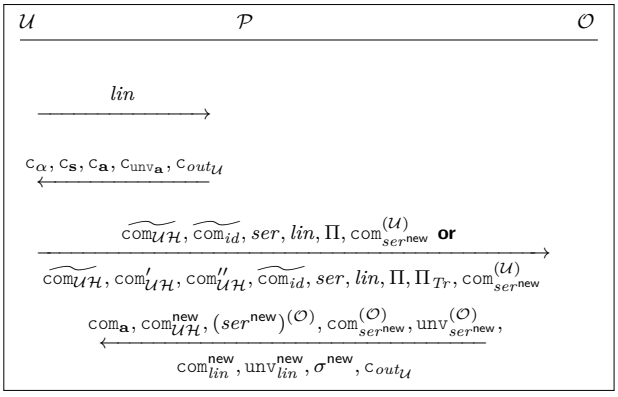

Fig. 37. Message overview for π-*Update*.

$L_{\mathsf{U}}[(\mathrm{com}_{\mathcal{UH}}, \mathrm{unv}_{\mathcal{UH}}, \widetilde{\mathrm{unv}_{\mathcal{UH}}}, \mathrm{com}_{ser}, \mathrm{unv}_{ser}, \mathrm{com}_{lin}, \mathrm{unv}_{lin}, \mathsf{pk}_\mathcal{U}, \mathrm{com}_{id}, \mathrm{unv}_{id}, \widetilde{\mathrm{unv}_{id}}, \sigma)$:
$\mathsf{COM.Unv}((\mathrm{com}_{\mathcal{UH}} \ominus \widetilde{\mathrm{com}_{\mathcal{UH}}}), (\mathrm{unv}_{\mathcal{UH}} \ominus \widetilde{\mathrm{unv}_{\mathcal{UH}}}), \mathbf{0}) = 1 \wedge$
$\mathsf{COM.Unv}(\mathrm{com}_{ser}, \mathrm{unv}_{ser}, ser) = 1 \wedge$
$\mathsf{COM.Unv}(\mathrm{com}_{lin}, \mathrm{unv}_{lin}, lin) = 1 \wedge$
$\mathsf{COM.Unv}(\mathrm{com}_{id}, \mathrm{unv}_{id}, \mathsf{pk}_\mathcal{U}) = 1 \wedge$
$\mathsf{COM.Unv}(\widetilde{\mathrm{com}_{id}}, \widetilde{\mathrm{unv}_{id}}, \mathsf{pk}_\mathcal{U}) = 1 \wedge$
$\mathsf{SIG.Vfy}(\mathsf{vk}_\mathcal{O}, \sigma, \mathrm{com}_{\mathcal{UH}} \| \mathrm{com}_{ser} \| \mathrm{com}_{lin} \| \mathrm{com}_{id}) = 1]$

Fig. 38. Language L_{U} from π-*Update* following the notation of [14].

F Applications

Here we sketch two privacy-preserving applications for PUBA, namely fraud detection for mobile payments and a targeted advertising network.

F.1 Fraud Detection for Mobile Payments

The scenario was already introduced in Section 4, therefore we only recap it shortly here. We construct a mobile payment service that supports anonymous payments and is equipped with strong fraud detection mechanisms. We consider a two-tier fraud detection mechanism: At each transaction at a point-of -sale a lightweight rule-based fraud detection mechanism is executed and a more complex machine learning-based fraud detection mechanism is executed with the operator after some threshold of payments has been reached.

Each time the user conducts a payment at a point-of-sale, a new transaction is created and stored in the UH. We assume that a transaction record t consists of the following data encoded as vector of $\mathbb{Z}_p$ elements:

$$t \coloneqq (\mathsf{acc}, \mathsf{ts}, \mathsf{loc}, \mathsf{type}, \mathsf{tval}),$$

where acc is one iff the transaction has been accepted, ts is a timestamp, loc indicates the geographic location the transaction took place, type describes the type of shop (e.g., grocery store, jewelry store, etc.), and tval is the transaction value. We stress that this is only an example, adding other attributes is pretty straightforward.

The UH contains the latest T transaction records, the user's balance *bal* and some important additional information to support the fraud detection mechanisms. These additional information include the account's current risk level *rsk*, a maximum value *max* a single transaction can have, and a limit *rem* on the number of payment transactions a user can perform before the complex fraud detection mechanism has to be run. Note that the latter two values depend on the current risk level.

Thus, the UH has the following form:

$$\mathcal{UH} \coloneqq (t_1^1, \ldots, t_5^1, \ \ldots \ , t_1^T, \ldots, t_5^T, rsk, rem, max, bal)$$

The first $5T$ slots store the last T transactions (each transaction t^i requires 5 slots), t^1 is the most recent transaction. The UH has a total length of $5T+4$ entries.

We now describe the individual tasks a user can perform. The details of the function $\Delta_{\mathrm{mpayment}}$ can be found in Fig. 39.

Registration. During registration the user obtains a UH with an empty balance and no stored transactions using the User Registration task, which calls the function $\Delta_{\mathrm{mpayment}}$ with input message *UReg*. Both parties input the initial values for the risk level, the remaining number of transactions and the maximum transaction value, i.e., (rsk, rem, max). The values *rem* and *max* can either be fixed constants or depend on the risk level. Alongside empty transactions, those are written into the new UH.

Top-Up. Our mobile payments service is prepaid-based; it is important that a user can top-up its balance. We propose a general method where user and bank agree on the amount that should be topped up and leave the actual transfer of money to the implementation method, such as anonymously depositing money at an ATM or making a transfer from the normal bank account (which would be identifying). The user invokes a Bookkeeping task where both parties input the amount to be deposited, which is added to the user's balance. In our example, a top-up transaction is not recorded in the user's transaction history (although this might be reasonable).

Δ_{mpayment}

Registration:
$(UReg, fp \coloneqq \bot, in_{\mathcal{U}} \coloneqq (rsk, rem, max), in_{\mathcal{O}} \coloneqq (rsk, rem, max))$
If $in_{\mathcal{U}} \neq in_{\mathcal{O}}$, abort.
Set $\mathcal{UH} \coloneqq (\bot, \ldots, \bot, rsk, rem, max, 0)$.
Return $(\mathcal{UH}, out_{\mathcal{U}} \coloneqq \mathsf{ok}, out_{\mathcal{O}} \coloneqq \mathsf{ok})$.

Top-Up:
$(BK, fp \coloneqq \bot, k \coloneqq 1, \mathcal{UH}, in_{\mathcal{U}} \coloneqq \mathsf{val}, in_{\mathcal{O}} \coloneqq \mathsf{val})$
If $in_{\mathcal{U}} \neq in_{\mathcal{O}}$, abort.
$\mathbf{a} \coloneqq (0, \ldots, 0, \mathsf{val})$.
Return $(\alpha \coloneqq \left(\begin{smallmatrix} 0 & \cdots & m-1 \\ 0 & \cdots & m-1 \end{smallmatrix}\right), \mathbf{s} \coloneqq (\bot, \ldots, \bot), \mathbf{a}, out_{\mathcal{U}} \coloneqq \mathsf{ok}, out_{\mathcal{O}} \coloneqq \mathsf{ok})$.

Payment:
$(BK, fp \coloneqq \bot, k \coloneqq 2, \mathcal{UH}, in_{\mathcal{U}} \coloneqq (\mathsf{ts}, \mathsf{loc}, \mathsf{type}, \mathsf{tval}), in_{\mathcal{O}} \coloneqq (\mathsf{ts}, \mathsf{loc}, \mathsf{type}, \mathsf{tval}))$
If $in_{\mathcal{U}} \neq in_{\mathcal{O}}$, abort.
Parse $(t_1^1, \ldots, t_5^1, \ \ldots \ , t_1^T, \ldots, t_5^T, rsk, rem, max, bal) \coloneqq \mathcal{UH}$.
If $bal < \mathsf{tval}$,
  then return $\big(\alpha \coloneqq \left(\begin{smallmatrix} 0 & \cdots & m-1 \\ 0 & \cdots & m-1 \end{smallmatrix}\right), \mathbf{s} \coloneqq (\bot, \ldots, \bot), \mathbf{a} \coloneqq (0, \ldots, 0),$
  $out_{\mathcal{U}} \coloneqq \bot, out_{\mathcal{O}} \coloneqq \bot\big)$.
$\mathsf{acc} \coloneqq f_{\text{fdsimple}}(\mathcal{UH}, (\mathsf{ts}, \mathsf{loc}, \mathsf{type}, \mathsf{tval}))$.
$(t_1^{\text{new}}, \ldots, t_5^{\text{new}}) \coloneqq (\mathsf{acc}, \mathsf{ts}, \mathsf{loc}, \mathsf{type}, \mathsf{tval})$.
If $\mathsf{acc} = 1$,
  then $bal^{\text{diff}} \coloneqq \mathsf{tval}$,
  else $bal^{\text{diff}} \coloneqq 0$.
$\alpha \coloneqq \left(\begin{smallmatrix} 0 & \cdots & 4 & 5 \ldots 9 & \cdots & 5T-5 \ \ldots \ 5T-1 & 5T \ \ldots \ 5T+3 \\ 5T-5 & \ldots & 5T-1 & 0 \ldots 4 & \cdots & 5T-10 \ \ldots \ 5T-6 & 5T \ \ldots \ 5T+3 \end{smallmatrix}\right)$.
$\mathbf{s} \coloneqq (t_1^{\text{new}}, \ldots, t_5^{\text{new}}, \bot, \ldots, \bot)$.
$\mathbf{a} \coloneqq (0, \ldots, 0, 0, -1, 0, -bal^{\text{diff}})$.
Return $\big(\alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}} \coloneqq \mathsf{acc}, out_{\mathcal{O}} \coloneqq \mathsf{acc}\big)$.

Risk Calculation:
$(OA, fp, k \coloneqq 3, (\mathcal{UH}, in_{\mathcal{U}} \coloneqq \bot), in_{\mathcal{O}} \coloneqq \bot)$
Parse $(t_1^1, \ldots, t_5^1, \ \ldots \ , t_1^T, \ldots, t_5^T, rsk, rem, max, bal) \coloneqq \mathcal{UH}$.
$(rsk^{\text{new}}, rem^{\text{new}}, max^{\text{new}}) \coloneqq f_{\text{fdcomp}}(fp, \mathcal{UH})$.
$\mathbf{s} \coloneqq (\bot, \ldots, \bot, rsk^{\text{new}}, \bot, max^{\text{new}}, \bot)$.
$\mathbf{a} \coloneqq (0, \ldots, 0, 0, -rem + rem^{\text{new}}, 0, 0)$.
Return $((\alpha \coloneqq \left(\begin{smallmatrix} 0 & \cdots & m-1 \\ 0 & \cdots & m-1 \end{smallmatrix}\right), \mathbf{s}, \mathbf{a}, out_{\mathcal{U}} \coloneqq \mathsf{ok}), out_{\mathcal{O}} \coloneqq \mathsf{ok})$.

Fig. 39. Instantiation of Δ for privacy-preserving mobile payments with fraud detection.

Payment (with simple fraud detection). To issue a payment, a user communicates with a point-of-sale that has a communication channel with the bank and forwards all messages between the user and the bank. The payment is done via the Bookkeeping task where both parties input all transaction details excluding the acceptance bit, i.e., timestamp, location, type of shop and transaction value. The function aborts if the transaction value exceeds the account's balance. Otherwise, the simple rule-based fraud detection mechanism f_{fdsimple} is executed to decide whether the transaction is accepted or not based on the last T transactions, the risk level rsk, the remaining number of transactions rem, the maximum transaction value max, and the details of the current transaction. In our example implementation in Section 5.4 we verify that the following conditions are all satisfied: 1. $\mathsf{tval} \leq max$ (the transaction value does not exceed the allowed amount) 2. $rem > 0$ (the number of payment transactions the user can perform before the complex fraud detection mechanism has to be run has not exceeded its limit). Of course, additional checks could be included: The transaction could be denied if the risk level is medium but there are more than three transactions within a 10 minute period, or if two consecutive transactions differ in their location so much that no user could have possibly traveled that far in such a short time period.

This simple mechanism already provides some fraud protection, but is lightweight enough to be computed by the user's resource-constrained device. As those can be public, the user can evaluate the rule-based fraud detection by itself and provide the point-of-sale with a ZK proof that it evaluated the mechanism correctly based on its logbook. This can significantly speed up the payment process compared to an MPC-based computation, assuming the rules are simple and efficiently compatible with the ZK proof system.

Only if the transaction is accepted it is physically executed and the user's balance gets updated accordingly. More specifically, the three output maps of Δ look as follows: The permutation shifts all past transactions to the right to make room for the new transaction, which is written into the UH with the direct update. This is done because the UH only stores the last T transactions to hide the total number of transactions. The additive increment then subtracts one from the number of remaining transactions rem and subtracts the transaction value tval from the balance bal iff the transaction was accepted. The risk level and the maximum transaction value stay the same.

More precisely, the new transaction is assembled as $(t_1^{\text{new}}, \ldots, t_5^{\text{new}}) = (\mathsf{acc}, \mathsf{ts}, \mathsf{loc}, \mathsf{type}, \mathsf{tval})$ and the new UH then looks as follows after applying the three maps: $\mathcal{UH}^{\text{new}} \coloneqq (t_1^{\text{new}}, \ldots, t_5^{\text{new}}, t_1^1, \ldots, t_5^1, \ \ldots \ , t_1^{T-1}, \ldots, t_5^{T-1}, rsk, rem - 1, max, bal^{\text{new}})$.

Risk Calculation (with complex fraud detection). Each time the user executes a payment transaction, the counter for the number of remaining transactions decreases by one. When this counter reaches zero, it forces the user to participate in this task, where the risk level gets updated and a more sophisticated fraud detection algorithm is executed. By choosing a suitable value for the initial value of that counter, we can ensure that users regularly participate in the complex fraud detection mechanism. As we assume this complex fraud detection mechanism to be based on machine learning, this might result in considerable computational effort.

Therefore, the Outsourced Analytics (OA) task is used. The operator inputs its FPs *fp* into OA. Note that, as usually, these FPs were verified by the TSA to pose no privacy-risk for the user and yet are not learned by the user. The fraud detection mechanism then computes the user's new risk level *rsk* along with a new maximum number of transactions *rem*, and maximum transaction value *max*. These new values are then stored in the UH. More specifically, only the direct update and additive increment are needed. The direct update sets the new values for the risk level and the new maximum transaction value at the corresponding slots and overwrites the old values in the process. Since the outsourcing triple is non-blocking regarding Bookkeeping operations, we have to take into account that the value of the remaining number of transaction may have changed since Outsource was called. Therefore, the additive increment adds the difference between the old remaining number of transactions (from the point when Outsource was called) and the new value to the corresponding slot.

F.2 Targeted Advertising System

We now briefly sketch a targeted mobile advertising system which can optionally be used as an extension for loyalty systems. For their cooperation, users are rewarded with vouchers targeted at their purchase behavior. The central idea is that the user's purchases are stored in the UH. From time to time, users submit their purchase history to the operator, who analyzes it together with the histories of several other users. The user is rewarded with a voucher targeted at the user's probable interests and is displayed alongside a suitable ad in the smartphone app.

We now assume that the operator acts as a conglomeration of supermarket chains and further participating shops. In the following, we describe how to use PUBA in this scenario.

Registration. Upon registration with the User Registration task, the user obtains an empty UH. Each slot in the UH represents a product category, e.g., "vegetables", "candy", or "fast food". The UH tracks the amount of money spent in each category.

Checkout. When purchasing goods at a participating store, the user updates the purchase history using the Bookkeeping task. The amount of money spent in each category is calculated and added to the corresponding slots in the UH.

Analytics. The Outsourced Analytics task lets the user provide data for analytical purposes. We assume the operator has an analytical function (for example for marketing analyses) which takes some FPs and multiple UHs as input and assigns to each UH a class that describes the most likely interests of the corresponding user. The user is rewarded with a voucher that matches this class and that can be redeemed at a participating shop. Additionally, the user gets a matching advertisement. For example, if the analysis reveals that the user likes chocolate, the user obtains advertisements for a new kind of chocolate and a voucher for a 10% discount on chocolate. The UHs of the participating users remain unchanged.

If the targeted advertising system is interconnected with a loyalty system, the user could also earn loyalty points instead of vouchers.

G Security

In this section, we prove the security of our system. That is, we show that the protocol π_{PUBA} is *at least* as secure, as our Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$, without relying on a trusted party to execute $\mathcal{F}_{PUBA}^{(\Delta)}$ on all parties inputs. To that end, we provide a simulator that simulates the protocol messages of honest parties without knowing the parties secret input, and prove that those simulated messages cannot be differentiated by any efficient environment $\mathcal{Z}$.

For technical reasons, we have to restrict our adversary to corrupting only *either* the proxy $\mathcal{P}$, or the operator $\mathcal{O}$. We split our simulator up in two parts. Appendix G.1 contains the simulator for all corruption scenarios related to the security of an honest user $\mathcal{U}$, even in the presence of other malicious users. Appendix G.2 contains the simulator for all corruption scenarios regarding the security of an honest operator $\mathcal{O}$. Combined, those two simulators cover all corruption scenarios, in which either $\mathcal{P}$ or $\mathcal{O}$ are honest.

G.1 User Security

In this section, we investigate the security of our system in scenarios that relate to the security of an honest user $\mathcal{U}$. To that end, we prove the following theorem:

Theorem G.1 (User Security)**.** *If instantiated with a trapdoor-commitment scheme* COM *and a dual-mode*

zero-knowledge protocol POK*, it holds that*

$$\pi_{PUBA}^{(\mathcal{F}_{\text{PPA}},\mathcal{F}_{\text{BB}},\mathcal{F}_{\text{KE}},\mathcal{F}_{\text{CRS}})} \geq_{UC} \mathcal{F}_{PUBA}^{(\Delta)}$$

against all PPT-adversaries $\mathcal{A}$ *that have statically corrupted the operator* $\mathcal{O}$ *and a subset of users* $\mathcal{U}$*.*

We use the UC-framework [17] and provide a simulator $\mathcal{S}$ for this case. The simulator provides a view for any PPT-environment $\mathcal{Z}$ (that is restricted to not corrupting any proxies) that is consistent with a *real* protocol execution.

The simulator is given as follows:

Simulator $\mathcal{S}_{\text{USec}}$ for a corrupted operator

π*-Shared State*:

- Trapdoor td_{sim} for simulating proofs.
- Verification key $\mathsf{vk}_{\mathcal{O}}$ of the operator
- Mapping f_{OI} on $\{pid_{\mathcal{P}}\} \times \{1,\ldots,K\}$ that maps $(pid_{\mathcal{P}}, k)$ to a list $f_{\text{OI}}(pid_{\mathcal{P}}, k)$ of entries $(lin, \text{sh}_{\mathcal{UH}}^{(\mathcal{P})}, \text{sh}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \text{sh}_{\text{o}_{\text{OUT}}}^{(\mathcal{P})}, \text{sh}_{\text{o}_{\text{UNV}}}^{(\mathcal{P})}, \text{sh}_{\text{o}_{\alpha}}^{(\mathcal{P})}, \text{sh}_{\text{o}_{\mathbf{s}}}^{(\mathcal{P})}, \text{sh}_{\text{o}_{\mathbf{a}}}^{(\mathcal{P})})$ for all proxies $\mathcal{P}$
- Partial mapping f_{UP} on $\{lin\}$: $lin \mapsto (\text{com}_{\mathbf{a}_z}, \text{unv}_{\mathbf{a}_z}, \mathbf{a}_z, \alpha, \mathbf{s}, \text{o}_{\alpha}, \text{o}_{\mathbf{s}}, \text{o}_{\mathbf{a}}, \text{o}_{\text{UNV}}, \text{c}_{out_z})$
- Partial mapping f_{LN} on $\{ssid\}$: $ssid \mapsto lin$
- Mapping f_{FP} that maps a given task *task* to a set of tuples $\{fp, \text{com}_{fp}, \sigma_{fp}\}$ of Function Parameters and corresponding signatures.

π*-Setup*:

1. Run a modified version of $crs \leftarrow \pi$*-Setup*(1^{κ}):
   - $crs_{\mathsf{pok}} \leftarrow$ SetupPoK is replaced by $(crs_{\mathsf{pok}}, td_{\mathsf{sim}}) \leftarrow$ SetupSPoK for simulating proofs.

π*-Init*:

1. Upon receiving instructions from $\mathcal{Z}$ to send $(\texttt{Register}, pid_{\mathcal{O}}, \mathsf{vk}_{\mathcal{O}})$ from $\mathcal{O}$ to $\mathcal{F}_{\text{BB}}$:
   - Check[a] that no key has previously been stored for $pid_{\mathcal{O}}$.
   - Store $\mathsf{vk}_{\mathcal{O}}$.

   ↪ Call Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ with input $(\texttt{operatorInit})$ in the name of $\mathcal{O}$.
2. Upon receiving output $(\texttt{ok})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ to $\mathcal{O}$:

   ↪ Report message $(\texttt{ok})$ from $\mathcal{F}_{\text{BB}}$ to $\mathcal{O}$.
3. Upon receiving $(\texttt{init}, pid_{\mathcal{T}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - Generate signature key pair: $(\mathsf{vk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}}) \leftarrow \mathsf{SIG}'.\mathsf{Gen}(\mathsf{gp})$
   - Respond to future calls of the form $(\texttt{Retrieve}, pid_{\mathcal{T}})$ to $\mathcal{F}_{\text{BB}}$ with $\mathsf{sk}_{\mathcal{T}}$.

π*-Sign Function Parameter*:

1. Upon receiving instructions from $\mathcal{Z}$ to send $(fp, \text{com}_{fp}, \text{unv}_{fp}, task, in_{\mathcal{O}})$ from $\mathcal{O}$ to $\mathcal{T}$:
   - Check[a] that $\mathsf{COM}'.\mathsf{Unv}(\text{com}_{fp}, \text{unv}_{fp}, fp) = 1$.

   ↪ Call Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ in the name of $\mathcal{O}$ with input $(\texttt{SFP}, fp, task, in_{\mathcal{O}})$.
2. Upon receiving output $\texttt{ok}$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ to $\mathcal{O}$:
   - Compute signature $\mathsf{SIG}'.\mathsf{Sgn}(\mathsf{sk}_{\mathcal{T}}, (task, \text{com}_{fp}))$.
   - Add $(fp, \text{com}_{fp}, \sigma_{fp})$ to $f_{\text{FP}}(task)$.

   ↪ Report message (σ_{fp}) from $\mathcal{T}$ to $\mathcal{O}$.

π*-User Registration*:

. . . . . .

$\mathcal{U}$ honest, $\mathcal{O}$ corrupted:

1. Upon receiving $(\texttt{UReg}, pid_{\mathcal{U}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - $id \xleftarrow{r} \mathbb{Z}_p, \mathsf{pk}_{\mathcal{U}} := id \cdot g_1$
   - $(\text{com}_{id}, \text{unv}_{id}) \leftarrow \mathsf{COM}.\mathsf{Com}(\mathbf{0})$
   - $stmt := (\mathsf{pk}_{\mathcal{U}}, \text{com}_{id})$
   - $\Pi \leftarrow \mathsf{SimProof}(stmt, L_{\text{R}})$
   - $(\text{com}_{ser^{\text{new}}}^{(\mathcal{U})}, \text{unv}_{ser^{\text{new}}}^{(\mathcal{U})}) \leftarrow \mathsf{COM}.\mathsf{Com}(\mathbf{0})$

   ↪ Report message $(\Pi, \text{com}_{id}, \text{com}_{ser^{\text{new}}}^{(\mathcal{U})})$ from $\mathcal{U}$ to $\mathcal{O}$.
2. Upon receiving $(\texttt{Retrieve}, pid_{\mathcal{U}})$ from $\mathcal{O}$ to $\mathcal{F}_{\text{BB}}$:

   ↪ Report message $(\texttt{Retrieve}, pid_{\mathcal{U}}, \mathsf{pk}_{\mathcal{U}})$ from $\mathcal{F}_{\text{BB}}$ to $\mathcal{O}$.
3. Upon receiving instructions from $\mathcal{Z}$ to send $(\text{com}_{fp}, \sigma_{fp})$ from $\mathcal{O}$ to $\mathcal{U}$:
   - Check[a] that there exists an entry $(\cdot, \text{com}_{fp}, \sigma_{fp})$ in $f_{\text{FP}}(\texttt{UReg})$.
4. Upon receiving $(\texttt{UReg}, fp, \text{unv}_{fp}, in_{\mathcal{O}})$ from $\mathcal{O}$ to $\mathcal{F}_{\text{PPA}}$:
   - Check[a] that $(\text{com}_{fp}, \sigma_{fp}) \in f_{\text{FP}}(\texttt{UReg})$.
   - Check[a] that $\mathsf{COM}.\mathsf{Unv}(\text{com}_{fp}, \text{unv}_{fp}, fp) = 1$.

↪ Call Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ with input $(\mathtt{UReg}, fp, in_{\mathcal{O}})$ in the name of $\mathcal{O}$.

5. Upon receiving leak ℓ from $\mathcal{F}_{PUBA}^{(\Delta)}$ and output $(out_{\mathcal{O}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - $(\mathtt{com}_{\mathcal{UH}}, \mathtt{unv}_{\mathcal{UH}}) \leftarrow \mathsf{Com}(0)$

   ↪ Report output $(\mathtt{com}_{\mathcal{UH}}, out_{\mathcal{O}})$ from $\mathcal{F}_{\mathrm{PPA}}$ to $\mathcal{O}$.
6. Upon receiving instructions from $\mathcal{Z}$ to send $(\mathtt{com}_{\mathcal{UH}}, ser^{(\mathcal{O})}, \mathtt{com}_{ser}^{(\mathcal{O})}, \mathtt{unv}_{ser}^{(\mathcal{O})}, \mathtt{com}_{lin}, \mathtt{unv}_{lin}, \mathtt{com}_{id}, \mathtt{unv}_{id}, \sigma)$ from $\mathcal{O}$ to $\mathcal{U}$:
   - $\mathtt{com}_{ser} := \mathtt{com}_{ser}^{(\mathcal{O})} \oplus \mathtt{com}_{ser}^{(\mathcal{U})}$
   - $\mathtt{unv}_{ser} := \mathtt{unv}_{ser}^{(\mathcal{O})} \oplus \mathtt{unv}_{ser}^{(\mathcal{U})}$
   - $\lambda^{\mathrm{new}} := \big(0, \mathtt{com}_{\mathcal{UH}}, \mathtt{unv}_{\mathcal{UH}}, ser^{(\mathcal{O})}, \mathtt{com}_{ser}, \mathtt{unv}_{ser}, 0, \mathtt{com}_{lin}, \mathtt{unv}_{lin}, 0, \mathtt{com}_{id}, \mathtt{unv}_{id}, \sigma\big)$
   - Call[a] $\pi_{\mathrm{Verify}}(\lambda^{\mathrm{new}})$

   ↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to deliver output to $\mathcal{U}$.

$\mathcal{U}$ and $\mathcal{O}$ corrupted: Nothing to do.

$\left\{\pi\text{-}Bookkeeping^{(k)}\right\}_{k=1}^{K}$:

$\mathcal{U}$ honest, $\mathcal{O}$ corrupted:

1. Upon receiving $(BK(k), \mathrm{User})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - $ser \xleftarrow{r} \mathbb{Z}_p$
   - $(\mathtt{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})}, \mathtt{unv}_{ser^{\mathrm{new}}}^{(\mathcal{U})}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$
   - $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, \widetilde{\mathtt{unv}_{\mathcal{UH}}}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$
   - $(\widetilde{\mathtt{com}_{lin}}, \widetilde{\mathtt{unv}_{lin}}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$
   - $(\widetilde{\mathtt{com}_{id}}, \widetilde{\mathtt{unv}_{id}}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$
   - $stmt^{Val} := (\widetilde{\mathtt{com}_{\mathcal{UH}}}, ser, \widetilde{\mathtt{com}_{lin}}, \widetilde{\mathtt{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
   - $\Pi_{Val} \leftarrow \mathsf{POK.SimProof}(td_{\mathsf{sim}}, stmt^{Val}, L_{\mathrm{B}}^{Val})$

   ↪ Report message $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, ser, \widetilde{\mathtt{com}_{lin}}, \widetilde{\mathtt{com}_{id}}, \Pi_{Val}, \mathtt{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})})$ from $\mathcal{U}$ to $\mathcal{O}$.
2. Upon receiving instructions from $\mathcal{Z}$ to send $(\mathtt{com}_{fp}, \sigma_{fp})$ from $\mathcal{O}$ to $\mathcal{U}$:
   - Check[a] that $(\mathtt{com}_{fp}, \sigma_{fp}) \in f_{\mathrm{FP}}(\mathtt{UReg})$.
3. Upon receiving $(BK(k), \widetilde{\mathtt{com}_{\mathcal{UH}}}', fp, \mathtt{unv}_{fp}, in_{\mathcal{O}})$ from $\mathcal{O}$ to $\mathcal{F}_{\mathrm{PPA}}$:
   - Check[a] $\widetilde{\mathtt{com}_{\mathcal{UH}}}' = \widetilde{\mathtt{com}_{\mathcal{UH}}}$
   - Check[a] $\mathsf{COM.Unv}(\mathtt{com}_{fp}, \mathtt{unv}_{fp}, fp) = 1$.

   ↪ Call Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ with input $(BK(k), fp, in_{\mathcal{O}})$ in the name of $\mathcal{O}$.
4. Upon receiving leak ℓ from $\mathcal{F}_{PUBA}^{(\Delta)}$ and output $(out_{\mathcal{O}}, \alpha, \mathbf{s})$ to $\mathcal{O}$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - $(\mathtt{com}_{\mathbf{a}}, \mathtt{unv}_{\mathbf{a}}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$
   - **if** $\alpha \neq \bot \vee \mathbf{s} \neq \bot$ **then**
     - $(\mathtt{com}'_{\mathcal{UH}}, \mathtt{unv}'_{\mathcal{UH}}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$
     - $(\mathtt{com}''_{\mathcal{UH}}, \mathtt{unv}''_{\mathcal{UH}}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$
     - $stmt^{Tr} := (\widetilde{\mathtt{com}_{\mathcal{UH}}}, \mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \alpha, \mathbf{s})$
     - $\Pi_{Tr} \leftarrow \mathsf{POK.SimProof}(td_{\mathsf{sim}}, stmt^{Tr}, L_{\mathrm{B}}^{Tr})$
     - $\mathtt{unv}_{\mathcal{UH}}^{\mathrm{new}} := \mathtt{unv}''_{\mathcal{UH}} \oplus \mathtt{unv}_{\mathbf{a}}$
   - **else**
     - $\mathtt{unv}_{\mathcal{UH}}^{\mathrm{new}} := \widetilde{\mathtt{unv}_{\mathcal{UH}}} \oplus \mathtt{unv}_{\mathbf{a}}$

   ↪ Report output $(\alpha, \mathbf{s}, \mathtt{com}_{\mathbf{a}}, out_{\mathcal{O}})$ from $\mathcal{F}_{\mathrm{PPA}}$ to $\mathcal{O}$.
5. **if** $\alpha \neq \bot \vee \mathbf{s} \neq \bot$ **then**

   ↪ Report message $(\mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \Pi_{Tr})$ from $\mathcal{U}$ to $\mathcal{O}$.
6. Upon receiving instructions from $\mathcal{Z}$ to send $(\mathtt{com}_{\mathcal{UH}}^{\mathrm{new}}, (ser^{\mathrm{new}})^{(\mathcal{O})}, \mathtt{com}_{ser^{\mathrm{new}}}^{(\mathcal{O})}, \mathtt{unv}_{ser^{\mathrm{new}}}^{(\mathcal{O})}, \sigma^{\mathrm{new}})$ from $\mathcal{O}$ to $\mathcal{U}$:
   - $ser^{\mathrm{new}} := (ser^{\mathrm{new}})^{(\mathcal{O})} + (ser^{\mathrm{new}})^{(\mathcal{U})}$
   - $\mathtt{com}_{ser}^{\mathrm{new}} := \mathtt{com}_{ser^{\mathrm{new}}}^{(\mathcal{O})} \oplus \mathtt{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})}$
   - $\mathtt{unv}_{ser}^{\mathrm{new}} := \mathtt{unv}_{ser^{\mathrm{new}}}^{(\mathcal{O})} \oplus \mathtt{unv}_{ser^{\mathrm{new}}}^{(\mathcal{U})}$
   - $\lambda^{\mathrm{new}} := (0, \mathtt{com}_{\mathcal{UH}}^{\mathrm{new}}, \mathtt{unv}_{\mathcal{UH}}^{\mathrm{new}}, (ser^{\mathrm{new}})^{(\mathcal{O})}, \mathtt{com}_{ser}^{\mathrm{new}}, \mathtt{unv}_{ser}^{\mathrm{new}}, 0, \widetilde{\mathtt{com}_{lin}}, \widetilde{\mathtt{unv}_{lin}}, 0, \widetilde{\mathtt{com}_{id}}, \widetilde{\mathtt{unv}_{id}}, \sigma^{\mathrm{new}})$
   - Call[a] $\pi_{\mathrm{Verify}}(\lambda^{\mathrm{new}})$

   ↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to deliver output to $\mathcal{U}$.

$\mathcal{U}$ and $\mathcal{O}$ corrupted: Nothing to do but relay messages.

$\left\{\pi\text{-}Outsource^{(k)}\right\}_{k=1}^{K}$:

$\mathcal{U}$ and $\mathcal{P}$ honest, $\mathcal{O}$ corrupted:

1. Upon receiving $(OS(k), \mathrm{User})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ and $(OS(k), pid_{\mathcal{P}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - $ser \xleftarrow{r} \mathbb{Z}_p$
   - $(\widetilde{\mathtt{com}_{id}}, \widetilde{\mathtt{unv}_{id}}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$
   - $(\mathtt{sh}_{\mathcal{UH}}^{(\mathcal{P})}, \mathtt{sh}_{\mathcal{UH}}^{(\mathcal{O})}) \leftarrow \pi_{\mathrm{Share}}\text{-}Share(\mathbf{0})$
   - $(\mathtt{sh}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \mathtt{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}) \leftarrow \pi_{\mathrm{Share}}\text{-}Share(\mathbf{0})$
   - $(\mathtt{sh}_{\mathtt{o}_{\mathtt{OUT}}}^{(\mathcal{P})}, \mathtt{sh}_{\mathtt{o}_{\mathtt{OUT}}}^{(\mathcal{O})}) \leftarrow \pi_{\mathrm{Share}}\text{-}Share(\mathbf{0})$
   - $(\mathtt{sh}_{\mathtt{o}_{\mathtt{UNV}}}^{(\mathcal{P})}, \mathtt{sh}_{\mathtt{o}_{\mathtt{UNV}}}^{(\mathcal{O})}) \leftarrow \pi_{\mathrm{Share}}\text{-}Share(\mathbf{0})$
   - $(\mathtt{sh}_{\mathtt{o}_{\alpha}}^{(\mathcal{P})}, \mathtt{sh}_{\mathtt{o}_{\alpha}}^{(\mathcal{O})}) \leftarrow \pi_{\mathrm{Share}}\text{-}Share(\mathbf{0})$
   - $(\mathtt{sh}_{\mathtt{o}_{\mathbf{s}}}^{(\mathcal{P})}, \mathtt{sh}_{\mathtt{o}_{\mathbf{s}}}^{(\mathcal{O})}) \leftarrow \pi_{\mathrm{Share}}\text{-}Share(\mathbf{0})$
   - $(\mathtt{sh}_{\mathtt{o}_{\mathbf{a}}}^{(\mathcal{P})}, \mathtt{sh}_{\mathtt{o}_{\mathbf{a}}}^{(\mathcal{O})}) \leftarrow \pi_{\mathrm{Share}}\text{-}Share(\mathbf{0})$
   - Parse $(\mathcal{UH}^{(\mathcal{P})}, \mathtt{com}_{\mathcal{UH}}^{(\mathcal{O})}, \mathtt{unv}_{\mathcal{UH}}^{(\mathcal{P})}) := \mathtt{sh}_{\mathcal{UH}}^{(\mathcal{P})}$
   - $stmt := (\mathtt{sh}_{\mathcal{UH}}^{(\mathcal{O})}, \mathtt{com}_{\mathcal{UH}}^{(\mathcal{O})}, ser, \widetilde{\mathtt{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
   - $\Pi \leftarrow \mathsf{POK.SimProof}(td_{\mathsf{sim}}, stmt, L_{\mathrm{O}})$
   - $(\mathtt{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})}, \mathtt{unv}_{ser^{\mathrm{new}}}^{(\mathcal{U})}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$

$\hookrightarrow$ Report message $(\mathrm{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o_{OUT}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o_{UNV}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\alpha}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o_s}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o_a}}, \mathtt{com}^{(\mathcal{O})}_{\mathcal{UH}}, ser, \widetilde{\mathtt{com}_{id}}, \Pi, \mathtt{com}^{(\mathcal{U})}_{ser^{\mathrm{new}}})$ from $\mathcal{U}$ to $\mathcal{O}$.

2. Upon receiving instructions from $\mathcal{Z}$ to send $(\mathtt{com}^{(\mathcal{O})}_{lin^{\mathrm{new}}}, \mathtt{com}^{(\mathcal{P})}_{\mathcal{UH}}, \mathtt{com}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathtt{com}^{(\mathcal{P})}_{\mathtt{o_{OUT}}}, \mathtt{com}^{(\mathcal{P})}_{\mathtt{o_{UNV}}}, \mathtt{com}^{(\mathcal{P})}_{\mathtt{o}_\alpha}, \mathtt{com}^{(\mathcal{P})}_{\mathtt{o_s}}, \mathtt{com}^{(\mathcal{P})}_{\mathtt{o_a}})$ from $\mathcal{O}$ to $\mathcal{P}$:
   - Check[a] that all commitments are the ones sent earlier.
   - $(lin^{\mathrm{new}})^{(\mathcal{P})} \xleftarrow{\mathrm{r}} \mathbb{Z}_p$

   Parse:
   - $\left(\mathcal{UH}^{(\mathcal{P})}, \mathtt{com}^{(\mathcal{O})}_{\mathcal{UH}}, \mathtt{unv}^{(\mathcal{P})}_{\mathcal{UH}}\right) := \mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}$
   - $\left(in^{\mathcal{P}}_{\mathcal{U}}, \mathtt{com}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathtt{unv}^{\mathcal{P}}_{in_{\mathcal{U}}}\right) := \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}$
   - $\left(\mathtt{o_{OUT}}^{\mathcal{P}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o_{OUT}}}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o_{OUT}}}\right) := \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_{OUT}}}$
   - $\left(\mathtt{o_{UNV}}^{\mathcal{P}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o_{UNV}}}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o_{UNV}}}\right) := \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_{UNV}}}$
   - $\left(\mathtt{o}_\alpha{}^{\mathcal{P}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o}_\alpha}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o}_\alpha}\right) := \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\alpha}$
   - $\left(\mathtt{o_s}^{\mathcal{P}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o_s}}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o_s}}\right) := \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_s}}$
   - $\left(\mathtt{o_a}^{\mathcal{P}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o_a}}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o_a}}\right) := \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_a}}$

   $\hookrightarrow$ Report message $((lin^{\mathrm{new}})^{(\mathcal{P})}, \mathtt{com}^{(\mathcal{O})}_{\mathcal{UH}}, \mathtt{com}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o_{OUT}}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o_{UNV}}}, \mathtt{com}^{(\mathcal{O})}_{\alpha}, \mathtt{com}^{(\mathcal{O})}_{\mathbf{s}}, \mathtt{com}^{(\mathcal{O})}_{\mathbf{a}})$ from $\mathcal{P}$ to $\mathcal{O}$.
3. Upon receiving instructions from $\mathcal{Z}$ to send $(lin^{\mathrm{new}})^{(\mathcal{O})}, \mathtt{unv}^{(\mathcal{O})}_{lin^{\mathrm{new}}})$ from $\mathcal{O}$ to $\mathcal{P}$:
   - Check[a] $\mathsf{COM.Unv}(\mathtt{com}^{(\mathcal{O})}_{lin^{\mathrm{new}}}, \mathtt{unv}^{(\mathcal{O})}_{lin^{\mathrm{new}}}, (lin^{\mathrm{new}})^{(\mathcal{O})}) \stackrel{?}{=} 1$
   - $lin^{\mathrm{new}} := (lin^{\mathrm{new}})^{(\mathcal{P})} + (lin^{\mathrm{new}})^{(\mathcal{O})}$
   - Append $(lin^{\mathrm{new}}, \mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_{UNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_a}})$ to $f_{\mathrm{OI}}(pid_{\mathcal{P}}, k)$
   - Load current Subsession Identifier *ssid* and append $ssid \mapsto lin^{\mathrm{new}}$ to f_{LN}
4. Upon receiving instructions from $\mathcal{Z}$ to send $((ser^{\mathrm{new}})^{(\mathcal{O})}, \mathtt{com}^{(\mathcal{O})}_{ser^{\mathrm{new}}}, \mathtt{unv}^{(\mathcal{O})}_{ser^{\mathrm{new}}}, \mathtt{com}^{\mathrm{new}}_{lin}, \mathtt{unv}^{\mathrm{new}}_{lin}, \sigma^{\mathrm{new}})$ from $\mathcal{O}$ to $\mathcal{U}$:
   - Parse $(\mathcal{UH}^{(\mathcal{O})}, \mathtt{com}^{(\mathcal{P})}_{\mathcal{UH}}, \mathtt{unv}^{(\mathcal{O})}_{\mathcal{UH}}) := \mathrm{sh}^{(\mathcal{O})}_{\mathcal{UH}}$
   - $\mathtt{com}^{\mathrm{new}}_{\mathcal{UH}} := \mathtt{com}^{(\mathcal{P})}_{\mathcal{UH}} \oplus \mathtt{com}^{(\mathcal{O})}_{\mathcal{UH}}$
   - $\mathtt{unv}^{\mathrm{new}}_{\mathcal{UH}} := \mathtt{unv}^{(\mathcal{P})}_{\mathcal{UH}} \oplus \mathtt{unv}^{(\mathcal{O})}_{\mathcal{UH}}$
   - $\mathtt{com}^{\mathrm{new}}_{ser} := \mathtt{com}^{(\mathcal{U})}_{ser^{\mathrm{new}}} \oplus \mathtt{com}^{(\mathcal{O})}_{ser^{\mathrm{new}}}$
   - $\mathtt{unv}^{\mathrm{new}}_{ser} := \mathtt{unv}^{(\mathcal{U})}_{ser^{\mathrm{new}}} \oplus \mathtt{unv}^{(\mathcal{O})}_{ser^{\mathrm{new}}}$
   - $\lambda^{\mathrm{new}} :- (\mathbf{0}, \mathtt{com}^{\mathrm{new}}_{\mathcal{UH}}, \mathtt{unv}^{\mathrm{new}}_{\mathcal{UH}}, (ser^{\mathrm{new}})^{(\mathcal{O})}, \mathtt{com}^{\mathrm{new}}_{ser}, \mathtt{unv}^{\mathrm{new}}_{ser}, lin^{\mathrm{new}}, \mathtt{com}^{\mathrm{new}}_{lin}, \mathtt{unv}^{\mathrm{new}}_{lin}, \mathbf{0}, \widetilde{\mathtt{com}_{id}}, \widetilde{\mathtt{unv}_{id}}, \sigma^{\mathrm{new}})$
   - Call[a] $\pi_{\mathrm{Verify}}(\lambda^{\mathrm{new}})$

   $\hookrightarrow$ Call Ideal Functionality $\mathcal{F}^{(\Delta)}_{PUBA}$ with input $(OS(k))$ in the name of $\mathcal{O}$.
5. Allow $\mathcal{F}^{(\Delta)}_{PUBA}$ to deliver output to all parties.

$\mathcal{P}$ honest, $\mathcal{U}$ and $\mathcal{O}$ corrupted:

1. Upon receiving $(OS(k), pid_{\mathcal{P}})$ from $\mathcal{F}^{(\Delta)}_{PUBA}$ and instructions from $\mathcal{Z}$ to send $(\mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_{UNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_a}})$ from $\mathcal{U}$ to $\mathcal{P}$ and $(\mathtt{com}^{(\mathcal{O})}_{lin^{\mathrm{new}}}, \mathtt{com}^{(\mathcal{P})}_{\mathcal{UH}}, \mathtt{com}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathtt{com}^{(\mathcal{P})}_{\mathtt{o_{OUT}}}, \mathtt{com}^{(\mathcal{P})}_{\mathtt{o_{UNV}}}, \mathtt{com}^{(\mathcal{P})}_{\mathtt{o}_\alpha}, \mathtt{com}^{(\mathcal{P})}_{\mathtt{o_s}}, \mathtt{com}^{(\mathcal{P})}_{\mathtt{o_a}})$ from $\mathcal{O}$ to $\mathcal{P}$:

   Parse:
   - $\left(\mathcal{UH}^{(\mathcal{P})}, \mathtt{com}^{(\mathcal{O})}_{\mathcal{UH}}, \mathtt{unv}^{(\mathcal{P})}_{\mathcal{UH}}\right) := \mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}$
   - $\left(in^{\mathcal{P}}_{\mathcal{U}}, \mathtt{com}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathtt{unv}^{\mathcal{P}}_{in_{\mathcal{U}}}\right) := \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}$
   - $\left(\mathtt{o_{OUT}}^{\mathcal{P}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o_{OUT}}}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o_{OUT}}}\right) := \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_{OUT}}}$
   - $\left(\mathtt{o_{UNV}}^{\mathcal{P}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o_{UNV}}}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o_{UNV}}}\right) := \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_{UNV}}}$
   - $\left(\mathtt{o}_\alpha{}^{\mathcal{P}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o}_\alpha}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o}_\alpha}\right) := \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\alpha}$
   - $\left(\mathtt{o_s}^{\mathcal{P}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o_s}}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o_s}}\right) := \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_s}}$
   - $\left(\mathtt{o_a}^{\mathcal{P}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o_a}}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o_a}}\right) := \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_a}}$

   Check shares
   - Check[a] $\mathsf{Unv}(\mathtt{com}^{(\mathcal{P})}_{\mathcal{UH}}, \mathtt{unv}^{(\mathcal{P})}_{\mathcal{UH}}, \mathcal{UH}^{(\mathcal{P})}) \stackrel{?}{=} 1$
   - Check[a] $\mathsf{Unv}(\mathtt{com}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathtt{unv}^{\mathcal{P}}_{in_{\mathcal{U}}}, in^{\mathcal{P}}_{\mathcal{U}}) \stackrel{?}{=} 1$
   - Check[a] $\mathsf{Unv}(\mathtt{com}^{(\mathcal{P})}_{\mathtt{o_{OUT}}}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o_{OUT}}}, \mathtt{o_{OUT}}^{\mathcal{P}}) \stackrel{?}{=} 1$
   - Check[a] $\mathsf{Unv}(\mathtt{com}^{(\mathcal{P})}_{\mathtt{o_{UNV}}}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o_{UNV}}}, \mathtt{o_{UNV}}^{\mathcal{P}}) \stackrel{?}{=} 1$
   - Check[a] $\mathsf{Unv}(\mathtt{com}^{(\mathcal{P})}_{\mathtt{o}_\alpha}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o}_\alpha}, \mathtt{o}_\alpha{}^{\mathcal{P}}) \stackrel{?}{=} 1$
   - Check[a] $\mathsf{Unv}(\mathtt{com}^{(\mathcal{P})}_{\mathtt{o_s}}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o_s}}, \mathtt{o_s}^{\mathcal{P}}) \stackrel{?}{=} 1$
   - Check[a] $\mathsf{Unv}(\mathtt{com}^{(\mathcal{P})}_{\mathtt{o_a}}, \mathtt{unv}^{\mathcal{P}}_{\mathtt{o_a}}, \mathtt{o_a}^{\mathcal{P}}) \stackrel{?}{=} 1$

   Perform coin toss with $\mathcal{O}$
   - $(lin^{\mathrm{new}})^{(\mathcal{P})} \xleftarrow{\mathrm{r}} \mathbb{Z}_p$

   $\hookrightarrow$ Report message $((lin^{\mathrm{new}})^{(\mathcal{P})}, \mathtt{com}^{(\mathcal{O})}_{\mathcal{UH}}, \mathtt{com}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o_{OUT}}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o_{UNV}}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o}_\alpha}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o_s}}, \mathtt{com}^{(\mathcal{O})}_{\mathtt{o_a}})$ from $\mathcal{P}$ to $\mathcal{O}$.
2. Upon receiving instructions from $\mathcal{Z}$ to send $((lin^{\mathrm{new}})^{(\mathcal{O})}, \mathtt{unv}^{(\mathcal{O})}_{lin^{\mathrm{new}}})$ from $\mathcal{O}$ to $\mathcal{P}$:
   - Check[a] $\mathsf{COM.Unv}(\mathtt{com}^{(\mathcal{O})}_{lin^{\mathrm{new}}}, \mathtt{unv}^{(\mathcal{O})}_{lin^{\mathrm{new}}}, (lin^{\mathrm{new}})^{(\mathcal{O})}) \stackrel{?}{=} 1$
   - $lin^{\mathrm{new}} := (lin^{\mathrm{new}})^{(\mathcal{P})} + (lin^{\mathrm{new}})^{(\mathcal{O})}$
   - Append $(lin^{\mathrm{new}}, \mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_{UNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o_a}})$ to $f_{\mathrm{OI}}(pid_{\mathcal{P}}, k)$
   - Load current Subsession Identifier *ssid* and append $ssid \mapsto lin^{\mathrm{new}}$ to f_{LN}.

   $\hookrightarrow$ Report message (lin^{new}) from $\mathcal{P}$ to $\mathcal{U}$.

   $\hookrightarrow$ Call Ideal Functionality $\mathcal{F}^{(\Delta)}_{PUBA}$ with input $(OS(k), \perp)$ in the name of $\mathcal{U}$.

   $\hookrightarrow$ Call Ideal Functionality $\mathcal{F}^{(\Delta)}_{PUBA}$ with input $(OS(k))$ in the name of $\mathcal{O}$.

3. Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to deliver output to all parties.

$\left\{\pi\text{-}Outsourced\ Analytics^{(k)}\right\}_{k=1}^{K}$:

$\mathcal{P}$ honest, $\mathcal{O}$ corrupted:

1. Upon receiving instructions from $\mathcal{Z}$ to send $(\mathtt{com}_{fp}, \sigma_{fp})$ from $\mathcal{O}$ to $\mathcal{U}$:
   - Check[b] that $(\mathtt{com}_{fp}, \sigma_{fp}) \in f_{\mathrm{FP}}(OA(k))$.
2. Upon receiving $(OA(k), pid_{\mathcal{P}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ and $(OA(k), \{(\mathrm{sh}_{\mathcal{UH}}^{(\mathcal{O})}, \mathrm{sh}_{in}^{(\mathcal{O})}, \mathrm{sh}_{\mathtt{o}_{\mathtt{OUT}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathtt{o}_{\mathtt{UNV}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathtt{o}_\alpha}^{(\mathcal{O})}, \mathrm{sh}_{\mathtt{o}_\mathbf{s}}^{(\mathcal{O})}, \mathrm{sh}_{\mathtt{o}_\mathbf{a}}^{(\mathcal{O})})_z\}_{z=1}^{Z_k}, fp, \mathrm{unv}_{fp}, in_{\mathcal{O}})$ from $\mathcal{O}$ to $\mathcal{F}_{\mathrm{PPA}}$:
   - Check[a] that $\mathsf{COM.Unv}(\mathtt{com}_{fp}, \mathrm{unv}_{fp}, fp) = 1$.
   - Load[a] and remove the first Z_k entries $\{(lin, \mathrm{sh}_{\mathcal{UH}}^{(\mathcal{P})}, \mathrm{sh}_{in}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\mathtt{OUT}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\mathtt{UNV}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_\alpha}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_\mathbf{s}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_\mathbf{a}}^{(\mathcal{P})})_z\}_{z=1}^{Z_k}$ from $f_{\mathrm{OI}}(pid_{\mathcal{P}}, k)$
   - For every z from 1 to Z, combine[a]:
     - $\mathcal{UH}_z \coloneqq \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathcal{UH}_z}^{(\mathcal{P})}, \mathrm{sh}_{\mathcal{UH}_z}^{(\mathcal{O})})$
     - $in_z \coloneqq \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{in_z}^{(\mathcal{P})}, \mathrm{sh}_{in_z}^{(\mathcal{O})})$
     - $\mathtt{o}_{\mathtt{OUT}\,z} \coloneqq \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathtt{o}_{\mathtt{OUT}\,z}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\mathtt{OUT}\,z}}^{(\mathcal{O})})$
     - $\mathtt{o}_{\mathtt{UNV}\,z} \coloneqq \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathtt{o}_{\mathtt{UNV}\,z}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\mathtt{UNV}\,z}}^{(\mathcal{O})})$
     - $\mathtt{o}_{\alpha z} \coloneqq \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathtt{o}_{\alpha z}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\alpha z}}^{(\mathcal{O})})$
     - $\mathtt{o}_{\mathbf{s}z} \coloneqq \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathtt{o}_{\mathbf{s}z}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\mathbf{s}z}}^{(\mathcal{O})})$
     - $\mathtt{o}_{\mathbf{a}z} \coloneqq \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathtt{o}_{\mathbf{a}z}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\mathbf{a}z}}^{(\mathcal{O})})$

   ↪ Call Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ with input $(OA(k), fp, in_{\mathcal{O}})$ in the name of $\mathcal{O}$ and receive leak ℓ.
3. Upon being asked by $\mathcal{F}_{PUBA}^{(\Delta)}$ for updated inputs for $(z_1, \ldots, z_n)$:

   ↪ Provide inputs $\{(\mathcal{UH}_{z_i}, in_{z_i}) \mid 1 \le i \le n\}$ to Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$.
4. Upon receiving $\{(z_i, \alpha_z, \mathbf{s}_z, \mathbf{a}_z, out_{z_i}) \mid 1 \le i \le n\}$ as leak from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - For every z_i with $1 \le i \le n$:
     - $(\mathtt{com}_{\mathbf{a}_{z_i}}, \mathrm{unv}_{\mathbf{a}_z}) \xleftarrow{\mathrm{r}} \mathsf{Com}(\mathbf{a}_z)$
     - $\mathtt{c}_{out_z} \coloneqq out_{z_i} + \mathtt{o}_{\mathtt{OUT}\,z}$
   - For all other $z \le Z_k$:
     - $(\mathtt{com}_{\mathbf{a}_{z_i}}, \mathrm{unv}_{\mathbf{a}_z}) \xleftarrow{\mathrm{r}} \mathsf{Com}(0)$
     - $r \xleftarrow{\mathrm{r}} \mathcal{IN}$
     - $\mathtt{c}_{out_z} \coloneqq r + \mathtt{o}_{\mathtt{OUT}\,z}$
     - $\mathbf{a}_z = \bot$

   ↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to continue.
5. Upon receiving output $(\{\alpha_z, \mathbf{s}_z\}_{z=1}^{Z_k}, out_{\mathcal{O}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ to $\mathcal{O}$:
   - For each $z \in 1 \ldots Z$:
     - $f_{\mathrm{UP}}(lin_z) \coloneqq (\mathtt{com}_{\mathbf{a}_z}, \mathrm{unv}_{\mathbf{a}_z}, \mathbf{a}_z, \alpha_z, \mathbf{s}_z, \mathtt{o}_{\alpha z}, \mathtt{o}_{\mathbf{s}z}, \mathtt{o}_{\mathbf{a}z}, \mathtt{o}_{\mathtt{UNV}}, \mathtt{c}_{out_z})$

   ↪ Report output $(\{(\alpha_z, \mathbf{s}_z, \mathtt{com}_{\mathbf{a}_z}, \mathtt{c}_{out_z}) \mid 1 \le z \le Z_k\}, out_{\mathcal{O}})$ from $\mathcal{F}_{\mathrm{PPA}}$ to $\mathcal{O}$.

π-*Update*:

$\mathcal{U}$ and $\mathcal{P}$ honest, $\mathcal{O}$ corrupted:

1. Upon receiving $(\mathtt{Upd}, \mathrm{User})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ and $(\mathtt{Upd}, pid_{\mathcal{P}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:

   ↪ Call Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ with input $(\mathtt{Upd})$ in the name of $\mathcal{O}$.
2. Upon receiving leak $ssid$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - Load $lin :- f_{\mathrm{LN}}(ssid)$
   - Load[a] and remove $(\mathtt{com}'_{\mathbf{a}}, \mathrm{unv}'_{\mathbf{a}}, \mathbf{a}', \alpha', \mathbf{s}', \mathtt{o}_\alpha, \mathtt{o}_{\mathbf{s}}, \mathtt{o}_{\mathbf{a}}, \mathtt{o}_{\mathtt{UNV}}, \mathtt{c}_{out_{\mathcal{U}}}{}') \coloneqq f_{\mathrm{UP}}(lin)$
   - $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, \widetilde{\mathrm{unv}_{\mathcal{UH}}}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$
   - $(\widetilde{\mathtt{com}_{id}}, \widetilde{\mathrm{unv}_{id}}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$
   - $ser \xleftarrow{\mathrm{r}} \mathbb{Z}_p$
   - $stmt \coloneqq (\widetilde{\mathtt{com}_{\mathcal{UH}}}, ser, lin, \widetilde{\mathtt{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
   - $\Pi \leftarrow \mathsf{POK.SimProof}(td_{\mathsf{sim}}, stmt, L_{\mathrm{U}})$
   - $(\mathtt{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})}, \mathrm{unv}_{ser^{\mathrm{new}}}^{(\mathcal{U})}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$
3. **if** $\alpha \neq \bot \wedge \mathbf{s} \neq \bot$ **then**
   - $(\mathtt{com}'_{\mathcal{UH}}, \mathrm{unv}'_{\mathcal{UH}}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$
   - $(\mathtt{com}''_{\mathcal{UH}}, \widetilde{\mathrm{unv}''_{\mathcal{UH}}}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$
   - $\mathrm{unv}_{\mathcal{UH}}^{\mathrm{new}} \coloneqq \widetilde{\mathrm{unv}''_{\mathcal{UH}}} \oplus \mathrm{unv}_{\mathbf{a}}$
   - $stmt^{Tr} \coloneqq (\widetilde{\mathtt{com}_{\mathcal{UH}}}, \mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \alpha, \mathbf{s})$
   - $\Pi_{Tr} \leftarrow \mathsf{POK.SimProof}(td_{\mathsf{sim}}, stmt^{Tr}, L_{\mathrm{B}}^{Tr})$

     ↪ Report message $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, \mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \widetilde{\mathtt{com}_{id}}, ser, lin, \Pi, \Pi_{Tr}, \mathtt{com}_{ser}^{(\mathcal{U})})$ from $\mathcal{U}$ to $\mathcal{O}$.
4. **else**
   - $\mathrm{unv}_{\mathcal{UH}}^{\mathrm{new}} \coloneqq \widetilde{\mathrm{unv}_{\mathcal{UH}}} \oplus \mathrm{unv}_{\mathbf{a}}$

     ↪ Report message $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, \widetilde{\mathtt{com}_{id}}, ser, lin, \Pi, \mathtt{com}_{ser}^{(\mathcal{U})})$ from $\mathcal{U}$ to $\mathcal{O}$.
5. **fi**

   ↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to continue.
6. Upon receiving instructions from $\mathcal{Z}$ to send $(\mathtt{com}_{\mathbf{a}}, \mathtt{com}_{\mathcal{UH}}^{\mathrm{new}}, (ser^{\mathrm{new}})^{(\mathcal{O})}, \mathtt{com}_{ser^{\mathrm{new}}}^{(\mathcal{O})}, \mathrm{unv}_{ser^{\mathrm{new}}}^{(\mathcal{O})}, \mathtt{com}_{lin}^{\mathrm{new}}, \mathrm{unv}_{lin}^{\mathrm{new}}, \sigma^{\mathrm{new}}, \mathtt{c}_{out_{\mathcal{U}}})$ from $\mathcal{O}$ to $\mathcal{U}$:
   - Check[a] $\mathtt{com}_{\mathbf{a}} \stackrel{?}{=} \mathtt{com}'_{\mathbf{a}}$
   - Check[a] $\mathtt{c}_{out_{\mathcal{U}}} \stackrel{?}{=} \mathtt{c}_{out_{\mathcal{U}}}{}'$
   - $\mathtt{com}_{ser}^{\mathrm{new}} \coloneqq \mathtt{com}_{ser^{\mathrm{new}}}^{(\mathcal{O})} \oplus \mathtt{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})}$
   - $\mathrm{unv}_{ser}^{\mathrm{new}} \coloneqq \mathrm{unv}_{ser^{\mathrm{new}}}^{(\mathcal{O})} \oplus \mathrm{unv}_{ser^{\mathrm{new}}}^{(\mathcal{U})}$

– $\lambda^{\text{new}} \coloneqq (0, \mathtt{com}^{\text{new}}_{\mathcal{UH}}, \mathtt{unv}^{\text{new}}_{\mathcal{UH}}, (ser^{\text{new}})^{(\mathcal{O})}, \mathtt{com}^{\text{new}}_{ser}, \mathtt{unv}^{\text{new}}_{ser}, 0, \mathtt{com}^{\text{new}}_{lin}, \mathtt{unv}^{\text{new}}_{lin}, \mathbf{0}, \widetilde{\mathtt{com}_{id}}, \widetilde{\mathtt{unv}_{id}}, \sigma^{\text{new}})$
– Call[a] $\pi_{\text{Verify}}(\lambda^{\text{new}})$

↪ Allow $\mathcal{F}^{(\Delta)}_{PUBA}$ to deliver output to all parties.

.....

$\mathcal{P}$ honest, $\mathcal{U}$ and $\mathcal{O}$ corrupted:

1. Upon receiving $(\mathtt{Upd}, pid_{\mathcal{P}})$ from $\mathcal{F}^{(\Delta)}_{PUBA}$ and instructions from $\mathcal{Z}$ to send *lin* from $\mathcal{U}$ to $\mathcal{P}$:
   – Load[a] and remove $(\mathtt{com}_{\mathbf{a}}, \mathtt{unv}_{\mathbf{a}}, \mathbf{a}, \alpha, \mathbf{s}, \mathtt{o}_\alpha, \mathtt{o}_{\mathbf{s}}, \mathtt{o}_{\mathbf{a}}, \mathtt{o}_{\mathtt{UNV}}, \mathtt{c}_{out_{\mathcal{U}}}) \coloneqq f_{\text{UP}}(lin)$
   – $\mathtt{c}_\alpha \coloneqq \alpha + \mathtt{o}_\alpha$
   – $\mathtt{c}_{\mathbf{s}} \coloneqq \mathbf{s} + \mathtt{o}_{\mathbf{s}}$
   – $\mathtt{c}_{\mathbf{a}} \coloneqq \mathbf{a} + \mathtt{o}_{\mathbf{a}}$
   – $\mathtt{c}_{\mathtt{unv}_{\mathbf{a}}} \coloneqq \mathtt{unv}_{\mathbf{a}} + \mathtt{o}_{\mathtt{UNV}}$
   – $\mathtt{c}_{out_{\mathcal{U}}} \coloneqq out_{\mathcal{U}} + \mathtt{o}_{\mathtt{OUT}}$

   ↪ Report message $(\mathtt{c}_\alpha, \mathtt{c}_{\mathbf{s}}, \mathtt{c}_{\mathbf{a}}, \mathtt{c}_{\mathtt{unv}_{\mathbf{a}}}, \mathtt{c}_{out_{\mathcal{U}}})$ from $\mathcal{P}$ to $\mathcal{U}$.

   ↪ Call Ideal Functionality $\mathcal{F}^{(\Delta)}_{PUBA}$ with input $(\mathtt{Upd})$ in the name of $\mathcal{O}$ and a (random) corrupted $\mathcal{U}$.
2. Receive output $(\mathtt{ok})$ from $\mathcal{F}^{(\Delta)}_{PUBA}$ to $\mathcal{U}$ and $\mathcal{O}$

a If this fails, output $\perp$ and abort.
b failAbortSimUSec

We now introduce a series of hybrid games H_i and corresponding simulators $\mathcal{S}_i$ for protocols π_{PUBAi}. Formally, given security parameter κ, each hybrid has the following form:

$$\mathsf{H}_i \coloneqq \text{EXEC}_{\pi_{PUBAi}, \mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{PPA}}, \mathcal{S}_i, \mathcal{Z}}(1^\kappa)$$

We then show for each pair of consecutive hybrids H_i and H_{i+1}, that, given our underlying assumptions, no distinguisher can distinguish the two games better than by guessing.

For our proof, we consider the following hybrid games H_i:

H_1 The hybrid H_1 is equivalent to the real experiment. That is,

$$\mathsf{H}_1 \coloneqq \text{EXEC}_{\pi_{PUBA}, \mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{PPA}}, \mathcal{S}_1, \mathcal{Z}}(1^\kappa)$$

This means that all parties execute the real protocol.

H_2 All calls to hybrid functionalities, namely to $\mathcal{F}_{\text{PPA}}$, $\mathcal{F}_{\text{KE}}$ and $\mathcal{F}_{\text{BB}}$, are replaced by calls to $\mathcal{S}_2$, who simulates their behavior.

H_3 Introduces a map f_{UP} to be used by the simulator $\mathcal{S}_3$, which is similar to what an honest proxy would store for the updates of the User History after an Outsourced Analytics. During simulation of $\mathcal{F}_{\text{PPA}}$ for $OA(k)$, i.e. after $\mathcal{O}$ sent a message $(OA(k), fp, \mathtt{unv}_{fp}, [\{(\mathtt{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathtt{sh}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{OUT}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{UNV}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_\alpha}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathbf{s}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathbf{a}}})_z\}_{z=1}^{Z_k}], in_{\mathcal{O}})$ to $\mathcal{F}_{\text{PPA}}$, and $\mathcal{P}$ has sent a message $[(OA(k), \mathtt{com}_{fp}, \{(\mathtt{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathtt{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathtt{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathtt{OUT}}}, \mathtt{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathtt{UNV}}}, \mathtt{sh}^{(\mathcal{P})}_{\mathtt{o}_\alpha}, \mathtt{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathbf{s}}}, \mathtt{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathbf{a}}})_z\}_{z=1}^{Z_k})]$ to $\mathcal{F}_{\text{PPA}}$, $\mathcal{S}_3$ computes Δ honestly (with fresh coins, if necessary), based on the two inputs. $\mathcal{S}_3$ uses π_{Share}-*Combine* on the inputs to reconstruct $\mathcal{UH}_z$, in_z, $\mathtt{o}_{\mathtt{OUT}z}$ $\mathtt{o}_{\mathtt{UNV}z}$, $\mathtt{o}_{\alpha z}$, $\mathtt{o}_{\mathbf{s}z}$ and $\mathtt{o}_{\mathbf{a}z}$ for each user $z \in [Z]$. If reconstruction on any of the shares fails, $\mathcal{S}_3$ aborts. With this, $\mathcal{S}_3$ computes $(\mathtt{com}_{\mathbf{a}_z}, \mathtt{unv}_{\mathbf{a}_z}) \leftarrow \mathsf{Com}(\mathbf{a}_z)$ and $\mathtt{c}_{out_z} \coloneqq out_z + \mathtt{o}_{\mathtt{OUT}z}$. $\mathcal{S}_3$ then adds a new entry $(lin_z \mapsto (\mathtt{com}_{\mathbf{a}_z}, \mathtt{unv}_{\mathbf{a}_z}, \mathbf{a}_z, \alpha_z, \mathbf{s}_z, \mathtt{o}_{\alpha z}, \mathtt{o}_{\mathbf{s}z}, \mathtt{o}_{\mathbf{a}z}, \mathtt{o}_{\mathtt{UNV}z}, \mathtt{c}_{out_z}))$ to f_{UP}.

H_4 During setup, instead of honestly sampling a *crs*, $\mathcal{S}_4$ computes $(crs_{\mathsf{pok}}, td_{\mathsf{sim}}) \leftarrow \mathsf{SetupSPoK}$ and publishes $crs \coloneqq crs_{\mathsf{pok}}$ as Common Reference String. Also, the simulator stores the verification key $\mathsf{vk}_{\mathcal{O}}$ of the operator $\mathcal{O}$, which is obtained by simulating $\mathcal{F}_{\text{BB}}$ during the initialization. The simulator stores td_{sim}, the remainder stays as it is.

H_5 Replaces all zero-knowledge proofs of honest parties by simulated proofs (using td_{sim}) created by the simulator. Note that the simulated proofs can be created *independently* from (thus without knowing) the actual witness.

H_6 During simulation of the Bookkeeping task, instead of computing the addition vector $\mathbf{a}$ alongside its commitment and decommitment information honestly according to the function Δ the simulator uses $\mathbf{a} = \mathbf{0}$ alongside commitment- and decommitment-information $(\mathtt{com}_{\mathbf{a}}, \mathtt{unv}_{\mathbf{a}}) \leftarrow \mathsf{COM.Com}(0)$. Note that the values for $\mathbf{a}$ and $\mathtt{unv}_{\mathbf{a}}$ are not needed for simulation since H_5 so the only remaining value visible to the environment is $\mathtt{com}_{\mathbf{a}}$.

H_7 All commitments that honest players create in the real protocol (i.e. those on *lin*, *ser*, $\mathcal{UH}$ and *id*) are now created by the simulator as $\mathtt{com}_{\mathbf{0}}$, i.e. commitments to the zero-vector of appropriate size. Also, whenever the user $\mathcal{U}$ is supposed to send a serial number, $\mathcal{S}_7$ samples a new value $ser \xleftarrow{\text{r}} \mathbb{Z}_p$ and sends this instead of a real serial number.

H_8 All proxies $\mathcal{P}$ are replaced by an equivalent machine $\mathcal{P}'$, which behave similar as $\mathcal{P}$, with the exception that during the Outsource-

task, $\mathcal{P}'$ sends to the simulator the shares $(lin, \mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{a}})$.

H_9 Introduces a map f_{OI} to be used by the simulator $\mathcal{S}_9$, which is similar to what an honest proxy would store for the outsource information. It maps $pid_{\mathcal{P}} \times [K]$ to a list $f_{\mathrm{OI}}(pid_{\mathcal{P}}, k)$ of entries $(lin, \mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{a}})$.
The map is only updated during Outsource-tasks. After $\mathcal{S}_9$ received the leak $(lin, \mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{a}})$ from $\mathcal{P}'$, $\mathcal{S}_9$ adds an entry $(pid_{\mathcal{P}}, k) \mapsto (lin, \mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{a}})$ to f_{OI}.

H_{10} Introduces a map f_{LN} for the simulator, which contains a mapping from the current Subsession Identifier *ssid* to the linking number *lin*. The map is only updated during Outsource-tasks. After receiving the leak $(lin, \mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{a}})$ from the proxy $\mathcal{P}'$, the simulator loads the current Subsession Identifier *ssid* and adds an entry $(ssid \to lin)$ to f_{LN}.

H_{11} All calls from honest parties of the form π_{Share}-*Share*(x) for an arbitrary $x \in \mathbb{Z}_p^n$ during the protocol execution are replaced by calls π_{Share}-*Share*$(\mathbf{0})$ from the simulator, where $\mathbf{0} = 0^n$ is the all-zero vector of appropriate size. Furthermore, the simulator takes the role of the proxy $\mathcal{P}$ during the computation of the linking number during Outsource.
Thus the proxy no longer leaks $(lin, \mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{a}})$ to $\mathcal{S}_{11}$.
During simulation of Outsource-tasks, linking numbers were created by the simulator and the operator. The simulator $\mathcal{S}_{11}$ still stores $(lin, \mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathsf{o}_\mathbf{a}})$ in $f_{\mathrm{OI}}(pid_{\mathcal{P}}, k)$.
The linking number *lin* is known to $\mathcal{S}_{11}$ due to its participation in the Blum-type coin toss. The shares are known regardless of the user's corruption. If the user is honest, the shares are created by $\mathcal{S}_{11}$ in the first place and can be stored directly. If the user is corrupted, the environment sends the shares to the proxy in the name of the user. This message is visible to the simulator.

H_{12} All honest user $\mathcal{U}$ are replaced by machines $\mathcal{U}'$ that run a similar code as $\mathcal{U}$ with the one exception that during Update-tasks, the actions are restricted to sending the linking number *lin* to the simulator $\mathcal{S}_{12}$. The remaining part of the honest user's protocol for the Update task is played by the simulator.
In the honest user case, $\mathcal{S}_{12}$ fetches $(\mathtt{com}_{\mathbf{a}_z}, \mathrm{unv}_{\mathbf{a}_z}, \mathbf{a}_z, \alpha_z, \mathbf{s}_z, \mathsf{o}_{\alpha z}, \mathsf{o}_{\mathbf{s} z}, \mathsf{o}_{\mathbf{a} z}, \mathsf{o}_{\mathtt{UNV} z}, \mathsf{c}_{out_z})$ from $f_{\mathrm{UP}}(lin)$ and follows the honest protocol of $\mathcal{U}$ from H_{11}.

H_{13} Introduces incorruptible entity $\mathcal{F}^{(\Delta)}_{PUBA}$ that follows the specification from Fig. 6 into the experiment, which is only accessible by honest users and the simulator through subroutine input/output tapes.

H_{14} Replaces the Trusted Signing Authority (TSA) $\mathcal{T}$ with a dummy party that immediately forwards its input to the ideal functionality $\mathcal{F}^{(\Delta)}_{PUBA}$ and leaks $in_{\mathcal{O}}$ to $\mathcal{S}_{14}$. All interactions of $\mathcal{T}$ are simulated by $\mathcal{S}_{14}$ by following the original protocol.

H_{15} Introduces a map f_{FP} to be used by the simulator $\mathcal{S}_{15}$, which maps a task $task \in \{\mathtt{UReg}, BK(K), OA(K)\}$ to a set of tuples $(fp, \mathtt{com}_{fp}, \sigma_{fp})$ of FP and corresponding commitment and signature. During simulation of the task Sign Function Parameter, in case an input *fp* can be used for task *task*, the simulator computes the signature σ_{fp} on $(task, \mathtt{com}_{fp})$ and stores $(\mathtt{com}_{fp}, \sigma_{fp})$ in $f_{\mathrm{FP}}(task)$.
The user $\mathcal{U}$ is replaced by a new user $\mathcal{U}'$, which skips verification of the signature on $(\mathtt{UReg}, \mathtt{com}_{fp})$ and instead asks the simulator if the certificate is valid. Instead of manually verifying the signature, $\mathcal{S}_{15}$ verifies that $(\mathtt{com}_{fp}, \sigma_{fp}) \in f_{\mathrm{FP}}(task)$.

H_{16} During simulation of $\mathcal{F}_{\mathrm{BB}}$ for the Init-Task, if the simulator receives a message $(\mathtt{Register}, pid_{\mathcal{O}}, \mathsf{vk}_{\mathcal{O}})$ from $\mathcal{O}$ to $\mathcal{F}_{\mathrm{BB}}$, he follows the simulation procedure of $\mathcal{F}_{\mathrm{BB}}$ correctly and, if it succeeded, calls $\mathcal{F}^{(\Delta)}_{PUBA}$ with input $(\mathtt{init})$ in the name of $\mathcal{O}$.

H_{17} Replaces all honest user $\mathcal{U}^*$ by dummy parties that immediately forward their input to the ideal functionality $\mathcal{F}^{(\Delta)}_{PUBA}$ with the additional property, that they still leak the linking number *lin* during the Update-task (see H_{12}) and still call $\mathcal{F}_{\mathrm{PPA}}$ with honest inputs when demanded by the protocol; the remaining protocol parts are executed just as specified in the protocol by the simulator $\mathcal{S}_{17}$.
$\mathcal{S}_{17}$ also controls input to $\mathcal{F}^{(\Delta)}_{PUBA}$ for corrupted users during simulation of Outsource- and Update-tasks. During the Outsource-task, $\mathcal{S}_{17}$ calls $\mathcal{F}^{(\Delta)}_{PUBA}$ in the name of $\mathcal{U}$ with empty input after the successful exchange of the linking number *lin*. In the Update-task, $\mathcal{S}_{17}$ calls $\mathcal{F}^{(\Delta)}_{PUBA}$ in the name of $\mathcal{U}$ with input $(\mathtt{Upd})$ after receiving the first message.

H_{18} Replaces the proxies $\mathcal{P}$ with dummy parties that forward their input to the ideal functionality $\mathcal{F}^{(\Delta)}_{PUBA}$. The remaining messages that the proxies sent will

be simulated by the simulator by honestly following the protocol of $\mathcal{P}$ from H_{17}.

H_{19} The simulator now enforces that for every task, $\mathcal{F}_{PUBA}^{(\Delta)}$ is called by the $\mathcal{O}$ with the correct inputs. This causes $\mathcal{F}_{PUBA}^{(\Delta)}$ to have input from all parties, which means that it behaves according to its definition and provides leaks and output to $\mathcal{S}_{19}$, which the simulator can use. Thus, $\mathcal{S}_{19}$ can stop simulating $\mathcal{F}_{\text{PPA}}$ by executing Δ and the appropriate simulator and instead uses the inputs received from $\mathcal{O}$ to $\mathcal{F}_{\text{PPA}}$ in order to call $\mathcal{F}_{PUBA}^{(\Delta)}$ in the name of $\mathcal{O}$. The leaks obtained by $\mathcal{F}_{PUBA}^{(\Delta)}$ are then used as output of $\mathcal{F}_{\text{PPA}}$.
When executing π_{Share}-*Combine* with an honest user and executing the Outsourced Analytics-task with arbitrary user corruption, the simulation of $\mathcal{F}_{\text{PPA}}$ also includes a consistency check for the operator's input: If reconstruction with share via π_{Share}-*Combine* fails, the simulator aborts. The operator shares are obtained via input to $\mathcal{F}_{\text{PPA}}$, the proxy shares are fetched from $f_{\text{OI}}(pid_{\mathcal{P}}, k)$.
When $\mathcal{F}_{PUBA}^{(\Delta)}$ asks $\mathcal{S}_{19}$ for updated inputs $(\mathcal{UH}_z, in_z)$ for corrupted users $\mathcal{U}_z$ during the Outsourced Analytics-task, $\mathcal{S}_{19}$ uses π_{Share}-*Combine* to reconstruct those values using the shares that $\mathcal{U}$ created during Outsource. The proxy-shares, $(\text{sh}_{\mathcal{UH}}^{(\mathcal{P})}, \text{sh}_{in}^{(\mathcal{P})})$, are taken from $f_{\text{OI}}(pid_{\mathcal{P}}, k)$. The operator-shares $\text{sh}_{\mathcal{UH}}^{(\mathcal{O})}$ and $\text{sh}_{in}^{(\mathcal{O})}$ are extracted from the simulation of $\mathcal{F}_{\text{PPA}}$.
The inputs of the operator $\mathcal{O}$ to both the Outsource- and the Update-task do not contain any secrets, so $\mathcal{S}_{19}$ calls $\mathcal{F}_{PUBA}^{(\Delta)}$ in the name of $\mathcal{O}$ after seeing the first message of an Outsource-task and immediately after the start of the simulation of the Update-task.

H_{20} Instead of relying on the leaked input $in_{\mathcal{T}}$ and the received data from the operator to compute whether or not the Function Parameters *fp* should be accepted during the Sign Function Parameter task, $\mathcal{S}_{20}$ accepts the Function Parameters if the functionality returs `ok` to the operator after both parties sent their inputs. The Trusted Signing Authority is replaced with a genuine dummy party which does not leak $in_{\mathcal{T}}$ to the simulator anymore.

H_{21} Instead of relying on the leak of *lin* send by the semi-dummy user $\mathcal{U}$ during simulation of the Update-task for honest users, $\mathcal{S}_{21}$ now uses the leaks on the *ssid* provided by $\mathcal{F}_{PUBA}^{(\Delta)}$ to infer the correct linking number. The honest users are replaced by dummy users, i.e. they only forward their input obtained by $\mathcal{Z}$ to $\mathcal{F}_{PUBA}^{(\Delta)}$. The remaining interactions with the operator are simulated using the simulator's knowledge. Therefore, the simulator uses its mapping from *ssid* to *lin* to get the correct linking number *lin* during the Update-task.

We now show, by a series of lemmata, that no environment $\mathcal{Z}$ can distinguish the real execution (H_1) from the simulated version in the ideal world (H_{21}) by proving indistinguishability of each pair of consecutive games.

Lemma G.2 (Indistinguishability of H_1 and H_2)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_1 and H_2 with probability $1/2 + \varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. We are using the UC composition theorems, here. Letting $\mathcal{S}_2$ emulate $\mathcal{F}_{\text{PPA}}$, $\mathcal{F}_{\text{KE}}$ and $\mathcal{F}_{\text{BB}}$ is possible, as all of them are UC-functionalities. As such, they can be replaced by a protocol, for which a simulator $\mathcal{S}_{\mathcal{F}_{\text{PPA}}}$, $\mathcal{S}_{\mathcal{F}_{\text{KE}}}$ and $\mathcal{S}_{\mathcal{F}_{\text{BB}}}$, respectively, exists that provides an indistinguishable view against *any environment* $\mathcal{Z}_{\mathcal{F}_{\text{PPA}}}$ and $\mathcal{Z}_{\mathcal{F}_{\text{BB}}}$. $\mathcal{S}_{\mathcal{F}_{\text{PPA}}}$ is a probabilistic polynomial time (PPT) algorithm that can be executed internally by $\mathcal{S}_2$. This neither breaks the requirement of $\mathcal{S}_2$ being PPT —as every party executing the protocol can only perform polynomially many calls to $\mathcal{F}_{\text{PPA}}$, for each of which $\mathcal{S}_2$ requires polynomial-time to successfully simulate —nor does it reduce the distinguishing advantage of $\mathcal{Z}$.

Assume for the sake of contradiction that there is an environment $\mathcal{Z}$ that can differentiate an honest execution of $\mathcal{F}_{\text{PPA}}$ in H_1 from a simulated execution of $\mathcal{F}_{\text{PPA}}$ by the simulator $\mathcal{S}_2$ in H_2. We can easily use this environment to build one that can differentiate between an execution of $\mathcal{F}_{\text{PPA}}$ in the real world and an ideal execution, where $\mathcal{S}_{\mathcal{F}_{\text{PPA}}}$ provides the view for $\mathcal{Z}_{\mathcal{F}_{\text{PPA}}}$. This works by *encapsulating* parties: Our new distinguishing environment $\mathcal{Z}_{\mathcal{F}_{\text{PPA}}}$ contains the environment $\mathcal{Z}$ that successfully distinguished H_1 and H_2, a set of users $\{\mathcal{U}\}$, a proxy $\mathcal{P}$ and an operator $\mathcal{O}$, all of which acting according to the protocol specified in H_1. $\mathcal{Z}_{\mathcal{F}_{\text{PPA}}}$ essentially forwards everything that $\mathcal{Z}$ says to the respective parties. After $\mathcal{Z}$ decides on the game, $\mathcal{Z}_{\mathcal{F}_{\text{PPA}}}$ adapts its choice: If $\mathcal{Z}$ outputs H_1, $\mathcal{Z}_{\mathcal{F}_{\text{PPA}}}$ outputs `real` to indicate that this is in the real world. If $\mathcal{Z}$ outputs H_2, $\mathcal{Z}_{\mathcal{F}_{\text{PPA}}}$ outputs `ideal` to indicate that this is an execution in the ideal world.

It is easy to see that the success probability of $\mathcal{Z}_{\mathcal{F}_{\text{PPA}}}$ in deciding whether is in the ideal or real world is equivalent to that of $\mathcal{Z}$ in deciding whether it is playing H_1 or H_2. Hence, if $\mathcal{Z}$ has a non-negligible advantage

over guessing, we found an environment that breaks the UC-security assumption of $\mathcal{F}_{\text{PPA}}$.

Note that the same line of argumentation also works for $\mathcal{F}_{\text{KE}}$ and $\mathcal{F}_{\text{BB}}$, which are also assumed to be UC-secure. □

Lemma G.3 (Indistinguishability of H_2 and H_3)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_2 and H_3 with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. The behavior of the two simulators is exactly identical, only that $\mathcal{S}_3$ has more information; namely the map $f_{\text{UP}}^{(\mathcal{O})}$. Since none of the messages depend on f_{UP}, indistinguishability trivially follows.

Since there is an abort-criteria for the simulator, we also have show that the abort by $\mathcal{S}_3$ in H_3 only occurs iff $\mathcal{P}$ aborts in H_2. Assume that $\mathcal{P}$ and $\mathcal{O}$ in H_2 have respective shares $\text{sh.}^{(\mathcal{P})}$ and $\text{sh.}^{(\mathcal{O})}$. W.l.o.g., assume that the share that caused the abort of $\mathcal{F}_{\text{PPA}}$ in H_2 be that of the user history $\mathcal{UH}$, as the cases for $in_{\mathcal{U}}$, $\mathtt{o}_{\mathtt{OUT}}$, $\mathtt{o}_{\mathtt{UNV}}$, $\mathtt{o}_{\alpha}$, $\mathtt{o}_{\mathbf{s}}$ and $\mathtt{o}_{\mathbf{a}}$ are analogous. In H_2, the two shares are handed over to the subfunctionality $\mathcal{F}_{\text{PPA}}$. There, they are merged with π_{Share}-*Combine*, which aborts if the verification of the shares fails.

The simulator $\mathcal{S}_3$ in H_3 does exactly the same steps; it aborts, iff verification in π_{Share}-*Combine* fails. Hence, the abort criteria are identical, the same code is executed by two different machines. So no environment $\mathcal{Z}$ can distinguish the two games. □

Lemma G.4 (Indistinguishability of H_3 and H_4)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_3 and H_4 with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. Indistinguishability trivially follows from the trapdoor-nature of COM and POK. If any environment $\mathcal{Z}$ could distinguish the execution of the protocol when using *crs* created by $crs \leftarrow \mathsf{SetupPoK}$ from *crs* created by $(crs, td_{\mathsf{sim}}) \leftarrow \mathsf{SetupSPoK}$ with probability $\frac{1}{2}+\varepsilon$, we can build a PPT-environment $\mathcal{Z}'$ that breaks the indistinguishability of the dual-mode property of POK, by having $\mathcal{Z}'$ execute the code of all parties in its head. This leads to the same success probability of $\frac{1}{2}+\varepsilon$, thus causing $\varepsilon \in negl(\kappa)$ by requirement of the chosen POK-scheme. □

Lemma G.5 (Indistinguishability of H_4 and H_5)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_4 and H_5 with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. Any PPT-environment $\mathcal{Z}$ that could distinguish those two games would trivially be able to successfully break the dual-mode property of POK, which is not possible by assumption. □

Lemma G.6 (Indistinguishability of H_5 and H_6)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_5 and H_6 with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. As already stated in the game description the only used value that is visible to the PPT environment is the commitment $\mathtt{com}_{\mathbf{a}}$. The values $\mathbf{a}$ and $\mathtt{unv}_{\mathbf{a}}$ are only used as part of the witness in the Zero-Knowledge proof in the original protocol and are not used since H_5. Hence, the environment can only see the commitment $\mathtt{com}_{\mathbf{a}}$.

For the PPT-environment $\mathcal{Z}$ distinguishing H_5 from H_6 comes down to breaking the hiding-property of the commitment scheme COM which is not possible (for the PPT-environment) by requirement. □

Lemma G.7 (Indistinguishability of H_6 and H_7)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_6 and H_7 with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. First, note that since H_5 the Zero-Knowledge proofs Π are simulated using td_{sim} instead of proving actual properties of the commitments. Hence, the commitments can be exchanged by zero-commitments $\mathsf{COM.Com}(\mathbf{0})$. Other than that, the commitments are only ever used for homomorphic addition, to which the environment $\mathcal{Z}$ only sees committed values. Any PPT-environment $\mathcal{Z}$ that could distinguish the two games H_6 and H_7 would be able to successfully break the hiding property of the commitment scheme COM, which by assumption is only possible with negligible advantage. □

Lemma G.8 (Indistinguishability of H_7 and H_8)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_7 and H_8 with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. The proxy sends the same messages in both games. None of the messages that $\mathcal{S}_8$ sends depend in any way on the leak provided by $\mathcal{P}'$. The leak is hidden from the environment. Hence, the distributions for both games are trivially equivalent. □

Lemma G.9 (Indistinguishability of H_8 and H_9)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_8 and H_9 with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. In both games, the same messages are sent. The simulator also behaves equivalently, as none of the messages that $\mathcal{S}_9$ sends depend on the information stored in f_{OI}. Hence, no (PPT) environment $\mathcal{Z}$ can differentiate the two games. □

Lemma G.10 (Indistinguishability of H_9 and H_{10})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_9 and H_{10} with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. Again, the only difference is that the simulator obtains and stores additional information. As no messages depend on the additional information, the environment $\mathcal{Z}$ is unable to differentiate the two games. □

Lemma G.11 (Indistinguishability of H_{10} and H_{11})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{10} and H_{11} with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. Through π_{Share}-*Share*, the user creates One-Time Pad encrypted inputs for $\mathcal{P}$ and $\mathcal{O}$; the dummy-adversary $\mathcal{D}$ (hence also $\mathcal{Z}$) only ever sees the share of the corrupted operator, not that of the proxy. $\mathcal{Z}$ only has a partial view, which information-theoretically hides the value that is to be shared due to properties of the One-Time Pad. The share itself is not used directly in any further arithmetic computations—only as input to the subfunctionality $\mathcal{F}_{\text{PPA}}$ during the Outsourced Analytics-task. There, the simulator does not work with the shares, but with the actual values to simulate $\mathcal{F}_{\text{PPA}}$ by computing Δ.
Hence, differentiation between shares of x and shares of $\mathbf{0}$ is not possible for any PPT-environment $\mathcal{Z}$ without breaking the security of One-Time Pad encryption.

The computation of the linking number happens via Blum coin toss, where no secrets are involved, meaning that $\mathcal{S}_{11}$ can easily simulate this part by following the protocol honestly. This change is hence only cosmetical; the same code is executed on a different machine. Hence, since the simulator performs the honest computation and does exactly the same as $\mathcal{P}'$ would, the distributions for both games regarding computation of *lin* are equivalent.

Since the shares are created by the simulator, $\mathcal{S}_{11}$ does not have to rely on leaks by $\mathcal{P}'$, but can store them directly, having the same information afterwards. □

Lemma G.12 (Indistinguishability of H_{11} and H_{12})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{11} and H_{12} with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. First, note that honest user leaking information does not change anything in the distribution of sent messages, which makes it impossible for the environment $\mathcal{Z}$ to distinguish. Hence, for indistinguishability, we only have to show that, given the correct linking number *lin*, the simulator for the honest-user case has all the information necessary to report the same messages that $\mathcal{U}$ would have sent in H_{11}.

During the Update-tasks with honest user $\mathcal{U}$ in game H_{11}, the following tasks are now performed by $\mathcal{S}_{12}$:

- Depending on α and $\mathbf{s}$ (both of which are stored in the clear in f_{UP} which is maintained by the simulator since H_3) the simulator either has to:
  - Send $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, \mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \widetilde{\mathtt{com}_{id}}, ser, lin, \Pi, \Pi_{Tr}, \mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}})$ as $\mathcal{U}$ to $\mathcal{O}$:
    All the commitments $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, \mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \widetilde{\mathtt{com}_{id}}$ and $\mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}})$ are commitments to a zero-vector (due to H_7) and both proofs Π and Π_{Tr} are simulated using td_{sim} (due to H_5). The old serial number *ser* is independent of the proof and drawn uniformly random by the simulator (since H_7). The new serial number $\mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}}$ is a zero-commitment (since H_7). Finally, note that the linking number *lin* is leaked by the semi-dummy user. Or, for trivial permutation and direct update vectors:
  - Send $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, \widetilde{\mathtt{com}_{id}}, ser, lin, \Pi, \mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}})$ as $\mathcal{U}$ to $\mathcal{O}$:
    As this message is a subset of the other message its simulatability automatically follows.
- Verifying outputs:
  As the simulator received the message from $\mathcal{O}$ and has the proxy's shares stored in f_{UP}, $\mathcal{S}_{12}$ can follow the honest protocol and abort whenever an honest user would. Hence, we have to show that $\mathcal{S}_{12}$ aborts in H_{12} iff $\mathcal{U}$ aborts in H_{11}.
  Since in both H_{11} and H_{12}, the proxy $\mathcal{P}$ is assumed to be honest, he sends the correct values to the user. Those values are exactly the same as the ones that $\mathcal{S}_{12}$ has stored in f_{UP}. Hence, the simulator already has the user's view on those values. Additionally, the simulator sees messages exchanged between parties; so he also has access to $\mathtt{c}_{out_{\mathcal{U}}}$. Since the simulator essentially follows the honest protocol from here on, the abort-criteria remain equivalent.

□

Lemma G.13 (Indistinguishability of H_{12} and H_{13})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{12} and H_{13} with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. Since there is no direct link between $\mathcal{Z}$ and $\mathcal{F}_{PUBA}^{(\Delta)}$, corrupted parties do not get to access $\mathcal{F}_{PUBA}^{(\Delta)}$. Honest parties only act according to the protocol from H_{12}, which does not contain any interaction with $\mathcal{F}_{PUBA}^{(\Delta)}$. Hence, there is no way for $\mathcal{Z}$ to distinguish the two games, as every action any honest party takes in H_{12} is equivalent to their actions in H_{13} and the corrupted parties act entirely independent of $\mathcal{F}_{PUBA}^{(\Delta)}$. □

Lemma G.14 (Indistinguishability of H_{13} and H_{14})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{13} and H_{14} with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. Indistinguishability easily follows from the fact that the TSA $\mathcal{T}$ leaks its input to the simulator who performs the same code; hence, the simulator can execute the protocol of $\mathcal{T}$ perfectly. Concretely, during the Init-task, the only input to $\mathcal{T}$ is `init`, with the key generation algorithm $\mathsf{SIG}'.\mathsf{Gen}$ is public knowledge, and for the Sign Function Parameter-task, the only inputs are $(\texttt{SFP}, in_{\mathcal{T}})$, and the protocol can be simulated when knowing $\mathsf{sk}_{\mathcal{T}}$ (which $\mathcal{S}_{14}$ does from simulation of the Init-task).

Thus, the distribution visible by $\mathcal{Z}$ is identical and the game hop is purely cosmetical. Our claim follows. □

Lemma G.15 (Indistinguishability of H_{14} and H_{15})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{14} and H_{15} with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. Indistinguishability follows from the EUF-CMA security of SIG': Let $\mathcal{Z}$ be an environment that distinguishes between H_{14} and H_{15} with probability $1/2+\varepsilon$ with *non-negligible* advantage ε. From $\mathcal{Z}$, we construct an adversary $\mathcal{A}$ on the EUF-CMA property of SIG'. Let $\mathcal{C}$ be the EUF-CMA challenger. $\mathcal{C}$ provides a signature oracle to $\mathcal{A}$. $\mathcal{A}$ flips a random coin and simulates either H_{14} or H_{15}. To that end, $\mathcal{A}$ creates all secrets honestly, except for $\mathsf{sk}_{\mathcal{T}}$ and $\mathsf{vk}_{\mathcal{T}}$. On every executiton of the Sign Function Parameter-task, whenever $\mathcal{A}$ is supposed to sign $(task, \texttt{com}_{fp})$ for $\mathcal{O}$ using $\mathsf{sk}_{\mathcal{T}}$, $\mathcal{A}$ forwards $(task, \texttt{com}_{fp})$ to the signature-oracle and uses the result as signature.

Now note that we assume that $\mathcal{Z}$ successfully distinguishes the two games H_{14} and H_{15} notably better than by guessing. Since the only difference is in the way signatures are handled, any distinguishing attack would require $\mathcal{Z}$ to input some distinguishing commitment $\texttt{com}_{fp}$ on Function Parameters fp into the game, which cause a different simulation.

First, note that if the signature on $\texttt{com}_{fp}$ is accepted in H_{15}, it is also accepted in H_{14}, as $(\texttt{com}_{fp}, \sigma_{fp}) \in f_{\text{FP}}(k)$ implies that the Sign Function Parameter task has been called with input $(fp, \texttt{com}_{fp})$ successfully and yielded signature σ_{fp}. However, the other way is not as clear; the only differing behavior that can be caused (and used by $\mathcal{Z}$ to detect the change) is by preparing some tuple $(\texttt{com}_{fp}, \sigma_{fp})$ that is rejected in H_{15} but accepted in H_{14}. Clearly, the latter implies that the signature σ_{fp} on $(task, \texttt{com}_{fp})$ is valid. The former, however, implies that $\mathcal{Z}$ never called the Sign Function Parameter-task on fp for function k in the name of $\mathcal{O}$. This means that (1) $\mathcal{A}$ never called the challenge oracle on input $(task, \texttt{com}_{fp})$, and (2) The signature provided by $\mathcal{Z}$ on $(task, \texttt{com}_{fp})$ verifies. Taking both together makes this a valid forgery, with which $\mathcal{A}$ can break the EUF-CMA property of SIG'. □

Lemma G.16 (Indistinguishability of H_{15} and H_{16})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{15} and H_{16} with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. The change induced in this game only *activates* the functionality $\mathcal{F}_{PUBA}^{(\Delta)}$. Since, at this point, it is not accessed by any honest party and neither the environment, nor the dummy adversary can access $\mathcal{F}_{PUBA}^{(\Delta)}$, indistinguishability between the two games trivially follows. □

Lemma G.17 (Indistinguishability of H_{16} and H_{17})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{16} and H_{17} with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. First, note that the environment $\mathcal{Z}$ is unable to differentiate between the case where the honest user (i.e. the ones that are not controlled by $\mathcal{Z}$) forwards his input to $\mathcal{F}_{PUBA}^{(\Delta)}$ and the one where he does not, as there is no direct line of communication between $\mathcal{F}_{PUBA}^{(\Delta)}$ and either the corrupted parties, or the environment. Hence, this change is impossible to detect for any environment $\mathcal{Z}$.

We claim indistinguishability of the remaining changes based on the fact that the simulator $\mathcal{S}_{17}$ sends exactly the same messages that an honest user would in H_{16}. Using the leaks, the simulator in H_{17} can create every message that the user would have sent in H_{16}, as we will show now:

User Registration. Here, the user has no secret input; the simulator can safely draw a random key similar to the real user and respond to calls to $\mathcal{F}_{\text{BB}}$ with $\mathsf{pk}_{\mathcal{U}}$. The verification step is only based on the messages, which the simulator sees; this means, that $\mathcal{S}_{17}$ can abort in H_{17} whenever $\mathcal{U}$ aborts in H_{16}.

Bookkeeping. The first message that $\mathcal{S}_{17}$ has to send here in the name of $\mathcal{U}$ is $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, ser, \widetilde{\mathtt{com}_{lin}}, \widetilde{\mathtt{com}_{id}}, \Pi_{Val}, \mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}})$.
Since H_7, the commitments are commitments on the all-zero vector and hence independent of the users secret inputs and can be simulated by $\mathcal{S}_{17}$. Since H_5, $\mathcal{S}_{17}$ simulates the zero-knowledge proof Π_{Val}, which is possible without knowing the witness. The old serial number, ser, is independent of both $\mathtt{com}_{\mathcal{UH}}$ and Π_{Val}, and can be drawn uniformly random.
In case any non-trivial permutation α or set vector $\mathbf{s}$ is returned from the simulation of $\mathcal{F}_{\text{PPA}}$ and hence the computation of the application-specific function Δ, $\mathcal{S}_{17}$ has to report a second message in which it proves correct update of the User History. This results in the message $(\mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \Pi_{Tr})$. The commitments are once again all-zero commitments due to H_7. The proof Π_{Tr} is simulated due to H_5 but requires knowledge of the full statement. This statement contains $\widetilde{\mathtt{com}_{\mathcal{UH}}}, \mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}$ (which were chosen as commitments on the all-zero vector due to H_7), and α and $\mathbf{s}$ (which are also required in order to decide if the second message has to be reported at all; note that both α and $\mathbf{s}$ are known by $\mathcal{O}$ and hence can not be forged or set arbitrarily). The latter is obtained by simulation of $\mathcal{F}_{\text{PPA}}$ and hence consistent with the operators view. Hence $\mathcal{S}_{17}$ can reconstruct the statement of the second proof according to L^{Tr}_{B} which proves that the output of $\mathcal{F}_{\text{PPA}}$ has been transferred to the UH accordingly. Again, we stress that this message is only reported for non-trivial values of α and $\mathbf{s}$.
Finally, the simulator has to perform the verification step, as $\mathcal{S}_{17}$ always used zero-vectors for shares and can hence verify the values received from $\mathcal{O}$. This is done by executing the honest protocol, namely by calling π_{Verify} with an honestly created logbook λ.

Outsource. During an Outsource-task, the shares have been created since H_{11} by the simulator anyways. Since H_5 and H_7 the simulator also creates the commitments and the zero-knowledge proof. The serial is again drawn at random since H_7 as it is independent of $\mathtt{com}_{ser}(= \mathtt{com}_{\mathbf{0}})$. Hence, the first message, $(\mathtt{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathtt{sh}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{OUT}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{UNV}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_\alpha}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathbf{s}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathbf{a}}}, \mathtt{com}^{(\mathcal{O})}_{\mathcal{UH}}, ser, \widetilde{\mathtt{com}_{id}}, \Pi, \mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}})$ is indistinguishable from the one in H_{16}.
Verification, again, is only dependent on what the simulator knows already and hence can be simulated by executing the honest protocol.

Update. The Update-task has been simulated already since H_{12}.

For security against a corrupted user $\mathcal{U}$ our definition of $\mathcal{F}^{(\Delta)}_{PUBA}$ requires $\mathcal{S}_{17}$ to call $\mathcal{F}^{(\Delta)}_{PUBA}$ in the name of $\mathcal{U}$ only during the Outsource-task and not during any of the other tasks:

- In **User Registration**, all that would change is that $\mathcal{F}^{(\Delta)}_{PUBA}$ adds $\mathcal{U}$ to the list of known users, which is never checked against corrupted users. Interaction with $\mathcal{F}_{\text{BB}}$ might still take place, though, but that one is simulated since H_2.
- In the **Bookkeeping**-task the environment essentially talks to itself. Updates to $\mathcal{UH}$ can be done by $\mathcal{Z}$ without access to $\mathcal{F}^{(\Delta)}_{PUBA}$ as the operator can sign any User History $\mathcal{UH}$ and thus create a valid new logbook λ. Simulation of $\mathcal{F}_{\text{PPA}}$ occurs outside of the actual protocol since $\mathcal{S}_{17}$ plays $\mathcal{S}_{\mathcal{F}_{\text{PPA}}}$ since H_2.
- For **Outsource** $\mathcal{F}^{(\Delta)}_{PUBA}$ stores the corrupted user's information in a list, alongside that of honest users, and which are used later for the Outsourced Analytics-task. There, it does not make a difference if the corrupted user's $\mathcal{UH}$ is input to $\mathcal{F}^{(\Delta)}_{PUBA}$ directly, or if it is equivocated during simulation of the subsequent Outsourced Analytics task.
- During **Outsourced Analytics** the simulator receives input from $\mathcal{O}$ to $\mathcal{F}_{\text{PPA}}$ which contains the shares the corrupted user $\mathcal{U}$ prepared for the operator: $(OA(k), \{(\mathtt{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathtt{sh}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{OUT}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{UNV}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_\alpha}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathbf{s}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathbf{a}}})_z\}^{Z_k}_{z=1}, fp, \mathtt{unv}_{fp}, in_{\mathcal{O}})$. The respective shares of the proxy were already stored in $f_{\text{OI}}(pid_{\mathcal{P}}, k)$ during simulation of the Outsource-task. Hence all shares can be reconstructed by $\mathcal{S}_{17}$ using the protocol π_{Share}-*Combine*. This value is equivalent to the input corrupted users that would have input to $\mathcal{F}^{(\Delta)}_{PUBA}$. So $\mathcal{S}_{17}$ can provide $\mathcal{F}^{(\Delta)}_{PUBA}$ with the correct tuples $(\mathcal{UH}, in)$ for corrupted users.
Correctness trivially follows as the simulator in H_{17} has, at this point, exactly the same values that are divided to $\mathcal{P}$ and $\mathcal{O}$ in H_{16}. Hence, any attempt by $\mathcal{Z}$ to manipulate data that would have worked in H_{16} also works in H_{17} and vice versa, making it impossible for any PPT-environment $\mathcal{Z}$ to distinguish.

– During the **Update**-task corrupted users only obtain masked values from the proxy. This step does not have any consequences for further interaction as $\mathcal{F}_{PUBA}^{(\Delta)}$ only loads and returns data that is not accessed at any further point throughout the lifetime of the system. Hence, it suffices to call $\mathcal{F}_{PUBA}^{(\Delta)}$ in the name of *any* corrupted user to have it deliver output to the Proxy.

□

Lemma G.18 (Indistinguishability of H_{17} and H_{18})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{17} and H_{18} with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. Intuitively, the proxy does not have any secret inputs. It only obtains information via secret shares of the user. At this point, the simulator $\mathcal{S}_{18}$ already simulates all the messages sent by the user and hence can act as the honest proxy would by following the protocol from H_{17}.

In more detail, the situation for an *honest* user is as follows:

Outsource. Since the proxy is now played by the simulator who also sends all messages on behalf of the user, all messages from user to proxy and vice versa can be ignored. This automatically resolves most of the Outsource-task; the only interaction between the proxy and the operator that has to be simulated is during a Blum coin toss to create *lin*, which does not depend on any secret inputs at all. This can honestly be executed by $\mathcal{S}_{18}$.
Managing f_{OI} is also trivially possible, as all the values stored there come from the user, who is played by the simulator at this point, anyway. Correctness of the values follows from the fact that $\mathcal{S}_{18}$ only stores information there in H_{18} when the honest proxy $\mathcal{P}$ in H_{17} would.

Outsourced Analytics. In H_{17}, the proxy loads the first Z_k entries of f_{OI} before calling $\mathcal{F}_{\mathrm{PPA}}$. We already argued for the Outsource-task that the information $\mathcal{S}_{18}$ stores in f_{OI} in H_{18} is equivalent to what $\mathcal{P}$ stores during H_{17}; hence, $\mathcal{S}_{18}$ can simulate by following the honest protocol.
Storing the information in $f_{\mathrm{UP}}^{(\mathcal{P})}$ after the simulation of $\mathcal{F}_{\mathrm{PPA}}$ is not required anymore, as it only contains information that the simulator can infer from f_{UP}.

Update. The protocol for the Update-task basically consists of two mostly disjoint parts: the interaction between the user and the proxy and the interaction between the user and the operator.
The latter is independent of anything the proxy does. The former is independent of anything the environment $\mathcal{Z}$ (or the dummy-adversary $\mathcal{D}$) do and does not have to be simulated, as $\mathcal{Z}$ does not have the ability to read messages exchanged between honest parties.

If the user is corrupted, this changes the behavior for the Outsource- and Update-tasks as follows:

Outsource. Here, everything works by following the protocol of $\mathcal{P}$ honestly. The simulator receives the shares, which are send from $\mathcal{U}$ to $\mathcal{P}$. Those can be stored in $f_{\mathrm{OI}}(pid_{\mathcal{P}}, k)$.
Since the simulator now also performs the Blum coin toss, $\mathcal{S}_{18}$ knows *lin* and can send it to the user directly.

Update. The only interaction that has to be simulated is the part where $\mathcal{P}$, after receiving *lin* from $\mathcal{U}$, sends $(\mathsf{c}_{\alpha}, \mathsf{c}_{\mathbf{s}}, \mathsf{c}_{\mathbf{a}}, \mathsf{c}_{\mathrm{unv}_{\mathbf{a}}}, \mathsf{c}_{out_{\mathcal{U}}})$ to $\mathcal{U}$. During simulation of the task for Outsourced Analytics both values were created honestly and stored in $f_{\mathrm{UP}}(lin)$. Since *lin* is obtained from $\mathcal{U}$, the proxy in H_{17} will send exactly the same message as the simulator in H_{18}.

This shows that the two games are indistinguishable for all PPT-environments $\mathcal{Z}$. □

Lemma G.19 (Indistinguishability of H_{18} and H_{19})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{18} and H_{19} with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. Indistinguishability of the first change, namely that computations of Δ are replaced by leaks from $\mathcal{F}_{PUBA}^{(\Delta)}$, follows from the following facts:

1. In both hybrid games, the result of the computation is based on the output of Δ. The simulator performs this computation in H_{18}; there, it acts honestly according to the simulator description, hence $\mathcal{S}_{19}$ would not cheat. The functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ is, by UC-conventions, modeled as an incorruptible entity and hence performs the same honest computation. Hence, both parties compute Δ honestly.
2. Both parties, $\mathcal{F}_{PUBA}^{(\Delta)}$ in H_{19} and the simulator in H_{18}, use exactly the same input. Until the point where the interaction takes place, the simulators in both H_{18} and H_{19} behave equivalently, leading to

an identical view for $\mathcal{Z}$. From there on, the simulator in H_{18} performs the computation $\Delta(\cdot)$ directly and obtains the outputs. In H_{19}, $\mathcal{S}_{19}$ forwards the same input to $\mathcal{F}_{PUBA}^{(\Delta)}$ (which is possible because both have the same interface). The functionality then uses this input in order to compute Δ. The outputs are leaked to $\mathcal{S}_{19}$ via functionality output, who now has exactly the same values as in H_{18} and can continue equivalently.

Hence, the parts involving interaction with $\mathcal{F}_{\text{PPA}}$ remain equivalent, as essentially the output data that $\mathcal{S}_{19}$ obtains in H_{19} via leakage has been created similarly to the output that the simulator computed in H_{18}.

We now have to show that the abort-criteria remain equivalent. In H_{19}, $\mathcal{S}_{19}$ aborts if the input of $\mathcal{O}$ to $\mathcal{F}_{\text{PPA}}$ differs from the shares that $\mathcal{U}$ sent to $\mathcal{O}$ during the Outsource-task. In H_{18}, the user aborts via $\mathcal{F}_{\text{PPA}}$, as shares that were changed by the operator would be recognized as forgery due to π_{Verify} with overwhelming probability.

Next, we have to prove indistinguishability of the equivocation step for corrupted users. To that end, we claim that the updated input shares that $\mathcal{S}_{19}$ inputs to $\mathcal{F}_{PUBA}^{(\Delta)}$ are the same that were shared during the Outsource-task and that would have been used by the proxy and operator in a real execution. The proxy-shares were treated similar to a real execution, as the simulator stored the message that was received during the Outsource-task in f_{OI}. The operator shares were taken from $\mathcal{O}$s input to $\mathcal{F}_{\text{PPA}}$ and are hence also visible to the simulator. Hence, the same values that would have been taken as input for Δ in H_{18} are also taken by $\mathcal{F}_{PUBA}^{(\Delta)}$ in H_{19}. As mentioned above, $\mathcal{F}_{PUBA}^{(\Delta)}$ computes Δ just as was done in previous games, so no environment can distinguish. □

Lemma G.20 (Indistinguishability of H_{19} and H_{20})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{19} and H_{20} with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. Again this gamehop is only cosmetical as the same code still is executed on the same inputs, but from a different machine. In H_{19} the simulator obtains $in_{\mathcal{T}}$ from the (semi-)dummy TSA and $(fp, \mathtt{com}_{fp}, \mathtt{unv}_{fp}, task, in_{\mathcal{O}})$ from the corrupted operator and then performs some consistency checks regarding the FPs before evaluating Δ. In H_{20} the simulator only obtains $(fp, \mathtt{com}_{fp}, \mathtt{unv}_{fp}, task, in_{\mathcal{O}})$ from the corrupted operator and then performs the same consistency checks regarding the FPs before inputting $(fp, task, in_{\mathcal{O}})$ to $\mathcal{F}_{PUBA}^{(\Delta)}$. The input from the TSA—$in_{\mathcal{T}}$—was already forwarded by the honest dummy party to $\mathcal{F}_{PUBA}^{(\Delta)}$ so the inputs are the same.

Indistinguishability thus follows directly. □

Lemma G.21 (Indistinguishability of H_{20} and H_{21})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{20} and H_{21} with probability $1/2+\varepsilon$. It holds that $\varepsilon \in negl(\kappa)$.*

Proof. The situation only changes during Update-tasks with an honest user, where the simulator $\mathcal{S}_{21}$ in the honest-user setting does not receive the leak *lin* from $\mathcal{U}$, but instead the Subsession Identifier *ssid* of the respective Outsource-instance from $\mathcal{F}_{PUBA}^{(\Delta)}$. Since the simulator updated the list f_{LN} correctly during the simulation of the Outsource-task, $\mathcal{S}_{21}$ can obtain the same linking number *lin* in H_{21} that $\mathcal{U}$ has sent in H_{20}.

Given that $\mathcal{F}_{PUBA}^{(\Delta)}$ is, by definition, incorruptible, it will always send the correct Subsession Identifier *ssid* to the simulator $\mathcal{S}_{21}$. During the Outsource-task, the linking number *lin* was honestly created by the simulator $\mathcal{S}_{21}$ in an interaction with $\mathcal{O}$. This number is used by all parties as linking number and is stored in f_{LN} just as honest parties would store it. During the Update-task, an honest user would look up his linking number, which can be simulated by following the program of $\mathcal{U}$ and fetching it from f_{LN}.

Hence, there is no way that the linking number an honest user would store during an Outsource- and later reveal during an Update-task in H_{20} would differ from the linking number that the simulator stores (and later reveals) during the simulation of H_{21}. □

The final game, H_{21}, corresponds to our ideal world. Since we have shown that no efficient environment $\mathcal{Z}$ can differentiate this from the real execution in H_1, our corollary follows:

Corollary G.22 (User Security)**.** *For all environments $\mathcal{Z}$ who statically corrupted the operator, it follows that*

$$\pi_{PUBA}^{(\mathcal{F}_{\text{PPA}},\mathcal{F}_{\text{BB}},\mathcal{F}_{\text{KE}},\mathcal{F}_{\text{CRS}})} \geq_{UC} \mathcal{F}_{PUBA}^{(\Delta)}$$

We have shown in Lemma G.2 to Lemma G.21, that under static corruption of the operator, the simulator $\mathcal{S}_{\text{USec}}$ acting in the ideal world can provide a view for $\mathcal{Z}$ that is indistinguishable from a real execution of the protocol:

$$view_{\mathcal{Z},\mathcal{A},\pi_{PUBA}} \approx^{c} view_{\mathcal{Z},\mathcal{S}_{\text{USec}},\mathcal{F}_{PUBA}^{(\Delta)}}$$

G.2 System Security

This section contains an investigation of the remaining corruption scenarios, namely the ones that are relevant to maintain privacy of an honest *operator*. That is, we consider scenarios where the any subset of users and proxies can be corrupted and present a simulator, which provides a view in the ideal world that cannot be distinguished from a real-world execution.

Simulator $\mathcal{S}_{\mathbf{SysSec}}$ for corrupted users and proxies

π-*Shared State*:

- Trapdoor td_{ext} for extracting proofs.
- Signature key pair $(\mathsf{vk}_{\mathcal{O}}, \mathsf{sk}_{\mathcal{O}})$
- Signature key pair $(\mathsf{vk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$
- List $\mathsf{L}_{\mathsf{SER}}$ of observed serial numbers.
- Partial mapping f_{ID} on $\mathbb{G}_1$: $\mathsf{pk}_{\mathcal{U}} \mapsto pid_{\mathcal{U}}$ that maps user public keys $\mathsf{pk}_{\mathcal{U}}$ to the pid of the corresponding user
- Mapping $f_{\mathrm{OI}}^{(\mathcal{P})}$ on $\{1,\dots,K\}$ that maps k to a list $f_{\mathrm{OI}}^{(\mathcal{P})}(k)$ of entries $(lin, \mathrm{sh}_{\mathcal{UH}}^{(\mathcal{P})}, \mathrm{sh}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o}_{\mathtt{OUT}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o}_{\mathtt{UNV}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o}_{\alpha}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o}_{\mathbf{s}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o}_{\mathbf{a}}}^{(\mathcal{P})})$ for every honest proxy $\mathcal{P}$
- Mapping $f_{\mathrm{OI}}^{(\mathcal{O})}$ on $\{pid_{\mathcal{P}}\}\times\{1,\dots,K\}$ that maps $(pid_{\mathcal{P}}, k)$ to a list $f_{\mathrm{OI}}^{(\mathcal{O})}(pid_{\mathcal{P}}, k)$ of entries $(lin, \mathrm{sh}_{\mathcal{UH}}^{(\mathcal{O})}, \mathrm{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o}_{\mathtt{OUT}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o}_{\mathtt{UNV}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o}_{\alpha}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o}_{\mathbf{s}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o}_{\mathbf{a}}}^{(\mathcal{O})})$
- One partial mapping $f_{\mathrm{UP}}^{(\mathcal{P})}$ on $\{lin\}$ with $lin \mapsto (\mathtt{c}_{\alpha}, \mathtt{c}_{\mathbf{s}}, \mathtt{c}_{\mathbf{a}}, \mathtt{c}_{\mathtt{unv}_{\mathbf{a}}}, \mathtt{c}_{out_{\mathcal{U}}})$ for every honest proxy $\mathcal{P}$
- Partial mapping $f_{\mathrm{UP}}^{(\mathcal{O})}$ on $\{lin\}$. $lin \mapsto (\alpha, \mathbf{s}, \mathtt{com}_{\mathbf{a}}, \mathtt{c}_{out_{\mathcal{U}}})$
- Partial mapping f_{LN} on $\{ssid\}$: $ssid \mapsto lin$
- Mapping f_{FP} on $(\{\mathtt{UReg}\cup BK(K)\cup OA(K)\}\times L)$ that maps a task $task$ and a FP-index ℓ to a tuple $(\mathtt{com}_{fp}, \mathtt{unv}_{fp}, \sigma_{fp})$ of a commitment and a signature.

π-*Setup*:

1. Run a modified version of $crs \leftarrow \pi\text{-}Setup(1^{\kappa})$
   - $crs_{\mathsf{pok}} \leftarrow \mathsf{SetupPoK}$ is replaced by $(crs_{\mathsf{pok}}, td_{\mathsf{ext}}) \leftarrow \mathsf{SetupEPoK}$ for extracting proofs.

π-*Init*:

1. Upon receiving $(\mathtt{init}, pid_{\mathcal{O}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - Create and store a signature-key pair $(\mathsf{sk}_{\mathcal{O}}, \mathsf{vk}_{\mathcal{O}}) \leftarrow \mathsf{SIG.Gen}(1^{\kappa})$
   - From now on, reply to $\mathcal{F}_{\mathrm{BB}}$ calls of the form $(\mathtt{Retrieve}, pid_{\mathcal{O}})$ with $\mathsf{vk}_{\mathcal{O}}$

   ↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to continue.
2. Upon receiving $(\mathtt{init}, pid_{\mathcal{T}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - Create and store a signature-key pair $(\mathsf{sk}_{\mathcal{T}}, \mathsf{vk}_{\mathcal{T}}) \leftarrow \mathsf{SIG'.Gen}(1^{\kappa})$
   - From now on, reply to $\mathcal{F}_{\mathrm{BB}}$ calls of the form $(\mathtt{Retrieve}, pid_{\mathcal{T}})$ with $\mathsf{vk}_{\mathcal{T}}$

   ↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to continue.

π-*Sign Function Parameter*:

. . . . . .

$\mathcal{O}$ honest, $\mathcal{T}$ honest:

1. Upon receiving leak $(task, \ell)$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - Compute $(\mathtt{com}_{\mathbf{0}}, \mathtt{unv}_{\mathbf{0}}) \leftarrow \mathsf{COM'.Com}(\mathbf{0})$.
   - Compute $\sigma_{fp} \leftarrow \mathsf{SIG.Sgn}((task, \mathtt{com}_{\mathbf{0}}), \mathsf{sk}_{\mathcal{T}})$.
   - Store $f_{\mathrm{FP}}(task, \ell) \coloneqq (\mathtt{com}_{\mathbf{0}}, \mathtt{unv}_{fp}, \sigma_{fp})$.

π-*User Registration*:

. . . . . .

$\mathcal{U}$ and $\mathcal{O}$ honest:

1. Draw random user public key $\mathsf{pk}_{\mathcal{U}} \xleftarrow{\mathrm{r}} \mathbb{G}_1$
2. From now on, reply to $\mathcal{F}_{\mathrm{BB}}$ calls of the form $(\mathtt{Retrieve}, pid_{\mathcal{U}})$ with $\mathsf{pk}_{\mathcal{U}}$
3. Report encrypted messages.

. . . . . .

$\mathcal{U}$ corrupted, $\mathcal{O}$ honest:

1. Upon receiving instructions from $\mathcal{Z}$ to send $(\mathtt{Retrieve}, pid_{\mathcal{O}})$ from $\mathcal{U}$ to $\mathcal{F}_{\mathrm{BB}}$:

   ↪ Report output $(\mathsf{vk}_{\mathcal{O}})$ from $\mathcal{F}_{\mathrm{BB}}$ to $\mathcal{U}$.
2. Upon receiving instructions from $\mathcal{Z}$ to send $(\mathtt{Register}, pid_{\mathcal{U}}, \mathsf{pk}_{\mathcal{U}})$ from $\mathcal{U}$ to $\mathcal{F}_{\mathrm{BB}}$:
   - If a previous call $(\mathtt{Register}, pid_{\mathcal{U}}, \mathsf{pk}'_{\mathcal{U}})$ to $\mathcal{F}_{\mathrm{BB}}$ for some $\mathsf{pk}'_{\mathcal{U}}$ has been previously recorded, abort in the name of $\mathcal{O}$.
3. Upon receiving leak ℓ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - Set[a] $(\mathtt{com}_{\mathbf{0}}, \mathtt{unv}_{\mathbf{0}}, \sigma_{fp}) \coloneqq f_{\mathrm{FP}}(\mathtt{UReg}, \ell)$.
4. Upon receiving instructions from $\mathcal{Z}$ to send $(\Pi, \mathtt{com}_{id}, \mathtt{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})})$ from $\mathcal{U}$ to $\mathcal{O}$:

– If $f_{\mathrm{ID}}(\mathsf{pk}_{\mathcal{U}}) \neq \perp$ abort, else set $f_{\mathrm{ID}}(\mathsf{pk}_{\mathcal{U}}) = pid_{\mathcal{U}}$
– $stmt \coloneqq (\mathsf{pk}_{\mathcal{U}}, \mathtt{com}_{id})$
– Check[a] $\mathsf{POK.Vfy}(\Pi, stmt, L_{\mathrm{R}}) \stackrel{?}{=} 1$

↪ Report message $(\mathtt{com}_{\mathbf{0}}, \sigma_{fp})$ from $\mathcal{O}$ to $\mathcal{U}$.

5. Upon receiving instructions from $\mathcal{Z}$ to send $(\mathtt{UReg}, \mathtt{com}_{fp}, in_{\mathcal{U}})$ from $\mathcal{U}$ to $\mathcal{F}_{\mathrm{PPA}}$:
   – Check[a] $\mathsf{COM.Unv}(\mathtt{com}_{fp}, \mathtt{unv}_{\mathbf{0}}, \mathbf{0}) = 1$.

   ↪ Call $\mathcal{F}_{PUBA}^{(\Delta)}$ with input $(\mathtt{UReg}, in_{\mathcal{U}})$ in the name of $\mathcal{U}$.
6. Upon receiving output $(\mathcal{UH}, out_{\mathcal{U}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   – $(\mathtt{com}_{\mathcal{UH}}, \mathtt{unv}_{\mathcal{UH}}) \leftarrow \mathsf{COM.Com}(\mathcal{UH})$
   – $(ser^{\mathrm{new}})^{(\mathcal{O})} \xleftarrow{\mathrm{r}} \mathbb{Z}_p$
   – $(\mathtt{com}^{(\mathcal{O})}_{ser^{\mathrm{new}}}, \mathtt{unv}^{(\mathcal{O})}_{ser^{\mathrm{new}}}) \leftarrow \mathsf{COM.Com}((ser^{\mathrm{new}})^{(\mathcal{O})})$
   – $\mathtt{com}_{ser^{\mathrm{new}}} \coloneqq \mathtt{com}^{(\mathcal{O})}_{ser^{\mathrm{new}}} \oplus \mathtt{com}^{(\mathcal{U})}_{ser^{\mathrm{new}}}$
   – $(\mathtt{com}_{lin}, \mathtt{unv}_{lin}) \leftarrow \mathsf{COM.Com}(0)$
   – $\sigma \leftarrow \mathsf{SIG.Sgn}(\mathsf{sk}_{\mathcal{O}}, \mathtt{com}_{\mathcal{UH}} \| \mathtt{com}_{ser^{\mathrm{new}}} \| \mathtt{com}_{lin} \| \mathtt{com}_{id})$

   ↪ Report output $(\mathcal{UH}, \mathtt{unv}_{\mathcal{UH}}, out_{\mathcal{U}})$ from $\mathcal{F}_{\mathrm{PPA}}$ to $\mathcal{U}$.

   ↪ Report message $(\mathtt{com}_{\mathcal{UH}}, (ser^{\mathrm{new}})^{(\mathcal{O})}, \mathtt{com}^{(\mathcal{O})}_{ser^{\mathrm{new}}}, \mathtt{unv}^{(\mathcal{O})}_{ser^{\mathrm{new}}}, \mathtt{com}_{lin}, \mathtt{unv}_{lin}, \mathtt{com}_{id}, \mathtt{unv}_{id}, \sigma)$ from $\mathcal{O}$ to $\mathcal{U}$.

   ↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to continue.

$\left\{\pi\text{-}Bookkeeping^{(k)}\right\}_{k=1}^{K}$:

$\mathcal{U}$ and $\mathcal{O}$ honest: Report encrypted messages.

$\mathcal{U}$ corrupted, $\mathcal{O}$ honest:

1. Upon receiving leak (ℓ) from $\mathcal{F}_{PUBA}^{(\Delta)}$ and instructions from $\mathcal{Z}$ to send $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, ser, \widetilde{\mathtt{com}_{lin}}, \widetilde{\mathtt{com}_{id}}, \Pi_{Val}, \mathtt{com}^{(\mathcal{U})}_{ser^{\mathrm{new}}})$ from $\mathcal{U}$ to $\mathcal{O}$:
   – $stmt^{Val} \coloneqq (\widetilde{\mathtt{com}_{\mathcal{UH}}}, ser, \widetilde{\mathtt{com}_{lin}}, \widetilde{\mathtt{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
   – Check[a] $\mathsf{POK.Vfy}(\Pi_{Val}, stmt^{Val}, L_{\mathrm{B}}^{Val}) \stackrel{?}{=} 1$
   – Check[a] $ser \notin \mathsf{L_{SER}}$
   – $\mathsf{L_{SER}} \coloneqq \mathsf{L_{SER}} \cup \{ser\}$
   – Set[a] $(\mathtt{com}_{\mathbf{0}}, \mathtt{unv}_{\mathbf{0}}, \sigma_{fp}) \coloneqq f_{\mathrm{FP}}(BK(k), \ell)$.

   ↪ Report message $(\mathtt{com}_{\mathbf{0}}, \sigma_{fp})$ from $\mathcal{O}$ to $\mathcal{U}$.

2. Upon receiving $(BK(k), \mathcal{UH}, \widetilde{\mathtt{unv}_{\mathcal{UH}}}, \mathtt{com}_{fp}, in_{\mathcal{U}})$ from $\mathcal{U}$ to $\mathcal{F}_{\mathrm{PPA}}$:
   – Check[a] $\mathsf{COM.Unv}(\mathtt{com}_{fp}, \mathtt{unv}_{\mathbf{0}}, \mathbf{0}) = 1$.
   – Check[a] $\mathsf{COM.Unv}(\widetilde{\mathtt{com}_{\mathcal{UH}}}, \widetilde{\mathtt{unv}_{\mathcal{UH}}}, \mathcal{UH}) \stackrel{?}{=} 1$
   – Extract $\mathsf{pk}_{\mathcal{U}}$ using $\mathsf{POK.ExtractWit}(td_{\mathsf{ext}}, \Pi_{Val}, stmt^{Val}, L_{\mathrm{B}}^{Val})$
   – Load[a] $pid_{\mathcal{U}'} \coloneqq f_{\mathrm{ID}}(\mathsf{pk}_{\mathcal{U}})$
   – Choose $\mathcal{U}'$ as the user corresponding to $pid_{\mathcal{U}'}$

   ↪ Call Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ with input $(BK(k), in_{\mathcal{U}})$ in the name of $\mathcal{U}'$
3. Upon receiving output $(\alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ to $\mathcal{U}$:
   – Compute $(\mathtt{com}_{\mathbf{a}}, \mathtt{unv}_{\mathbf{a}}) \leftarrow \mathsf{COM.Com}(\mathbf{a})$

   ↪ Report message $(\alpha, \mathbf{s}, \mathbf{a}, \mathtt{com}_{\mathbf{a}}, \mathtt{unv}_{\mathbf{a}}, out_{\mathcal{U}})$ from $\mathcal{F}_{\mathrm{PPA}}$ to $\mathcal{U}$
4. **if** $\alpha \neq \perp \vee \mathbf{s} \neq \perp$ **then**
   – Upon receiving instructions from $\mathcal{Z}$ to send $(\mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \Pi_{Tr})$ from $\mathcal{U}$ to $\mathcal{O}$:
     – $stmt^{Tr} \coloneqq (\widetilde{\mathtt{com}_{\mathcal{UH}}}, \mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \alpha, \mathbf{s})$
     – Check[a] $\mathsf{POK.Vfy}(\Pi_{Tr}, stmt^{Tr}, L_{\mathrm{B}}^{Tr}) \stackrel{?}{=} 1$

   fi
5. Compute new User History.
   – $\mathtt{com}^{\mathrm{new}}_{\mathcal{UH}} \coloneqq \mathtt{com}''_{\mathcal{UH}} \oplus \mathtt{com}_{\mathbf{a}}$
   – $(ser^{\mathrm{new}})^{(\mathcal{O})} \xleftarrow{\mathrm{r}} \mathbb{Z}_p$
   – $(\mathtt{com}^{(\mathcal{O})}_{ser^{\mathrm{new}}}, \mathtt{unv}^{(\mathcal{O})}_{ser^{\mathrm{new}}}) \leftarrow \mathsf{COM.Com}((ser^{\mathrm{new}})^{(\mathcal{O})})$
   – $\mathtt{com}^{\mathrm{new}}_{ser} \coloneqq \mathtt{com}^{(\mathcal{O})}_{ser^{\mathrm{new}}} \oplus \mathtt{com}^{(\mathcal{U})}_{ser^{\mathrm{new}}}$
   – $\sigma^{\mathrm{new}} \leftarrow \mathsf{SIG.Sgn}(\mathsf{sk}_{\mathcal{O}}, \mathtt{com}^{\mathrm{new}}_{\mathcal{UH}} \| \mathtt{com}^{\mathrm{new}}_{ser} \| \widetilde{\mathtt{com}_{lin}} \| \widetilde{\mathtt{com}_{id}})$

   ↪ Report message $(\mathtt{com}^{\mathrm{new}}_{\mathcal{UH}}, (ser^{\mathrm{new}})^{(\mathcal{O})}, \mathtt{com}^{(\mathcal{O})}_{ser^{\mathrm{new}}}, \mathtt{unv}^{(\mathcal{O})}_{ser^{\mathrm{new}}}, \sigma^{\mathrm{new}})$ from $\mathcal{O}$ to $\mathcal{U}$
   ↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to deliver output to $\mathcal{O}$

$\left\{\pi\text{-}Outsource^{(k)}\right\}_{k=1}^{K}$:

$\mathcal{U}$, $\mathcal{P}$ and $\mathcal{O}$ honest:

1. Upon receiving $(OS(k), \mathrm{User})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$, $(OS(k), pid_{\mathcal{P}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ and $(OS(k), pid_{\mathcal{O}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   – Append an empty entry $(\perp, \ldots, \perp)$ to $f_{\mathrm{OI}}^{(\mathcal{P})}(k)$
   – Append an empty entry $(\perp, \ldots, \perp)$ to $f_{\mathrm{OI}}^{(\mathcal{O})}(pid_{\mathcal{P}}, k)$

   ↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to continue.

$\mathcal{U}$ corrupted, $\mathcal{P}$ and $\mathcal{O}$ honest:

1. Upon receiving $(OS(k), pid_{\mathcal{O}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ and $(OS(k), pid_{\mathcal{P}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ and instructions from $\mathcal{Z}$ to send $(\mathrm{sh}_{\mathcal{UH}}^{(\mathcal{O})}, \mathrm{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o_{OUT}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o_{UNV}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o}_\alpha}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o_s}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o_a}}^{(\mathcal{O})}, ser, \widetilde{\mathrm{com}_{id}}, \Pi, \mathrm{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})})$ from $\mathcal{U}$ to $\mathcal{O}$ and $(\mathrm{sh}_{\mathcal{UH}}^{(\mathcal{P})}, \mathrm{sh}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_{OUT}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_{UNV}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o}_\alpha}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_s}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_a}}^{(\mathcal{P})})$ from $\mathcal{U}$ to $\mathcal{P}$:
   - $stmt \coloneqq (\mathrm{sh}_{\mathcal{UH}}^{(\mathcal{O})}, \mathrm{com}_{\mathcal{UH}}^{(\mathcal{O})}, ser, \widetilde{\mathrm{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
   - Check[a] $\mathsf{POK.Vfy}(\Pi, stmt, L_{\mathrm{O}}) \stackrel{?}{=} 1$
   - Check[a] $ser \notin \mathsf{L_{SER}}$
   - $\mathsf{L_{SER}} \coloneqq \mathsf{L_{SER}} \cup \{ser\}$
   - Set[b] $\mathcal{UH} \coloneqq \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathcal{UH}}^{(\mathcal{O})}, \mathrm{sh}_{\mathcal{UH}}^{(\mathcal{P})})$
   - Set[b] $in_{\mathcal{U}} \coloneqq \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}, \mathrm{sh}_{in_{\mathcal{U}}}^{(\mathcal{P})})$
   - Set[b] $\mathrm{o_{OUT}} \coloneqq \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathrm{o_{OUT}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o_{OUT}}}^{(\mathcal{P})})$
   - Set[b] $\mathrm{o_{UNV}} \coloneqq \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathrm{o_{UNV}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o_{UNV}}}^{(\mathcal{P})})$
   - Set[b] $\mathrm{o}_\alpha \coloneqq \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathrm{o}_\alpha}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o}_\alpha}^{(\mathcal{P})})$
   - Set[b] $\mathrm{o_s} \coloneqq \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathrm{o_s}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o_s}}^{(\mathcal{P})})$
   - Set[b] $\mathrm{o_a} \coloneqq \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathrm{o_a}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o_a}}^{(\mathcal{P})})$
   - $lin^{\mathrm{new}} \xleftarrow{\mathrm{r}} \mathbb{Z}_p$
   - Append $(lin^{\mathrm{new}}, \mathrm{sh}_{\mathcal{UH}}^{(\mathcal{P})}, \mathrm{sh}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_{OUT}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_{UNV}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o}_\alpha}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_s}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_a}}^{(\mathcal{P})})$ to $f_{\mathrm{OI}}^{(\mathcal{P})}(k)$
   - Append $(lin^{\mathrm{new}}, \mathrm{sh}_{\mathcal{UH}}^{(\mathcal{O})}, \mathrm{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o_{OUT}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o_{UNV}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o}_\alpha}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o_s}}^{(\mathcal{O})}, \mathrm{sh}_{\mathrm{o_a}}^{(\mathcal{O})})$ to $f_{\mathrm{OI}}^{(\mathcal{O})}(pid_{\mathcal{P}}, k)$
   - Extract $\mathsf{pk}_{\mathcal{U}}$ using $\mathsf{POK.ExtractWit}(td_{\mathsf{ext}}, \Pi, stmt, L_{\mathrm{B}}^{Val})$
   - Load[a] $pid_{\mathcal{U}'} \coloneqq f_{\mathrm{ID}}(\mathsf{pk}_{\mathcal{U}})$
   - Load current Subsession Identifier $ssid$ and append $ssid \mapsto lin^{\mathrm{new}}$ to f_{LN}

   ↪ Call Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ with input $(OS(k), in_{\mathcal{U}})$ in the name of $\mathcal{U}'$.
2. Upon receiving output $(\mathtt{ok})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ to $\mathcal{U}'$:
   - $(ser^{\mathrm{new}})^{(\mathcal{O})} \xleftarrow{\mathrm{r}} \mathbb{Z}_p$
   - $(\mathrm{com}_{ser^{\mathrm{new}}}^{(\mathcal{O})}, \mathrm{unv}_{ser^{\mathrm{new}}}^{(\mathcal{O})}) \leftarrow \mathsf{COM.Com}((ser^{\mathrm{new}})^{(\mathcal{O})})$
   - $\mathrm{com}_{\mathcal{UH}}^{\mathrm{new}} \coloneqq \mathrm{com}_{\mathcal{UH}}^{(\mathcal{P})} \oplus \mathrm{com}_{\mathcal{UH}}^{(\mathcal{O})}$
   - $\mathrm{com}_{ser}^{\mathrm{new}} \coloneqq \mathrm{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})} \oplus \mathrm{com}_{ser^{\mathrm{new}}}^{(\mathcal{O})}$
   - $(\mathrm{com}_{lin}^{\mathrm{new}}, \mathrm{unv}_{lin}^{\mathrm{new}}) \leftarrow \mathsf{COM.Com}(lin^{\mathrm{new}})$
   - $\sigma^{\mathrm{new}} \leftarrow \mathsf{SIG.Sgn}(\mathsf{sk}_{\mathcal{O}}, \mathrm{com}_{\mathcal{UH}}^{\mathrm{new}} \| \mathrm{com}_{ser}^{\mathrm{new}} \| \mathrm{com}_{lin}^{\mathrm{new}} \| \widetilde{\mathrm{com}_{id}})$

   ↪ Report message $((ser^{\mathrm{new}})^{(\mathcal{O})}, \mathrm{com}_{ser^{\mathrm{new}}}^{(\mathcal{O})}, \mathrm{unv}_{ser^{\mathrm{new}}}^{(\mathcal{O})}, \mathrm{com}_{lin}^{\mathrm{new}}, \mathrm{unv}_{lin}^{\mathrm{new}}, \sigma^{\mathrm{new}})$ from $\mathcal{O}$ to $\mathcal{U}$.

   ↪ Report message (lin^{new}) from $\mathcal{P}$ to $\mathcal{U}$.

   ↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to deliver outputs to $\mathcal{P}$ and $\mathcal{O}$.

$\mathcal{U}$ honest, $\mathcal{P}$ corrupted, $\mathcal{O}$ honest:

1. Upon receiving $(OS(k), \mathrm{User})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ and $(OS(k),$ $pid_{\mathcal{O}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - $(\mathrm{sh}_{\mathcal{UH}}^{(\mathcal{P})}, \mathrm{sh}_{\mathcal{UH}}^{(\mathcal{O})}) \leftarrow \pi_{\mathrm{Share}}\text{-}Share(\mathbf{0})$
   - $(\mathrm{sh}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \mathrm{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}) \leftarrow \pi_{\mathrm{Share}}\text{-}Share(\mathbf{0})$
   - $(\mathrm{sh}_{\mathrm{o_{OUT}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_{OUT}}}^{(\mathcal{O})}) \leftarrow \pi_{\mathrm{Share}}\text{-}Share(\mathbf{0})$
   - $(\mathrm{sh}_{\mathrm{o_{UNV}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_{UNV}}}^{(\mathcal{O})}) \leftarrow \pi_{\mathrm{Share}}\text{-}Share(\mathbf{0})$
   - $(\mathrm{sh}_{\mathrm{o}_\alpha}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o}_\alpha}^{(\mathcal{O})}) \leftarrow \pi_{\mathrm{Share}}\text{-}Share(\mathbf{0})$
   - $(\mathrm{sh}_{\mathrm{o_s}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_s}}^{(\mathcal{O})}) \leftarrow \pi_{\mathrm{Share}}\text{-}Share(\mathbf{0})$
   - $(\mathrm{sh}_{\mathrm{o_a}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_a}}^{(\mathcal{O})}) \leftarrow \pi_{\mathrm{Share}}\text{-}Share(\mathbf{0})$

   ↪ Report message $(\mathrm{sh}_{\mathcal{UH}}^{(\mathcal{P})}, \mathrm{sh}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_{OUT}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_{UNV}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o}_\alpha}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_s}}^{(\mathcal{P})}, \mathrm{sh}_{\mathrm{o_a}}^{(\mathcal{P})})$ from $\mathcal{U}$ to $\mathcal{P}$ and encrypted message from $\mathcal{U}$ to $\mathcal{O}$.

   ↪ Call Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ with input $(OS(k))$ in the name of $\mathcal{P}$.
2. Upon receiving output $(\mathtt{ok})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ to $\mathcal{P}$:
   - $(lin^{\mathrm{new}})^{\mathcal{O}} \xleftarrow{\mathrm{r}} \mathbb{Z}_p$
   - $(\mathrm{com}_{lin^{\mathrm{new}}}^{(\mathcal{O})}, \mathrm{unv}_{lin^{\mathrm{new}}}^{(\mathcal{O})}) \leftarrow \mathsf{COM.Com}((lin^{\mathrm{new}})^{\mathcal{O}})$
   - Parse:
     - $\left(\mathcal{UH}^{(\mathcal{O})}, \mathrm{com}_{\mathcal{UH}}^{(\mathcal{P})}, \mathrm{unv}_{\mathcal{UH}}^{(\mathcal{O})}\right) \coloneqq \mathrm{sh}_{\mathcal{UH}}^{(\mathcal{O})}$
     - $\left(in_{\mathcal{U}}^{\mathcal{O}}, \mathrm{com}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \mathrm{unv}_{in_{\mathcal{U}}}^{\mathcal{O}}\right) \coloneqq \mathrm{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}$
     - $\left(\mathrm{o_{OUT}}^{\mathcal{O}}, \mathrm{com}_{\mathrm{o_{OUT}}}^{(\mathcal{P})}, \mathrm{unv}_{\mathrm{o_{OUT}}}^{\mathcal{O}}\right) \coloneqq \mathrm{sh}_{\mathrm{o_{OUT}}}^{(\mathcal{O})}$
     - $\left(\mathrm{o_{UNV}}^{\mathcal{O}}, \mathrm{com}_{\mathrm{o_{UNV}}}^{(\mathcal{P})}, \mathrm{unv}_{\mathrm{o_{UNV}}}^{\mathcal{O}}\right) \coloneqq \mathrm{sh}_{\mathrm{o_{UNV}}}^{(\mathcal{O})}$
     - $\left(\mathrm{o}_\alpha{}^{\mathcal{O}}, \mathrm{com}_{\mathrm{o}_\alpha}^{(\mathcal{P})}, \mathrm{unv}_{\mathrm{o}_\alpha}^{\mathcal{O}}\right) \coloneqq \mathrm{sh}_{\mathrm{o}_\alpha}^{(\mathcal{O})}$
     - $\left(\mathrm{o_s}^{\mathcal{O}}, \mathrm{com}_{\mathrm{o_s}}^{(\mathcal{P})}, \mathrm{unv}_{\mathrm{o_s}}^{\mathcal{O}}\right) \coloneqq \mathrm{sh}_{\mathrm{o_s}}^{(\mathcal{O})}$
     - $\left(\mathrm{o_a}^{\mathcal{O}}, \mathrm{com}_{\mathrm{o_a}}^{(\mathcal{P})}, \mathrm{unv}_{\mathrm{o_a}}^{\mathcal{O}}\right) \coloneqq \mathrm{sh}_{\mathrm{o_a}}^{(\mathcal{O})}$

   ↪ Report message $(\mathrm{com}_{lin^{\mathrm{new}}}^{(\mathcal{O})}, \mathrm{com}_{\mathcal{UH}}^{(\mathcal{P})}, \mathrm{com}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \mathrm{com}_{\mathrm{o_{OUT}}}^{(\mathcal{P})}, \mathrm{com}_{\mathrm{o_{UNV}}}^{(\mathcal{P})}, \mathrm{com}_{\mathrm{o}_\alpha}^{(\mathcal{P})}, \mathrm{com}_{\mathrm{o_s}}^{(\mathcal{P})}, \mathrm{com}_{\mathrm{o_a}}^{(\mathcal{P})})$ from $\mathcal{O}$ to $\mathcal{P}$.
3. Upon receiving instructions from $\mathcal{Z}$ to send $((lin^{\mathrm{new}})^{(\mathcal{P})}, \mathrm{com}_{\mathcal{UH}}^{(\mathcal{O})}, \mathrm{com}_{in_{\mathcal{U}}}^{(\mathcal{O})}, \mathrm{com}_{\mathrm{o_{OUT}}}^{(\mathcal{O})}, \mathrm{com}_{\mathrm{o_{UNV}}}^{(\mathcal{O})}, \mathrm{com}_{\mathrm{o}_\alpha}^{(\mathcal{O})}, \mathrm{com}_{\mathrm{o_s}}^{(\mathcal{O})}, \mathrm{com}_{\mathrm{o_a}}^{(\mathcal{O})})$ from $\mathcal{P}$ to $\mathcal{O}$:
   - Check[a] $\mathsf{Unv}(\mathrm{com}_{\mathcal{UH}}^{(\mathcal{O})}, \mathrm{unv}_{\mathcal{UH}}^{(\mathcal{O})}, \mathcal{UH}^{(\mathcal{O})}) \stackrel{?}{=} 1$
   - Check[a] $\mathsf{Unv}(\mathrm{com}_{in_{\mathcal{U}}}^{(\mathcal{O})}, \mathrm{unv}_{in_{\mathcal{U}}}^{\mathcal{O}}, in_{\mathcal{U}}^{\mathcal{O}}) \stackrel{?}{=} 1$
   - Check[a] $\mathsf{Unv}(\mathrm{com}_{\mathrm{o_{OUT}}}^{(\mathcal{O})}, \mathrm{unv}_{\mathrm{o_{OUT}}}^{\mathcal{O}}, \mathrm{o_{OUT}}^{\mathcal{O}}) \stackrel{?}{=} 1$
   - Check[a] $\mathsf{Unv}(\mathrm{com}_{\mathrm{o_{UNV}}}^{(\mathcal{O})}, \mathrm{unv}_{\mathrm{o_{UNV}}}^{\mathcal{O}}, \mathrm{o_{UNV}}^{\mathcal{O}}) \stackrel{?}{=} 1$
   - Check[a] $\mathsf{Unv}(\mathrm{com}_{\mathrm{o}_\alpha}^{(\mathcal{O})}, \mathrm{unv}_{\mathrm{o}_\alpha}^{\mathcal{O}}, \mathrm{o}_\alpha{}^{\mathcal{O}}) \stackrel{?}{=} 1$
   - Check[a] $\mathsf{Unv}(\mathrm{com}_{\mathrm{o_s}}^{(\mathcal{O})}, \mathrm{unv}_{\mathrm{o_s}}^{\mathcal{O}}, \mathrm{o_s}^{\mathcal{O}}) \stackrel{?}{=} 1$
   - Check[a] $\mathsf{Unv}(\mathrm{com}_{\mathrm{o_a}}^{(\mathcal{O})}, \mathrm{unv}_{\mathrm{o_a}}^{\mathcal{O}}, \mathrm{o_a}^{\mathcal{O}}) \stackrel{?}{=} 1$

   ↪ Report message $((lin^{\mathrm{new}})^{\mathcal{O}}, \mathrm{unv}_{lin^{\mathrm{new}}}^{(\mathcal{O})})$ from $\mathcal{O}$ to $\mathcal{P}$.
4. Upon receiving instructions from $\mathcal{Z}$ to send (lin^{new}) from $\mathcal{P}$ to $\mathcal{U}$:
   - Check[a] $lin^{\mathrm{new}} \stackrel{?}{=} (lin^{\mathrm{new}})^{(\mathcal{O})} + (lin^{\mathrm{new}})^{(\mathcal{P})}$

– Load current Subsession Identifier *ssid* and append
ssid $\mapsto$ lin^{new} to f_{LN}
– Append $(lin^{\text{new}}, \text{sh}_{\mathcal{UH}}^{(\mathcal{O})}, \text{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}, \text{sh}_{\text{o}_{\text{OUT}}}^{(\mathcal{O})}, \text{sh}_{\text{o}_{\text{UNV}}}^{(\mathcal{O})}, \text{sh}_{\text{o}_\alpha}^{(\mathcal{O})}, \text{sh}_{\text{o}_\mathbf{s}}^{(\mathcal{O})}, \text{sh}_{\text{o}_\mathbf{a}}^{(\mathcal{O})})$ to $f_{\text{OI}}^{(\mathcal{O})}(pid_{\mathcal{P}}, k)$

↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to deliver outputs to $\mathcal{U}$ and $\mathcal{O}$.

$\mathcal{U}$ and $\mathcal{P}$ corrupted, $\mathcal{O}$ honest:

1. Upon receiving $(OS(k), pid_{\mathcal{O}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ and instructions from $\mathcal{Z}$ to send $(\text{sh}_{\mathcal{UH}}^{(\mathcal{O})}, \text{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}, \text{sh}_{\text{o}_{\text{OUT}}}^{(\mathcal{O})}, \text{sh}_{\text{o}_{\text{UNV}}}^{(\mathcal{O})}, \text{sh}_{\text{o}_\alpha}^{(\mathcal{O})}, \text{sh}_{\text{o}_\mathbf{s}}^{(\mathcal{O})}, \text{sh}_{\text{o}_\mathbf{a}}^{(\mathcal{O})}, ser, \widetilde{\text{com}_{id}}, \Pi, \text{com}_{ser^{\text{new}}}^{(\mathcal{U})})$ from $\mathcal{U}$ to $\mathcal{O}$:
   – Check[a] $ser \notin \mathsf{L_{SER}}$
   – $\mathsf{L_{SER}} \coloneqq \mathsf{L_{SER}} \cup \{ser\}$
   – Parse $\left(\mathcal{UH}^{(\mathcal{O})}, \text{com}_{\mathcal{UH}}^{(\mathcal{P})}, \text{unv}_{\mathcal{UH}}^{(\mathcal{O})}\right) \coloneqq \text{sh}_{\mathcal{UH}}^{(\mathcal{O})}$
   – Parse $\left(in_{\mathcal{U}}^{\mathcal{O}}, \text{com}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \text{unv}_{in_{\mathcal{U}}}^{\mathcal{O}}\right) \coloneqq \text{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}$
   – Parse $\left(\text{o}_{\text{OUT}}{}^{\mathcal{O}}, \text{com}_{\text{o}_{\text{OUT}}}^{(\mathcal{P})}, \text{unv}_{\text{o}_{\text{OUT}}}^{\mathcal{O}}\right) \coloneqq \text{sh}_{\text{o}_{\text{OUT}}}^{(\mathcal{O})}$
   – Parse $\left(\text{o}_{\text{UNV}}{}^{\mathcal{O}}, \text{com}_{\text{o}_{\text{UNV}}}^{(\mathcal{P})}, \text{unv}_{\text{o}_{\text{UNV}}}^{\mathcal{O}}\right) \coloneqq \text{sh}_{\text{o}_{\text{UNV}}}^{(\mathcal{O})}$
   – Parse $\left(\text{o}_\alpha{}^{\mathcal{O}}, \text{com}_{\text{o}_\alpha}^{(\mathcal{P})}, \text{unv}_{\text{o}_\alpha}^{\mathcal{O}}\right) \coloneqq \text{sh}_{\text{o}_\alpha}^{(\mathcal{O})}$
   – Parse $\left(\text{o}_\mathbf{s}{}^{\mathcal{O}}, \text{com}_{\text{o}_\mathbf{s}}^{(\mathcal{P})}, \text{unv}_{\text{o}_\mathbf{s}}^{\mathcal{O}}\right) \coloneqq \text{sh}_{\text{o}_\mathbf{s}}^{(\mathcal{O})}$
   – Parse $\left(\text{o}_\mathbf{a}{}^{\mathcal{O}}, \text{com}_{\text{o}_\mathbf{a}}^{(\mathcal{P})}, \text{unv}_{\text{o}_\mathbf{a}}^{\mathcal{O}}\right) \coloneqq \text{sh}_{\text{o}_\mathbf{a}}^{(\mathcal{O})}$
   – $(lin^{\text{new}})^{\mathcal{O}} \xleftarrow{\text{r}} \mathbb{Z}_p$
   – $(\text{com}_{lin^{\text{new}}}^{(\mathcal{O})}, \text{unv}_{lin^{\text{new}}}^{(\mathcal{O})}) \leftarrow \mathsf{COM.Com}((lin^{\text{new}})^{\mathcal{O}})$

   ↪ Report message $(\text{com}_{lin^{\text{new}}}^{(\mathcal{O})}, \text{com}_{\mathcal{UH}}^{(\mathcal{P})}, \text{com}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \text{com}_{\text{o}_{\text{OUT}}}^{(\mathcal{P})}, \text{com}_{\text{o}_{\text{UNV}}}^{(\mathcal{P})}, \text{com}_{\text{o}_\alpha}, \text{com}_{\text{o}_\mathbf{s}}, \text{com}_{\text{o}_\mathbf{a}})$ from $\mathcal{O}$ to $\mathcal{P}$
2. Upon receiving instructions from $\mathcal{Z}$ to send $((lin^{\text{new}})^{(\mathcal{P})}, \text{com}_{\mathcal{UH}}^{(\mathcal{O})}, \text{com}_{in_{\mathcal{U}}}^{(\mathcal{O})}, \text{com}_{\text{o}_{\text{OUT}}}^{(\mathcal{O})}, \text{com}_{\text{o}_{\text{UNV}}}^{(\mathcal{O})}, \text{com}_{\text{o}_\alpha}^{(\mathcal{O})}, \text{com}_{\text{o}_\mathbf{s}}^{(\mathcal{O})}, \text{com}_{\text{o}_\mathbf{a}}^{(\mathcal{O})})$ from $\mathcal{P}$ to $\mathcal{O}$:
   – $stmt \coloneqq (\text{sh}_{\mathcal{UH}}^{(\mathcal{O})}, \text{com}_{\mathcal{UH}}^{(\mathcal{O})}, ser, \widetilde{\text{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
   – Check[a] $\mathsf{POK.Vfy}(\Pi, stmt, L_{\text{O}}) \stackrel{?}{=} 1$
   – Check[a] $\mathsf{Unv}(\text{com}_{\mathcal{UH}}^{(\mathcal{O})}, \text{unv}_{\mathcal{UH}}^{(\mathcal{O})}, \mathcal{UH}^{(\mathcal{O})}) \stackrel{?}{=} 1$
   – Check[a] $\mathsf{Unv}(\text{com}_{in_{\mathcal{U}}}^{(\mathcal{O})}, \text{unv}_{in_{\mathcal{U}}}^{\mathcal{O}}, in_{\mathcal{U}}^{\mathcal{O}}) \stackrel{?}{=} 1$
   – Check[a] $\mathsf{Unv}(\text{com}_{\text{o}_{\text{OUT}}}^{(\mathcal{O})}, \text{unv}_{\text{o}_{\text{OUT}}}^{\mathcal{O}}, \text{o}_{\text{OUT}}{}^{\mathcal{O}}) \stackrel{?}{=} 1$
   – Check[a] $\mathsf{Unv}(\text{com}_{\text{o}_{\text{UNV}}}^{(\mathcal{O})}, \text{unv}_{\text{o}_{\text{UNV}}}^{\mathcal{O}}, \text{o}_{\text{UNV}}{}^{\mathcal{O}}) \stackrel{?}{=} 1$
   – Check[a] $\mathsf{Unv}(\text{com}_{\text{o}_\alpha}^{(\mathcal{O})}, \text{unv}_{\text{o}_\alpha}^{\mathcal{O}}, \text{o}_\alpha{}^{\mathcal{O}}) \stackrel{?}{=} 1$
   – Check[a] $\mathsf{Unv}(\text{com}_{\text{o}_\mathbf{s}}^{(\mathcal{O})}, \text{unv}_{\text{o}_\mathbf{s}}^{\mathcal{O}}, \text{o}_\mathbf{s}{}^{\mathcal{O}}) \stackrel{?}{=} 1$
   – Check[a] $\mathsf{Unv}(\text{com}_{\text{o}_\mathbf{a}}^{(\mathcal{O})}, \text{unv}_{\text{o}_\mathbf{a}}^{\mathcal{O}}, \text{o}_\mathbf{a}{}^{\mathcal{O}}) \stackrel{?}{=} 1$
   – Extract $\mathsf{pk}_{\mathcal{U}}$ using $\mathsf{POK.ExtractWit}(td_{\mathsf{ext}}, \Pi, stmt, L_{\text{O}})$
   – Load[a] $pid_{\mathcal{U}'} \coloneqq f_{\text{ID}}(\mathsf{pk}_{\mathcal{U}})$

   ↪ Call Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ with input $(OS(k), \bot)$ in the name of $\mathcal{U}'$
   ↪ Call Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ with input $(OS(k))$ in the name of $\mathcal{P}$
3. Upon receiving output (ok) from $\mathcal{F}_{PUBA}^{(\Delta)}$ to $\mathcal{U}'$ and output (ok) from $\mathcal{F}_{PUBA}^{(\Delta)}$ to $\mathcal{P}$:
   Compute commitments and signature:
   – Set $lin \coloneqq (lin^{\text{new}})^{\mathcal{O}} + (lin^{\text{new}})^{(\mathcal{P})}$
   – Append $(lin^{\text{new}}, \text{sh}_{\mathcal{UH}}^{(\mathcal{O})}, \text{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}, \text{sh}_{\text{o}_{\text{OUT}}}^{(\mathcal{O})}, \text{sh}_{\text{o}_{\text{UNV}}}^{(\mathcal{O})}, \text{sh}_{\text{o}_\alpha}^{(\mathcal{O})}, \text{sh}_{\text{o}_\mathbf{s}}^{(\mathcal{O})}, \text{sh}_{\text{o}_\mathbf{a}}^{(\mathcal{O})})$ to $f_{\text{OI}}^{(\mathcal{O})}(pid_{\mathcal{P}}, k)$
   – $(ser^{\text{new}})^{(\mathcal{O})} \xleftarrow{\text{r}} \mathbb{Z}_p$
   – $(\text{com}_{ser^{\text{new}}}^{(\mathcal{O})}, \text{unv}_{ser^{\text{new}}}^{(\mathcal{O})}) \leftarrow \mathsf{COM.Com}((ser^{\text{new}})^{(\mathcal{O})})$
   – $\text{com}_{\mathcal{UH}}^{\text{new}} \coloneqq \text{com}_{\mathcal{UH}}^{(\mathcal{P})} \oplus \text{com}_{\mathcal{UH}}^{(\mathcal{O})}$
   – $\text{com}_{ser}^{\text{new}} \coloneqq \text{com}_{ser^{\text{new}}}^{(\mathcal{U})} \oplus \text{com}_{ser^{\text{new}}}^{(\mathcal{O})}$
   – $(\text{com}_{lin}^{\text{new}}, \text{unv}_{lin}^{\text{new}}) \leftarrow \mathsf{COM.Com}(lin^{\text{new}})$
   – $\sigma^{\text{new}} \leftarrow \mathsf{SIG.Sgn}(\mathsf{sk}_{\mathcal{O}}, \text{com}_{\mathcal{UH}}^{\text{new}} \| \text{com}_{ser}^{\text{new}} \| \text{com}_{lin}^{\text{new}} \| \widetilde{\text{com}_{id}})$

   ↪ Report message $((lin^{\text{new}})^{(\mathcal{O})}, \text{unv}_{lin^{\text{new}}}^{(\mathcal{O})})$ from $\mathcal{O}$ to $\mathcal{P}$ and message $((ser^{\text{new}})^{(\mathcal{O})}, \text{com}_{ser^{\text{new}}}^{(\mathcal{O})}, \text{unv}_{ser^{\text{new}}}^{(\mathcal{O})}, \text{com}_{lin}^{\text{new}}, \text{unv}_{lin}^{\text{new}}, \sigma^{\text{new}})$ from $\mathcal{O}$ to $\mathcal{U}$.
   ↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to deliver outputs to $\mathcal{O}$.

$\left\{\pi\text{-}Outsourced\ Analytics^{(k)}\right\}_{k=1}^{K}$:

$\mathcal{P}$ and $\mathcal{O}$ honest:

1. Upon receiving $(OA(k), pid_{\mathcal{P}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ and $(OA(k), pid_{\mathcal{O}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   – Load[a] and remove the first Z_k entries $\{(lin, \text{sh}_{\mathcal{UH}}^{(\mathcal{P})}, \text{sh}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \text{sh}_{\text{o}_{\text{OUT}}}^{(\mathcal{P})}, \text{sh}_{\text{o}_{\text{UNV}}}^{(\mathcal{P})}, \text{sh}_{\text{o}_\alpha}^{(\mathcal{P})}, \text{sh}_{\text{o}_\mathbf{s}}^{(\mathcal{P})}, \text{sh}_{\text{o}_\mathbf{a}}^{(\mathcal{P})})_z\}_{z=1}^{Z_k}$ from $f_{\text{OI}}^{(\mathcal{P})}(k)$
   – Load[a] and remove the first Z_k entries $\{(lin, \text{sh}_{\mathcal{UH}}^{(\mathcal{O})}, \text{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}, \text{sh}_{\text{o}_{\text{OUT}}}^{(\mathcal{O})}, \text{sh}_{\text{o}_{\text{UNV}}}^{(\mathcal{O})}, \text{sh}_{\text{o}_\alpha}^{(\mathcal{O})}, \text{sh}_{\text{o}_\mathbf{s}}^{(\mathcal{O})}, \text{sh}_{\text{o}_\mathbf{a}}^{(\mathcal{O})})_z\}_{z=1}^{Z_k}$ from $f_{\text{OI}}^{(\mathcal{O})}(pid_{\mathcal{P}}, k)$
   – For every $z \in \{1, \ldots, Z_k\}$ with $lin_z \neq \bot$, combine[a]:
      – $\mathcal{UH}_z \coloneqq \pi_{\text{Share}}\text{-}Combine(\text{sh}_{\mathcal{UH}_z}^{(\mathcal{P})}, \text{sh}_{\mathcal{UH}_z}^{(\mathcal{O})})$
      – $in_{\mathcal{U}_z} \coloneqq \pi_{\text{Share}}\text{-}Combine(\text{sh}_{in_{\mathcal{U}_z}}^{(\mathcal{P})}, \text{sh}_{in_{\mathcal{U}_z}}^{(\mathcal{O})})$
      – $\text{o}_{\text{OUT}\,z} \coloneqq \pi_{\text{Share}}\text{-}Combine(\text{sh}_{\text{o}_{\text{OUT}\,z}}^{(\mathcal{P})}, \text{sh}_{\text{o}_{\text{OUT}\,z}}^{(\mathcal{O})})$
      – $\text{o}_{\text{UNV}\,z} \coloneqq \pi_{\text{Share}}\text{-}Combine(\text{sh}_{\text{o}_{\text{UNV}\,z}}^{(\mathcal{P})}, \text{sh}_{\text{o}_{\text{UNV}\,z}}^{(\mathcal{O})})$
      – $\text{o}_{\alpha\,z} \coloneqq \pi_{\text{Share}}\text{-}Combine(\text{sh}_{\text{o}_{\alpha\,z}}^{(\mathcal{P})}, \text{sh}_{\text{o}_{\alpha\,z}}^{(\mathcal{O})})$
      – $\text{o}_{\mathbf{s}\,z} \coloneqq \pi_{\text{Share}}\text{-}Combine(\text{sh}_{\text{o}_{\mathbf{s}_z}}^{(\mathcal{P})}, \text{sh}_{\text{o}_{\mathbf{s}_z}}^{(\mathcal{O})})$
      – $\text{o}_{\mathbf{a}\,z} \coloneqq \pi_{\text{Share}}\text{-}Combine(\text{sh}_{\text{o}_{\mathbf{a}_z}}^{(\mathcal{P})}, \text{sh}_{\text{o}_{\mathbf{a}_z}}^{(\mathcal{O})})$

↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to continue.

2. Upon receiving leak $(z, \alpha_z, \mathbf{s}_z, \mathbf{a}_z, out_{\mathcal{U}_z})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$, do for each z for which a leak $(z, \cdot, \cdot, \cdot, \cdot)$ exists:
   - $(\mathtt{com}_{\mathbf{a}_z}, \mathrm{unv}_{\mathbf{a}_z}) \xleftarrow{r} \mathsf{Com}(\mathbf{a}_z)$
   - $\mathtt{c}_{\alpha_z} := \alpha_z + \mathtt{o}_{\alpha z}$
   - $\mathtt{c}_{\mathbf{s}_z} := \mathbf{s}_z + \mathtt{o}_{\mathbf{s} z}$
   - $\mathtt{c}_{\mathbf{a}_z} := \mathbf{a}_z + \mathtt{o}_{\mathbf{a} z}$
   - $\mathtt{c}_{\mathrm{unv}_{\mathbf{a}_z}} := \mathrm{unv}_{\mathbf{a}_z} + \mathtt{o}_{\mathtt{UNV}}$
   - $\mathtt{c}_{out_{\mathcal{U}_z}} := out_{\mathcal{U}_z} + \mathtt{o}_{\mathtt{OUT}\, z}$
   - $f_{\mathrm{UP}}^{(\mathcal{P})}(lin_z) := (\mathtt{c}_\alpha, \mathtt{c}_{\mathbf{s}}, \mathtt{c}_{\mathbf{a}}, \mathtt{c}_{\mathrm{unv}_{\mathbf{a}}}, \mathtt{c}_{out_{\mathcal{U}}})_z$
   - $f_{\mathrm{UP}}^{(\mathcal{O})}(lin_z) := (\alpha_z, \mathbf{s}_z, \mathtt{com}_{\mathbf{a}_z}, \mathtt{c}_{out_{\mathcal{U}}})_z$

   ↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to deliver outputs to all parties.

$\mathcal{P}$ corrupted, $\mathcal{O}$ honest:

1. Upon receiving $(OA(k), pid_{\mathcal{O}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ and leak ℓ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - $(\mathtt{com}_{\mathbf{0}}, \mathrm{unv}_{\mathbf{0}}, \sigma_{fp}) \leftarrow f_{\mathrm{FP}}(OA(k), \ell)$

   ↪ Report message $(\mathtt{com}_{\mathbf{0}}, \sigma_{fp})$ from $\mathcal{O}$ to $\mathcal{P}$.
2. Upon receiving instructions from $\mathcal{Z}$ to send $(OA(k), \mathtt{com}_{fp}, \{(\mathrm{sh}_{\mathcal{UH}}^{(\mathcal{P})}, \mathrm{sh}_{in_{\mathcal{U}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\mathtt{OUT}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\mathtt{UNV}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_\alpha}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\mathbf{s}}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\mathbf{a}}}^{(\mathcal{P})})_z\}_{z=1}^{Z_k})$ from $\mathcal{P}$ to $\mathcal{F}_{\mathrm{PPA}}$:
   - Check[a] $\mathsf{COM.Unv}(\mathtt{com}_{fp}, \mathrm{unv}_{\mathbf{0}}, \mathbf{0}) = 1$.
   - Load[a] and remove the first Z_k entries $\{(lin, \mathrm{sh}_{\mathcal{UH}}^{(\mathcal{O})}, \mathrm{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathtt{o}_{\mathtt{OUT}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathtt{o}_{\mathtt{UNV}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathtt{o}_\alpha}^{(\mathcal{O})}, \mathrm{sh}_{\mathtt{o}_{\mathbf{s}}}^{(\mathcal{O})}, \mathrm{sh}_{\mathtt{o}_{\mathbf{a}}}^{(\mathcal{O})})_z\}_{z=1}^{Z_k}$ from $f_{\mathrm{OI}}^{(\mathcal{O})}(pid_{\mathcal{P}}, k)$
   - For every z from 1 to Z_k, combine[a]:
     - $\mathcal{UH}_z \leftarrow \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathcal{UH}_z}^{(\mathcal{P})}, \mathrm{sh}_{\mathcal{UH}_z}^{(\mathcal{O})})$
     - $in_{\mathcal{U}_z} \leftarrow \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{in_z}^{(\mathcal{P})}, \mathrm{sh}_{in_z}^{(\mathcal{O})})$
     - $\mathtt{o}_{\mathtt{OUT}\, z} \leftarrow \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathtt{o}_{\mathtt{OUT}\, z}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\mathtt{OUT}\, z}}^{(\mathcal{O})})$
     - $\mathtt{o}_{\mathtt{UNV}\, z} \leftarrow \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathtt{o}_{\mathtt{UNV}\, z}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\mathtt{UNV}\, z}}^{(\mathcal{O})})$
     - $\mathtt{o}_{\alpha z} \leftarrow \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathtt{o}_{\alpha z}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\alpha z}}^{(\mathcal{O})})$
     - $\mathtt{o}_{\mathbf{s} z} \leftarrow \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathtt{o}_{\mathbf{s}_z}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\mathbf{s}_z}}^{(\mathcal{O})})$
     - $\mathtt{o}_{\mathbf{a} z} \leftarrow \pi_{\mathrm{Share}}\text{-}Combine(\mathrm{sh}_{\mathtt{o}_{\mathbf{a}_z}}^{(\mathcal{P})}, \mathrm{sh}_{\mathtt{o}_{\mathbf{a}_z}}^{(\mathcal{O})})$

   ↪ Call Ideal Functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ with input $(OA(k))$ in the name of $\mathcal{P}$.
3. Upon being asked by $\mathcal{F}_{PUBA}^{(\Delta)}$ for updated inputs for a set $id_{\mathrm{corrupted}} := \{\{1, \ldots, Z_k\}, \mathcal{U}_z \text{ corrupted}\}$:
   - Send inputs $\{in_{\mathcal{U}_z} | z \in id_{\mathrm{corrupted}}\}$ to $\mathcal{F}_{PUBA}^{(\Delta)}$

   ↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to continue.
4. Upon receiving leak $\{(z, \alpha_z, \mathbf{s}_z, \mathbf{a}_z, out_{\mathcal{U}_z}) | z \in id_{\text{ corrupted}}\}$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ and output $(\mathtt{ok})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ to $\mathcal{P}$:
   - For each z in $1, \ldots, Z_k$:
     - If an entry $\{(z), \cdot, \cdot, \cdot, \cdot\}$ exists in the leaked set:
       - $\mathtt{c}_{\alpha_z} := \alpha_z + \mathtt{o}_{\alpha z}$
       - $\mathtt{c}_{\mathbf{s}_z} := \mathbf{s}_z + \mathtt{o}_{\mathbf{s} z}$
       - $\mathtt{c}_{\mathbf{a}_z} := \mathbf{a}_z + \mathtt{o}_{\mathbf{a} z}$
       - $(\mathtt{com}_{\mathbf{a}_z}, \mathrm{unv}_{\mathbf{a}_z}) \leftarrow \mathsf{Com}(\mathbf{a}_z)$
       - $\mathtt{c}_{\mathrm{unv}_{\mathbf{a}_z}} := \mathrm{unv}_{\mathbf{a}_z} + \mathtt{o}_{\mathtt{UNV}\, z}$
       - $\mathtt{c}_{out_{\mathcal{U}_z}} := out_{\mathcal{U}_z} + \mathtt{o}_{\mathtt{OUT}\, z}$
       - $f_{\mathrm{UP}}^{(\mathcal{P})}(lin_z) := (\mathtt{c}_{\alpha_z}, \mathtt{c}_{\mathbf{s}_z}, \mathtt{c}_{\mathbf{a}_z}, \mathtt{c}_{\mathrm{unv}_{\mathbf{a}_z}}, \mathtt{c}_{out_{\mathcal{U}_z}})$
       - $f_{\mathrm{UP}}^{(\mathcal{O})}(lin_z) := (\alpha, \mathbf{s}, \mathtt{com}_{\mathbf{a}_z}, \mathtt{c}_{out_{\mathcal{U}_z}})$
     - Else:
       - Draw random $\mathtt{c}_{\alpha_z}$, $\mathtt{c}_{\mathbf{s}_z}$, $\mathtt{c}_{\mathbf{a}_z}, \mathtt{c}_{out_{\mathcal{U}_z}}$ and $\mathtt{c}_{\mathrm{unv}_{\mathbf{a}_z}}$
       - $f_{\mathrm{UP}}^{(\mathcal{P})}(lin_z) := (\mathtt{c}_{\alpha_z}, \mathtt{c}_{\mathbf{s}_z}, \mathtt{c}_{\mathbf{a}_z}, \mathtt{c}_{\mathrm{unv}_{\mathbf{a}_z}}, \mathtt{c}_{out_{\mathcal{U}_z}})$

   ↪ Report message $\{(\mathtt{c}_{\alpha_z}, \mathtt{c}_{\mathbf{s}_z}, \mathtt{c}_{\mathbf{a}_z}, \mathtt{c}_{\mathrm{unv}_{\mathbf{a}_z}}, \mathtt{c}_{out_{\mathcal{U}_z}})\}_{z=1}^{Z_k}$ from $\mathcal{F}_{\mathrm{PPA}}$ to $\mathcal{P}$.

   ↪ Allow $\mathcal{F}_{PUBA}^{(\Delta)}$ to deliver outputs to $\mathcal{O}$.

π-Update:

$\mathcal{U}$, $\mathcal{P}$ and $\mathcal{O}$ honest: Report encrypted messages.

$\mathcal{U}$ corrupted, $\mathcal{P}$ and $\mathcal{O}$ honest:

1. Upon receiving $(\mathtt{Upd}, pid_{\mathcal{P}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ and instructions from $\mathcal{Z}$ to send (lin) from $\mathcal{U}$ to $\mathcal{P}$:
   - If $f_{\mathrm{UP}}^{(\mathcal{P})}(lin) \neq \perp$:
     - Load and remove $(\mathtt{c}_\alpha, \mathtt{c}_{\mathbf{s}}, \mathtt{c}_{\mathbf{a}}, \mathtt{c}_{\mathrm{unv}_{\mathbf{a}}}, \mathtt{c}_{out_{\mathcal{U}}}) := f_{\mathrm{UP}}^{(\mathcal{P})}(lin)$

     ↪ Report message $(\mathtt{c}_\alpha, \mathtt{c}_{\mathbf{s}}, \mathtt{c}_{\mathbf{a}}, \mathtt{c}_{\mathrm{unv}_{\mathbf{a}}}, \mathtt{c}_{out_{\mathcal{U}}})$ from $\mathcal{P}$ to $\mathcal{U}$.
   - otherwise continue without reporting a message
2. Upon receiving $(\mathtt{Upd}, pid_{\mathcal{O}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$:
   - Load[a] and remove $(\alpha, \mathbf{s}, \mathtt{com}_{\mathbf{a}}, \mathtt{c}_{out_{\mathcal{U}}}) := f_{\mathrm{UP}}^{(\mathcal{O})}(lin)$
3. **if** $\alpha \neq \perp \vee \mathbf{s} \neq \perp$ **then**

   Upon receiving instructions from $\mathcal{Z}$ to send $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, \mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \widetilde{\mathtt{com}_{id}}, ser, lin, \Pi, \Pi_{Tr}, \mathtt{com}_{ser^{\mathrm{new}}}^{(\mathcal{U})})$ from $\mathcal{U}$ to $\mathcal{O}$:

   Check proof:
   - $stmt := (\widetilde{\mathtt{com}_{\mathcal{UH}}}, ser, lin, \widetilde{\mathtt{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
   - $stmt^{Tr} := (\widetilde{\mathtt{com}_{\mathcal{UH}}}, \mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \alpha, \mathbf{s})$
   - Check[a] $\mathsf{POK.Vfy}(\Pi, stmt, L_{\mathrm{U}}) \stackrel{?}{=} 1$
   - Check[a] $\mathsf{POK.Vfy}(\Pi_{Tr}, stmt^{Tr}, L_{\mathrm{B}}^{Tr}) \stackrel{?}{=} 1$

– $\mathtt{com}^{\text{new}}_{\mathcal{UH}} \coloneqq \mathtt{com}''_{\mathcal{UH}} \oplus \mathtt{com}_{\mathbf{a}}$

4. **else**
   Upon receiving instructions from $\mathcal{Z}$ to send $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, \widetilde{\mathtt{com}_{id}}, ser, lin, \Pi, \mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}})$ from $\mathcal{U}$ to $\mathcal{O}$:
   Check proof:
   – $stmt \coloneqq (\widetilde{\mathtt{com}_{\mathcal{UH}}}, ser, lin, \widetilde{\mathtt{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
   – Check$^{\mathrm{a}}$ $\mathsf{POK.Vfy}(\Pi, stmt, L_{\mathrm{U}}) \stackrel{?}{=} 1$
   – $\mathtt{com}^{\text{new}}_{\mathcal{UH}} \coloneqq \widetilde{\mathtt{com}_{\mathcal{UH}}} \oplus \mathtt{com}_{\mathbf{a}}$
5. **fi**
   Check serial number:
   – Check$^{\mathrm{a}}$ $ser \notin \mathsf{L_{SER}}$
   – $\mathsf{L_{SER}} \coloneqq \mathsf{L_{SER}} \cup \{ser\}$
   – Extract $\mathsf{pk}_{\mathcal{U}}$ using $\mathsf{POK.ExtractWit}(td_{\mathsf{ext}}, \Pi, stmt, L_{\mathrm{U}})$
   – Load$^{\mathrm{a}}$ $pid_{\mathcal{U}'} \coloneqq f_{\mathrm{ID}}(\mathsf{pk}_{\mathcal{U}})$
   ↪ Call Ideal Functionality $\mathcal{F}^{(\Delta)}_{PUBA}$ with input (Upd) in the name of $\mathcal{U}'$.
6. Upon receiving output $(\alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}})$ from $\mathcal{F}^{(\Delta)}_{PUBA}$ to $\mathcal{U}'$:
   Draw share of new serial number:
   – $(ser^{\text{new}})^{(\mathcal{O})} \xleftarrow{\mathrm{r}} \mathbb{Z}_p$
   – $(\mathtt{com}^{(\mathcal{O})}_{ser^{\text{new}}}, \mathtt{unv}^{(\mathcal{O})}_{ser^{\text{new}}}) \leftarrow \mathsf{COM.Com}((ser^{\text{new}})^{(\mathcal{O})})$
   Compute commitments and signature for updated user history:
   – $\mathtt{com}^{\text{new}}_{ser} \coloneqq \mathtt{com}^{(\mathcal{O})}_{ser^{\text{new}}} \oplus \mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}}$
   – $(\mathtt{com}^{\text{new}}_{lin}, \mathtt{unv}^{\text{new}}_{lin}) \leftarrow \mathsf{COM.Com}(0)$
   – $\sigma^{\text{new}} \leftarrow \mathsf{SIG.Sgn}(\mathsf{sk}_{\mathcal{O}}, \mathtt{com}^{\text{new}}_{\mathcal{UH}} \| \mathtt{com}^{\text{new}}_{ser} \| \mathtt{com}^{\text{new}}_{lin} \| \widetilde{\mathtt{com}_{id}})$
   ↪ Report message $(\mathtt{com}_{\mathbf{a}}, \mathtt{com}^{\text{new}}_{\mathcal{UH}}, (ser^{\text{new}})^{(\mathcal{O})}, \mathtt{com}^{(\mathcal{O})}_{ser^{\text{new}}}, \mathtt{unv}^{(\mathcal{O})}_{ser^{\text{new}}}, \mathtt{com}^{\text{new}}_{lin}, \mathtt{unv}^{\text{new}}_{lin}, \sigma^{\text{new}}, \mathtt{c}_{out_{\mathcal{U}}})$ from $\mathcal{O}$ to $\mathcal{U}$.
   ↪ Allow $\mathcal{F}^{(\Delta)}_{PUBA}$ to deliver outputs.

.....

$\mathcal{U}$ honest, $\mathcal{P}$ corrupted, $\mathcal{O}$ honest:

1. Upon receiving $(\mathtt{Upd}, pid_{\mathcal{O}})$ and $(\mathtt{Upd}, \mathrm{User})$ from $\mathcal{F}^{(\Delta)}_{PUBA}$:
   ↪ Call Ideal Functionality $\mathcal{F}^{(\Delta)}_{PUBA}$ with input (Upd) in the name of $\mathcal{P}$.
2. Upon receiving leak $ssid$ from $\mathcal{F}^{(\Delta)}_{PUBA}$:
   – Load$^{\mathrm{a}}$ $lin \coloneqq f_{\mathrm{LN}}(ssid)$
   ↪ Report message (lin) from $\mathcal{U}$ to $\mathcal{P}$.
3. Upon receiving instructions from $\mathcal{Z}$ to send $(\mathtt{c}_{\alpha}, \mathtt{c}_{\mathbf{s}}, \mathtt{c}_{\mathbf{a}}, \mathtt{c}_{\mathtt{unv}_{\mathbf{a}}}, \mathtt{c}_{out_{\mathcal{U}}})$ from $\mathcal{P}$ to $\mathcal{U}$:
   – Check$^{\mathrm{a}}$ $(\mathtt{c}_{\alpha}, \mathtt{c}_{\mathbf{s}}, \mathtt{c}_{\mathbf{a}}, \mathtt{c}_{\mathtt{unv}_{\mathbf{a}}}, \mathtt{c}_{out_{\mathcal{U}}}) \stackrel{?}{=} f^{(\mathcal{P})}_{\mathrm{UP}}(lin)$ and remove the entry.
   ↪ Allow $\mathcal{F}^{(\Delta)}_{PUBA}$ to continue.

.....

$\mathcal{U}$ and $\mathcal{P}$ corrupted, $\mathcal{O}$ honest:

1. Upon receiving $(\mathtt{Upd}, pid_{\mathcal{O}})$ from $\mathcal{F}^{(\Delta)}_{PUBA}$:
   – Load$^{\mathrm{a}}$ and remove $(\alpha, \mathbf{s}, \mathtt{com}_{\mathbf{a}}, \mathtt{c}_{out_{\mathcal{U}}}) \coloneqq f^{(\mathcal{O})}_{\mathrm{UP}}(lin)$.
2. **if** $\alpha \neq \bot \vee \mathbf{s} \neq \bot$ **then**
   Upon receiving instructions from $\mathcal{Z}$ to send $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, \mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \widetilde{\mathtt{com}_{id}}, ser, lin, \Pi, \Pi_{Tr}, \mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}})$ from $\mathcal{U}$ to $\mathcal{O}$:
   Check proof:
   – $stmt \coloneqq (\widetilde{\mathtt{com}_{\mathcal{UH}}}, ser, lin, \widetilde{\mathtt{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
   – $stmt^{Tr} \coloneqq (\widetilde{\mathtt{com}_{\mathcal{UH}}}, \mathtt{com}'_{\mathcal{UH}}, \mathtt{com}''_{\mathcal{UH}}, \alpha, \mathbf{s})$
   – Check$^{\mathrm{a}}$ $\mathsf{POK.Vfy}(\Pi, stmt, L_{\mathrm{U}}) \stackrel{?}{=} 1$
   – Check$^{\mathrm{a}}$ $\mathsf{POK.Vfy}(\Pi_{Tr}, stmt^{Tr}, L^{Tr}_{\mathrm{B}}) \stackrel{?}{=} 1$
   – $\mathtt{com}^{\text{new}}_{\mathcal{UH}} \coloneqq \mathtt{com}''_{\mathcal{UH}} \oplus \mathtt{com}_{\mathbf{a}}$
3. **else**
   Upon receiving instructions from $\mathcal{Z}$ to send $(\widetilde{\mathtt{com}_{\mathcal{UH}}}, \widetilde{\mathtt{com}_{id}}, ser, lin, \Pi, \mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}})$ from $\mathcal{U}$ to $\mathcal{O}$:
   Check proof:
   – $stmt \coloneqq (\widetilde{\mathtt{com}_{\mathcal{UH}}}, ser, lin, \widetilde{\mathtt{com}_{id}}, \mathsf{vk}_{\mathcal{O}})$
   – Check$^{\mathrm{a}}$ $\mathsf{POK.Vfy}(\Pi, stmt, L_{\mathrm{U}}) \stackrel{?}{=} 1$
   – $\mathtt{com}^{\text{new}}_{\mathcal{UH}} \coloneqq \widetilde{\mathtt{com}_{\mathcal{UH}}} \oplus \mathtt{com}_{\mathbf{a}}$
4. **fi**
   Check serial number:
   – Check$^{\mathrm{a}}$ $ser \notin \mathsf{L_{SER}}$
   – $\mathsf{L_{SER}} \coloneqq \mathsf{L_{SER}} \cup \{ser\}$
   – Extract $\mathsf{pk}_{\mathcal{U}}$ using $\mathsf{POK.ExtractWit}(td_{\mathsf{ext}}, \Pi, stmt, L_{\mathrm{U}})$
   – Load$^{\mathrm{a}}$ $pid_{\mathcal{U}'} \coloneqq f_{\mathrm{ID}}(\mathsf{pk}_{\mathcal{U}})$
   ↪ Call Ideal Functionality $\mathcal{F}^{(\Delta)}_{PUBA}$ with input (Upd) in the name of $\mathcal{U}'$.
   ↪ Call Ideal Functionality $\mathcal{F}^{(\Delta)}_{PUBA}$ with input (Upd) in the name of $\mathcal{P}$.
5. Upon receiving output (ok) from $\mathcal{F}^{(\Delta)}_{PUBA}$ to $\mathcal{P}$ and output $(\alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}})$ from $\mathcal{F}^{(\Delta)}_{PUBA}$ to $\mathcal{U}'$:
   – $(ser^{\text{new}})^{(\mathcal{O})} \xleftarrow{\mathrm{r}} \mathbb{Z}_p$
   – $(\mathtt{com}^{(\mathcal{O})}_{ser^{\text{new}}}, \mathtt{unv}^{(\mathcal{O})}_{ser^{\text{new}}}) \leftarrow \mathsf{COM.Com}((ser^{\text{new}})^{(\mathcal{O})})$
   – $\mathtt{com}^{\text{new}}_{ser} \coloneqq \mathtt{com}^{(\mathcal{O})}_{ser^{\text{new}}} \oplus \mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}}$
   – $(\mathtt{com}^{\text{new}}_{lin}, \mathtt{unv}^{\text{new}}_{lin}) \leftarrow \mathsf{COM.Com}(0)$
   – $\sigma^{\text{new}} \leftarrow \mathsf{SIG.Sgn}(\mathsf{sk}_{\mathcal{O}}, \mathtt{com}^{\text{new}}_{\mathcal{UH}} \| \mathtt{com}^{\text{new}}_{ser} \| \mathtt{com}^{\text{new}}_{lin} \| \widetilde{\mathtt{com}_{id}})$

↪ Report message $(\mathtt{com}_{\mathbf{a}}, \mathtt{com}^{\text{new}}_{\mathcal{UH}}, (ser^{\text{new}})^{(\mathcal{O})}, \mathtt{com}^{(\mathcal{O})}_{ser^{\text{new}}}, \mathrm{unv}^{(\mathcal{O})}_{ser^{\text{new}}}, \mathtt{com}^{\text{new}}_{lin}, \mathrm{unv}^{\text{new}}_{lin}, \sigma^{\text{new}}, \mathtt{c}_{out_{\mathcal{U}}})$ from $\mathcal{O}$ to $\mathcal{U}$.

↪ Allow $\mathcal{F}^{(\Delta)}_{PUBA}$ to deliver output to $\mathcal{O}$.

a If this fails, output $\perp$ and abort.

b If this fails, use a default value $\mathbf{0}$.

For our proof, we consider the following hybrid games H_i:

H_1 The hybrid H_1 is equivalent to the real experiment. That is,

$$\mathsf{H}_1 \coloneqq \mathrm{EXEC}_{\pi_{PUBA},\mathcal{F}_{\mathrm{BB}},\mathcal{F}_{\mathrm{PPA}},\mathcal{S}_1,\mathcal{Z}}(1^\kappa)$$

This means that all parties execute the real protocol.

H_2 All calls to hybrid functionalities, namely to $\mathcal{F}_{\mathrm{PPA}}$, $\mathcal{F}_{\mathrm{KE}}$ and $\mathcal{F}_{\mathrm{BB}}$, are replaced by calls to $\mathcal{S}_2$, who simulates their behavior using the respective code of the simulators $\mathcal{S}_{\mathcal{F}_{\mathrm{PPA}}}$ and $\mathcal{S}_{\mathcal{F}_{\mathrm{BB}}}$.

H_3 The simulator now maintains the list $\mathsf{L}_{\mathsf{SER}}$, in which he stores the serials that were opened by a user. That is, whenever a user proves that the used logbook λ is "fresh" and hasn't been used before by sending a serial number *ser* together with a proof Π during any of the tasks for Bookkeeping, Outsourced Analytics or Update, $\mathcal{S}_3$ verifies the proof and aborts, if either the proof fails to verify or if serial is already contained in $\mathsf{L}_{\mathsf{SER}}$. If no abort happened, $\mathcal{S}_3$ adds *ser* to the list $\mathsf{L}_{\mathsf{SER}}$. The code of the operator $\mathcal{O}$ is changed in such a way, that the checks that the simulator does now are not performed again by the operator.

H_4 Introduces a map $f^{(\mathcal{O})}_{\mathrm{UP}}$ and for each proxy a map $f^{(\mathcal{P})}_{\mathrm{UP}}$ to be used by the simulator $\mathcal{S}_4$: $f^{(\mathcal{O})}_{\mathrm{UP}}$ is similar to what an honest operator would store for the updates after the Outsourced Analytics-task, $f^{(\mathcal{P})}_{\mathrm{UP}}$ is the equivalent for the respective proxy. During simulation of $\mathcal{F}_{\mathrm{PPA}}$ for Outsourced Analytics-tasks, i.e. after $\mathcal{O}$ sent a message $(OA(k), fp, \mathrm{unv}_{fp}, \{(\mathrm{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\alpha}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathbf{s}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathbf{a}}})_z\}^{Z_k}_{z=1}, in_{\mathcal{O}})$ and $\mathcal{P}$ input $(OA(k), \mathtt{com}_{fp}, \{(\mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathbf{s}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathbf{a}}})_z\}^{Z_k}_{z=1})$ to $\mathcal{F}_{\mathrm{PPA}}$, Simulator $\mathcal{S}_4$ computes Δ honestly (with fresh coins, if necessary), based on the two inputs. $\mathcal{S}_4$ uses the protocol π_{Share}-*Combine* on both received shares to reconstruct $\mathcal{UH}_z$, $\mathtt{o}_{\alpha z}$, $\mathtt{o}_{\mathbf{s}z}$, $\mathtt{o}_{\mathbf{a}z}$, $\mathtt{o}_{\mathtt{OUT}z}$ and $\mathtt{o}_{\mathtt{UNV}z}$ for each user $z \in [Z]$. If reconstruction on any of the shares fails, $\mathcal{S}_4$ aborts. With this, $\mathcal{S}_4$ computes $\mathtt{c}_{\alpha_z} \coloneqq \alpha_z + \mathtt{o}_{\alpha z}$, $\mathtt{c}_{\mathbf{s}_z} \coloneqq \mathbf{s}_z + \mathtt{o}_{\mathbf{s}z}$, $\mathtt{c}_{\mathbf{a}_z} \coloneqq \mathbf{a}_z + \mathtt{o}_{\mathbf{a}z}$, $(\mathtt{com}_{\mathbf{a}_z}, \mathrm{unv}_{\mathbf{a}_z}) \leftarrow \mathsf{COM.Com}(\mathbf{a}_z)$, $\mathtt{c}_{\mathrm{unv}_{\mathbf{a}_z}} \coloneqq \mathrm{unv}_{\mathbf{a}_z} + \mathtt{o}_{\mathtt{UNV}}$ and $\mathtt{c}_{out_z} \coloneqq out_z + \mathtt{o}_{\mathtt{OUT}z}$. $\mathcal{S}_4$ then adds a new entry $(lin_z \mapsto (\mathtt{c}_\alpha, \mathtt{c}_{\mathbf{s}}, \mathtt{c}_{\mathbf{a}}, \mathtt{c}_{\mathrm{unv}_{\mathbf{a}}, \mathtt{c}_{out_{\mathcal{U}}}}))$ to $f^{(\mathcal{P})}_{\mathrm{UP}}$ and a new entry $(lin_z \mapsto (\alpha, \mathbf{s}, \mathtt{com}_{\mathbf{a}}, \mathtt{c}_{out_{\mathcal{U}}}))$ to $f^{(\mathcal{O})}_{\mathrm{UP}}$.
$\mathcal{S}_4$ also uses $f^{(\mathcal{P})}_{\mathrm{UP}}$ to verify that a corrupted $\mathcal{P}$ sent the correct values during the Update-task to an honest $\mathcal{U}$, which replaces $\mathcal{U}$s check with the One-Time Pads.

H_5 During setup, the reference string *crs* is created by $(crs_{\mathsf{pok}}, td_{\mathsf{ext}}) \leftarrow \mathsf{SetupEPoK}$. Also, the operator now leaks the signature key pair $(\mathsf{vk}_{\mathcal{O}}, \mathsf{sk}_{\mathcal{O}})$ to $\mathcal{S}_5$ during the Init-task;; the simulator then stores $(\mathsf{vk}_{\mathcal{O}}, \mathsf{sk}_{\mathcal{O}})$. The simulator stores td_{ext}, the remainder stays as it is.

H_6 Introduces a new map f_{ID} for the simulator that uniquely maps user's public keys $\mathsf{pk}_{\mathcal{U}}$ to the pid $pid_{\mathcal{U}}$ the respective user had during user registration.
During simulation of $\mathcal{F}_{\mathrm{BB}}$, after $\mathcal{Z}$ gave instructions to send a message $(\mathtt{Register}, pid_{\mathcal{U}}, \mathsf{pk}_{\mathcal{U}})$ from a corrupted user $\mathcal{U}$ to $\mathcal{F}_{\mathrm{BB}}$ and simulation succeeded (i.e. did not abort), $\mathcal{S}_6$ adds a new entry $(\mathsf{pk}_{\mathcal{U}} \mapsto pid_{\mathcal{U}})$ to f_{ID}.
During simulation of the task for User Registration, after $\mathcal{Z}$ gave instructions to send a message $(\Pi, \mathtt{com}_{id}, \mathtt{com}^{(\mathcal{U})}_{ser^{\text{new}}})$ from a corrupted $\mathcal{U}$ to $\mathcal{O}$, $\mathcal{S}_6$ takes $\mathsf{pk}'_{\mathcal{U}}$ from Π (since it is contained in the statement of L_{R}) and aborts if either $f_{\mathrm{ID}}(\mathsf{pk}'_{\mathcal{U}}) \neq pid_{\mathcal{U}}$ or if a user with public key $\mathsf{pk}'_{\mathcal{U}}$ is already registered.

H_7 Whenever $\mathcal{Z}$ instructs $\mathcal{S}_7$ to send a message containing a zero-knowledge proof Π in the name of a corrupted user $\mathcal{U}$ during the tasks for Bookkeeping, Outsourced Analytics and Update, $\mathcal{S}_7$ uses the trapdoor td_{ext} to extract the complete witness *wit* from Π. It then uses the extracted public key $\mathsf{pk}_{\mathcal{U}}$ to get the pid $pid_{\mathcal{U}} \coloneqq f_{\mathrm{ID}}(\mathsf{pk}_{\mathcal{U}})$ of the user whom the User History belongs to and aborts if $pid_{\mathcal{U}} = \perp$.

H_8 Introduces an incorruptible entity $\mathcal{F}^{(\Delta)}_{PUBA}$ that follows the specification from Fig. 6 into the experiment, which is only accessible by honest parties and the simulator through subroutine input/output tapes.

H_9 Replaces the Trusted Signing Authority $\mathcal{T}$ with a dummy party that immediately forwards its input to the ideal functionality $\mathcal{F}^{(\Delta)}_{PUBA}$. All interactions of $\mathcal{T}$ are simulated by $\mathcal{S}_9$ by following the original protocol.

H_{10} Introduces a map f_{FP} that maps a given task $task \in \{\mathtt{UReg} \cup BK(K) \cup OA(K)\}$ and a given identifier $\ell \in [L]$ to a tuple $(fp, \mathtt{com}_{fp}, \mathrm{unv}_{fp}, \sigma_{fp})$.

Replaces the operator $\mathcal{O}$ with a new operator $\mathcal{O}'$ that acts like the original one, but has a few minor changes. After having confirmation that *fp* can be used for a task *task* during Sign Function Parameter, $\mathcal{O}'$ sends $(fp, task)$ to $\mathcal{S}_{10}$, who computes $\mathtt{com}_{fp}$ as commitment on the actual Function Parameters and σ_{fp} as corresponding signatures honestly and stores it in $f_{\mathrm{FP}}(fp, \mathtt{com}_{fp}, \sigma_{fp})$. Furthermore, the new operator sends the *fp* to $\mathcal{S}_{10}$ during UReg, BK and OA tasks and uses the $(\mathtt{com}_{fp}, \sigma_{fp})$ obtained from $\mathcal{S}_{10}$, which the simulator obtains by looking if an entry $(fp, \cdot, \cdot)$ exists within f_{FP}.

Also, during simulation of $\mathcal{F}_{\mathrm{PPA}}$, $\mathcal{S}_{10}$ uses the decommitment information stored in $f_{\mathrm{FP}}(task, \ell)$ to verify the commitment $\mathtt{com}_{fp}$ the user or proxy input into $\mathcal{F}_{\mathrm{PPA}}$.

H_{11} Replaces the operator $\mathcal{O}$ by a dummy party, which, when receiving input by $\mathcal{Z}$, forwards the input immediately to $\mathcal{F}_{PUBA}^{(\Delta)}$. During the tasks for User Registration, Bookkeeping and Outsourced Analytics, the new operator $\mathcal{O}$ also leaks the input $in_{\mathcal{O}}$ and *fp* to $\mathcal{S}_{11}$. All messages that were sent by $\mathcal{O}$ in H_{10} are now created by $\mathcal{S}_{11}$ in H_{11} and send in the name of $\mathcal{O}$.

H_{12} Replaces the map f_{FP} introduced in H_{10} with one that has the same input space $(task, \ell)$, but whose output space only contains $(\mathtt{com}_{fp}, \mathtt{unv}_{fp}, \sigma_{fp})$, and not the actual Function Parameters.

Instead of relying on the leaked *fp* for a mapping, $\mathcal{S}_{12}$ uses the leak ℓ obtained from $\mathcal{F}_{PUBA}^{(\Delta)}$ to obtain a consistent $(\mathtt{com}_{fp}, \sigma_{fp})$ tuple. Furthermore, instead of creating these tuples honestly, $\mathcal{S}_{12}$ uses the leaked $(task, \ell)$ obtained during the Sign Function Parameter task to sample a new $(\mathtt{com}_{\mathbf{0}}, \mathtt{unv}_{\mathbf{0}}) \leftarrow \mathsf{COM.Com}(\mathbf{0})$ on the all-zero vector $\mathbf{0}$ instead of $\mathtt{com}_{fp}$ on *fp*, and uses the signing key $\mathsf{sk}_{\mathcal{T}}$ to sign $\mathtt{com}_{\mathbf{0}}$ and the given task *task*. The resulting tuple $(\mathtt{com}_{\mathbf{0}}, \sigma_{fp})$ is then stored in $f_{\mathrm{FP}}(task, \ell)$ and used whenever $\mathcal{F}_{PUBA}^{(\Delta)}$ leaks ℓ during UReg, BK or OA.

H_{13} Replaces all honest users $\mathcal{U}$ by dummy parties, which, when receiving input by $\mathcal{Z}$, forward the input immediately to $\mathcal{F}_{PUBA}^{(\Delta)}$. During the tasks for User Registration, Bookkeeping and Outsourced Analytics, the new user machines $\mathcal{U}$ also leak the input $in_{\mathcal{U}}$ to $\mathcal{S}_{13}$. All messages that were sent by honest user $\mathcal{U}$ in H_{12} are now created by $\mathcal{S}_{13}$ in H_{13} and send in the name of $\mathcal{U}$.

Note that all leaks by $\mathcal{F}_{PUBA}^{(\Delta)}$ are ignored by $\mathcal{S}_{13}$.

H_{14} Replaces all proxies $\mathcal{P}$ with dummy parties, which, when receiving input by $\mathcal{Z}$, forward the input directly to $\mathcal{F}_{PUBA}^{(\Delta)}$. All messages that were sent by honest proxies $\mathcal{P}$ in H_{13} are now created by $\mathcal{S}_{14}$ in H_{14} and send in the name of $\mathcal{P}$.

Note that all leaks by $\mathcal{F}_{PUBA}^{(\Delta)}$ are ignored by $\mathcal{S}_{14}$.

H_{15} $\mathcal{S}_{15}$ now calls $\mathcal{F}_{PUBA}^{(\Delta)}$ in the name of the corrupted parties with the correct input. This causes $\mathcal{F}_{PUBA}^{(\Delta)}$ to fully perform as defined by its specification, as all inputs are provided. Hence, instead of computing the function Δ on the inputs in order to simulate $\mathcal{F}_{\mathrm{PPA}}$, $\mathcal{S}_{15}$ now relies on the leaks provided by $\mathcal{F}_{PUBA}^{(\Delta)}$.

This game also introduces a map f_{LN}, that maps leaked *ssid* values to linking numbers *lin*.

$\mathcal{S}_{15}$ obtains the input of the corrupted parties as follows:

User Registration, $\mathcal{U}$ corrupted, $\mathcal{O}$ honest.
After simulating $\mathcal{F}_{\mathrm{PPA}}$, $\mathcal{S}_{15}$ has obtained $in_{\mathcal{U}}$ from the corrupted $\mathcal{U}$. Hence, $\mathcal{S}_{15}$ calls $\mathcal{F}_{PUBA}^{(\Delta)}$ in the name of the $\mathcal{U}$ belonging to $pid_{\mathcal{U}}$ with input $(\mathtt{UReg}, in_{\mathcal{U}})$. Since $\mathcal{F}_{PUBA}^{(\Delta)}$ now has full input, $\mathcal{S}_{15}$ uses the output $(\alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ to $\mathcal{U}$ in order to simulate $\mathcal{F}_{\mathrm{PPA}}$.

Bookkeeping, $\mathcal{U}$ corrupted, $\mathcal{O}$ honest. After $\mathcal{Z}$ gave instructions to send $(BK(k), \mathcal{UH}, \widetilde{\mathtt{unv}_{\mathcal{UH}}}, \mathtt{com}_{fp}, in_{\mathcal{U}})$ to $\mathcal{F}_{\mathrm{PPA}}$ in the name of $\mathcal{U}$, $\mathcal{S}_{15}$ uses the extracted $pid_{\mathcal{U}}$ (see H_7) to obtain the user $\mathcal{U}_{pid_{\mathcal{U}}}$ who registered for the used public key. After verifying that the commitments are valid, that is, $\mathsf{COM.Unv}(\widetilde{\mathtt{com}_{\mathcal{UH}}}, \widetilde{\mathtt{unv}_{\mathcal{UH}}}, \mathcal{UH}) = 1$ and $\mathsf{COM.Unv}(\mathtt{com}_{fp}, \mathtt{unv}_{\mathbf{0}}, \mathbf{0}) = 1$ (see hybrid H_{12}), $\mathcal{S}_{15}$ calls $\mathcal{F}_{PUBA}^{(\Delta)}$ in the name of $\mathcal{U}_{pid_{\mathcal{U}}}$ with input $(BK(k), in_{\mathcal{U}})$ where $in_{\mathcal{U}}$ has been learned from simulation of $\mathcal{F}_{\mathrm{PPA}}$.

Also, since $\mathcal{F}_{PUBA}^{(\Delta)}$ now has complete input, $\mathcal{S}_{15}$ obtains leaks. Hence, instead of computing Δ in his head, $\mathcal{S}_{15}$ uses the output $(\alpha, \mathbf{s}, \mathbf{a}, out_{\mathcal{U}})$ from $\mathcal{F}_{PUBA}^{(\Delta)}$ to $\mathcal{U}$ in order generate the output of $\mathcal{F}_{\mathrm{PPA}}$ to $\mathcal{U}$ by first computing valid commitment information $(\mathtt{com}_{\mathbf{a}}, \mathtt{unv}_{\mathbf{a}}) \leftarrow \mathsf{COM.Com}(\mathbf{a})$ and then reporting message $(\alpha, \mathbf{s}, \mathbf{a}, \mathtt{com}_{\mathbf{a}}, \mathtt{unv}_{\mathbf{a}}, out_{\mathcal{U}})$.

Outsource, $\mathcal{U}$ corrupted, $\mathcal{O}$ and $\mathcal{P}$ honest.
$\mathcal{S}_{15}$ obtains the input of the user by using π_{Share}-*Combine* on the shares $\mathcal{U}$ sent to both $\mathcal{O}$ and $\mathcal{P}$: after $\mathcal{Z}$ sent instructions to send $(\mathtt{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathtt{sh}^{(\mathcal{O})}_{in_{\mathcal{U}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o_{OUT}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o_{UNV}}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o}_\alpha}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o_s}}, \mathtt{sh}^{(\mathcal{O})}_{\mathtt{o_a}}, ser, \widetilde{\mathtt{com}_{id}}, \Pi, \mathtt{com}^{(\mathcal{U})}_{ser^{\mathrm{new}}})$ from $\mathcal{U}$ to $\mathcal{O}$ and to send $(\mathtt{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathtt{sh}^{(\mathcal{P})}_{in_{\mathcal{U}}}, \mathtt{sh}^{(\mathcal{P})}_{\mathtt{o_{OUT}}}, \mathtt{sh}^{(\mathcal{P})}_{\mathtt{o_{UNV}}}, \mathtt{sh}^{(\mathcal{P})}_{\mathtt{o}_\alpha}, \mathtt{sh}^{(\mathcal{P})}_{\mathtt{o_s}}, \mathtt{sh}^{(\mathcal{P})}_{\mathtt{o_a}})$ from $\mathcal{U}$ to $\mathcal{P}$, $\mathcal{S}_{15}$ uses the shares from both $\mathcal{O}$ and $\mathcal{P}$ to reconstruct $in_{\mathcal{U}} \coloneqq \pi_{\mathrm{Share}}$-*Combine*(

$\mathrm{sh}^{(\mathcal{P})}_{in_\mathcal{U}}, \mathrm{sh}^{(\mathcal{O})}_{in_\mathcal{U}}$); if this fails, $\mathcal{S}_{15}$ sets $in_\mathcal{U} \coloneqq \bot$. $\mathcal{S}_{15}$ calls $\mathcal{F}^{(\Delta)}_{PUBA}$ in the name of $\mathcal{U}_{pid_\mathcal{U}}$, where $pid_\mathcal{U}$ corresponds to the user whose identity was extracted from Π, using input $(OS(k), in_\mathcal{U})$.

Outsource, $\mathcal{U}$ honest, $\mathcal{O}$ honest, $\mathcal{P}$ corrupted. $\mathcal{P}$ is designed to neither learn secrets, nor to have secrets itself. Simulator $\mathcal{S}_{15}$ calls $\mathcal{F}^{(\Delta)}_{PUBA}$ in the name of $\mathcal{P}$ with input $(OS(k))$. Also, $\mathcal{S}_{15}$ sets $f_{\mathrm{ID}}(ssid) \coloneqq lin^{\mathrm{new}}$ after negotiating the linking number.

Outsource, $\mathcal{U}$ and $\mathcal{P}$ corrupted, $\mathcal{O}$ honest. $\mathcal{S}_{15}$ calls $\mathcal{F}^{(\Delta)}_{PUBA}$ for both the proxy $\mathcal{P}$ (who has no secret input whatsoever) and the user $\mathcal{U}_{pid_\mathcal{U}}$ (who was identified using the $\mathsf{pk}_\mathcal{U}$ in Π), but using $in_\mathcal{U} \coloneqq \bot$.

Outsourced Analytics, $\mathcal{P}$ corrupted, $\mathcal{O}$ honest. Since H_{11}, $\mathcal{S}_{15}$ maintains the list $f^{(\mathcal{O})}_{\mathrm{UP}}$ in the same way an honest operator would. As $\mathcal{S}_{15}$ follows the operators code (due to H_{11}) it stores all the tuples $(lin^{\mathrm{new}}, \mathrm{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{O})}_{in_\mathcal{U}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\alpha}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\mathbf{a}})$ during the Outsource task; given the additional inputs from simulation of $\mathcal{F}_{\mathrm{PPA}}$ $\mathcal{S}_{15}$ now has a complete view of the used shares.

After receiving instructions from $\mathcal{Z}$ to send a message $(OA(k), \mathtt{com}_{fp}, \{\mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_\mathcal{U}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\mathbf{a}}\}^{Z}_{z=1})$ from $\mathcal{P}$ to $\mathcal{F}_{\mathrm{PPA}}$, $\mathcal{S}_{15}$ restores each input using the information stored in $f^{(\mathcal{O})}_{\mathrm{UP}}$ and aborts if the reconstruction via π_{Share}-*Combine* fails. When asked by $\mathcal{F}^{(\Delta)}_{PUBA}$ for inputs for an index set $I \subset [Z_k]$ of corrupted users $\{\mathcal{U}_z\}_{z\in I}$, $\mathcal{S}_{15}$ enters the inputs $in_\mathcal{U}$ from the respective indices.

Reconstruction of the output now only happens for parties $z \in I$ in the corrupted party set; information related to honest parties $(\mathtt{c}_{\alpha_z}, \mathtt{c}_{\mathbf{s}_z}, \mathtt{c}_{\mathbf{a}_z}, \mathtt{c}_{\mathtt{unv}_{\mathbf{a}_z}}, \mathtt{c}_{out_{\mathcal{U}_z}})$ is drawn at random and put to $f^{(\mathcal{P})}_{\mathrm{UP}}$.

Update, any corrupted party. The inputs to the Update-task contain no secrets; so $\mathcal{S}_{15}$ can call $\mathcal{F}^{(\Delta)}_{PUBA}$ in the name of any corrupted party with input (Upd). For corrupted users, $\mathcal{S}_{15}$ awaits the proof Π to extract the correct user. For corrupted Proxies, $\mathcal{S}_{15}$ awaits their first message. Also, if *only* the proxy is corrupted, $\mathcal{S}_{15}$ awaits the leaked Subsession Identifier *ssid* from $\mathcal{F}^{(\Delta)}_{PUBA}$ to obtain $lin \coloneqq f_{\mathrm{LN}}(ssid)$ and to report the first message from $\mathcal{U}$ to $\mathcal{P}$.

H_{16} Introduces a map $f^{(\mathcal{O})}_{\mathrm{OI}}$ and for each proxy a map $f^{(\mathcal{P})}_{\mathrm{OI}}$ to be used by the simulator $\mathcal{S}_{16}$: $f^{(\mathcal{O})}_{\mathrm{OI}}$ is similar to what an honest operator would store for the Outsourced Information after the Outsource-task, $f^{(\mathcal{P})}_{\mathrm{OI}}$ is the equivalent for the respective proxy.
The map is used during the tasks for Outsource and Outsourced Analytics:

Outsource, $\mathcal{U}$, $\mathcal{P}$ and $\mathcal{O}$ honest. $\mathcal{S}_{16}$ adds a vector of empty entries $(\bot, \ldots, \bot)$ to both $f^{(\mathcal{P})}_{\mathrm{OI}}(k)$ and $f^{(\mathcal{O})}_{\mathrm{OI}}(k)$.

Outsource, $\mathcal{U}$ corrupted, $\mathcal{P}$ and $\mathcal{O}$ honest. lin^{new} is now randomly sampled by $\mathcal{S}_{16}$, which replaces the coin-toss of $\mathcal{O}$ and $\mathcal{P}$.

After receiving instructions from $\mathcal{Z}$ to send a message $(\mathrm{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{O})}_{in_\mathcal{U}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\alpha}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\mathbf{a}}, ser, \widetilde{\mathtt{com}_{id}}, \Pi, \mathtt{com}^{(\mathcal{U})}_{ser^{\mathrm{new}}})$ from a corrupted $\mathcal{U}$ to $\mathcal{O}$, $\mathcal{S}_{16}$ adds a new entry $(lin^{\mathrm{new}}, \mathrm{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{O})}_{in_\mathcal{U}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\alpha}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\mathbf{a}})$ to $f^{(\mathcal{O})}_{\mathrm{OI}}$.

After receiving instructions from $\mathcal{Z}$ to send a message $(\mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_\mathcal{U}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\mathbf{a}})$ from a corrupted $\mathcal{U}$ to $\mathcal{P}$, $\mathcal{S}_{16}$ adds a new entry $(lin^{\mathrm{new}}, \mathrm{sh}^{(\mathcal{P})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{P})}_{in_\mathcal{U}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\alpha}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{P})}_{\mathtt{o}_\mathbf{a}})$ to $f^{(\mathcal{P})}_{\mathrm{OI}}$.

Outsource, $\mathcal{U}$ and $\mathcal{O}$ honest, $\mathcal{P}$ corrupted. When the honest $\mathcal{U}$ is supposed to create shares of $\mathcal{UH}$, $in_\mathcal{U}$, $\mathtt{o}_{\mathtt{OUT}}$, $\mathtt{o}_{\mathtt{UNV}}$, $\mathtt{o}_\alpha$, $\mathtt{o}_\mathbf{s}$ and $\mathtt{o}_\mathbf{a}$, $\mathcal{S}_{16}$ creates shares of the zero-vector using π_{Share}-$Share(\mathbf{0})$ and uses those in the same way the user uses the actual shares in the protocol. After having the linking number lin^{new} created honestly, $\mathcal{S}_{16}$ stores the values in $f^{(\mathcal{O})}_{\mathrm{OI}}$ and ignores $f^{(\mathcal{P})}_{\mathrm{OI}}$.

Outsource, $\mathcal{U}$ and $\mathcal{P}$ corrupted, $\mathcal{O}$ honest. $\mathcal{S}_{16}$ follows the protocol of $\mathcal{O}$ regarding f_{OI}, that is, after $\mathcal{Z}$ sent instructions to send a message $(\mathrm{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{O})}_{in_\mathcal{U}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\alpha}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\mathbf{a}}, ser, \widetilde{\mathtt{com}_{id}}, \Pi, \mathtt{com}^{(\mathcal{U})}_{ser^{\mathrm{new}}})$ from $\mathcal{U}$ to $\mathcal{O}$ and after $\mathcal{S}_{16}$ took the role of $\mathcal{O}$ in honestly computing the linking number, it adds $(lin^{\mathrm{new}}, \mathrm{sh}^{(\mathcal{O})}_{\mathcal{UH}}, \mathrm{sh}^{(\mathcal{O})}_{in_\mathcal{U}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{OUT}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_{\mathtt{UNV}}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\alpha}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\mathbf{s}}, \mathrm{sh}^{(\mathcal{O})}_{\mathtt{o}_\mathbf{a}})$ to $f^{(\mathcal{O})}_{\mathrm{OI}}$.

H_{17} All remaining honest parties are replaced by dummy parties, which, upon receiving input by $\mathcal{Z}$, only forward their input into $\mathcal{F}^{(\Delta)}_{PUBA}$.

H_{18} All messages between honest parties are simulated by having $\mathcal{S}_{18}$ report messages of zero-vectors of correct size. Consequently, all operations that do not result in messages are removed.

H_{19} During the User Registration-task with an honest user, instead of honestly creating a user id *id*, interpreting it as secret key and computing a public key from it, $\mathcal{S}_{19}$ uniformly samples a public key $\mathsf{pk}_{\mathcal{U}} \xleftarrow{r} G_1$ directly.

H_{20} If $\mathcal{P}$ is corrupted and $\mathcal{U}$ and $\mathcal{O}$ are honest, $\mathcal{S}_{20}$ changes its behavior during simulation of the Outsource-task. Instead of calling π_{Verify} on the whole logbook, $\mathcal{S}_{20}$ only verifies that the linking number lin^{new} received by $\mathcal{P}$ confirms with the two shares $(lin^{\text{new}})^{(\mathcal{O})} + (lin^{\text{new}})^{(\mathcal{P})}$.

We now prove lemmata claiming indistinguishability of each consecutive pair of games against every PPT-environment $\mathcal{Z}$. Thus, with H_1 being the real world and H_{20} corresponding to the ideal world with a simulator acting as described above, we have proven the protocol π_{PUBA} to be as secure as the functionality $\mathcal{F}_{PUBA}^{(\Delta)}$.

Lemma G.23 (Indistinguishability of H_1 and H_2)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_1 and H_2 with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. $\mathcal{F}_{\text{PPA}}$, $\mathcal{F}_{\text{KE}}$ and $\mathcal{F}_{\text{BB}}$ are considered to be *hybrid functionalities*, which can be instantiated by a real, UC-secure protocol. UC-security of the instantiations implies, that $\mathcal{F}_{\text{BB}}$, $\mathcal{F}_{\text{KE}}$ and $\mathcal{F}_{\text{PPA}}$ all have a *simulator*, which runs in polynomial time and which can provide the view of a real protocol execution to any PPT-environment $\mathcal{Z}$, but is only using the ideal functionalities, for any given corruption scenario. Hence, $\mathcal{S}_2$ can execute this code without runtime restrictions, whenever an interaction between any party and either $\mathcal{F}_{\text{BB}}$, $\mathcal{F}_{\text{KE}}$ or $\mathcal{F}_{\text{PPA}}$ is requested by $\mathcal{Z}$.

Indistinguishability now easily follows from the security of $\mathcal{F}_{\text{BB}}$, $\mathcal{F}_{\text{KE}}$ and $\mathcal{F}_{\text{PPA}}$. Their UC-security implies, that the resp. protocol (which is used in H_1) can not be distinguished from the simulated view (which is used in H_2). If $\mathcal{Z}$ could differentiate H_1 and H_2 with non-negligible advantage over guessing, we can build a distinguisher $\mathcal{Z}'$ for the real and ideal view of the resp. hybrid functionalities as follows:

- $\mathcal{Z}'$ internally simulates all users $\mathcal{U}$, all proxies $\mathcal{P}$, the operator $\mathcal{O}$ and the environment $\mathcal{Z}$ that can distinguish H_1 and H_2 with probability $\frac{1}{2} + \varepsilon$ by executing their code in its head.
- $\mathcal{Z}'$ uses the distinguishing algorithm to let $\mathcal{Z}$ output a single bit: 0 for H_1 and 1 for H_2.
- $\mathcal{Z}'$ outputs this bit, where 0 means that it is in the real world and 1 means it is in the ideal world.

Note that the environment $\mathcal{Z}'$ also has a distinguishing advantage of $\frac{1}{2} + \varepsilon$. UC-security of $\mathcal{F}_{\text{BB}}$, $\mathcal{F}_{\text{KE}}$ and $\mathcal{F}_{\text{PPA}}$ thus require $\varepsilon \in negl(\kappa)$, which concludes the proof. □

Lemma G.24 (Indistinguishability of H_2 and H_3)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_2 and H_3 with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. The simulator mimics the behavior of an honest operator. Since none of the code regarding $\mathsf{L}_{\mathsf{SER}}$ is in any way dependent on the secret input $\mathcal{Z}$ gives to $\mathcal{U}$ or $\mathcal{O}$, this can be done without any loss of generality. In particular, assume that any party (that is, either $\mathcal{O}$ or $\mathcal{S}_3$) aborts due to a duplicate serial number in any game. In either case, this would mean that there was a previous interaction of either the task for Bookkeeping, Outsource, or Update, where $\mathcal{U}$ opened a value of $\mathtt{com}_{ser}$ to the same *ser* that is now seen by the respective party. Since the different parties execute the same code, their abort-criteria is equivalent. The same is true for the verification of the proof Π. □

Lemma G.25 (Indistinguishability of H_3 and H_4)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_3 and H_4 with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. Indistinguishability here holds for the same reason that it held in Theorem G.24. The contents of $f_{\text{UP}}^{(\mathcal{P})}$ and $f_{\text{UP}}^{(\mathcal{O})}$ only depend on *messages* which $\mathcal{S}_4$ can access via simulation of $\mathcal{F}_{\text{PPA}}$, and not (directly) on *secret input*, which is hidden from it. Hence, we only have a new encapsulation, where (PPT-)code that depends only on previous messages was executed by $\mathcal{O}$ or $\mathcal{P}$ in H_3 is now executed by $\mathcal{S}_4$ in H_4.

The equivalence of their contents easily follows from the same argument. The contents themselves depend on the message that $\mathcal{U}$ sent to the resp. parties, the order depends on the (adversarialy chosen) scheduling mechanism, which, given any environment $\mathcal{Z}$ that tries to distinguish the two games, is equivalent.

The final change contains the check in the corruption scenario where $\mathcal{P}$ is corrupted, but $\mathcal{U}$ and $\mathcal{O}$ are honest. Here, in H_3, $\mathcal{U}$ aborts if either $\mathsf{Unv}(\mathtt{com}_{\mathbf{a}}, \mathtt{unv}_{\mathbf{a}}, \mathbf{a}) \neq 1$, or if $\mathtt{c}_{out_{\mathcal{U}}}$ differs from the value the honest receiver sent.

Equivalence for the latter is straightforward. If $\mathcal{U}$ aborts due to the former condition ($\mathsf{Unv}(\mathtt{com}_{\mathbf{a}}, \mathtt{unv}_{\mathbf{a}}, \mathbf{a}) \neq 1$), then this means that some value was tampered with. $\mathtt{com}_{\mathbf{a}}$ was received by $\mathcal{O}$, who, in this scenario, is honest, hence it is correct. The One-Time Pads $\mathtt{o}_{\mathbf{a}}, \mathtt{o}_{\mathtt{UNV}}$ used to mask $\mathbf{a}$ and $\mathtt{unv}_{\mathbf{a}}$ were created by the user $\mathcal{U}$ during the Outsource task and hence are also correct. Thus, the only values that *could* be tampered with are $\mathtt{c}_{\mathbf{a}}$ and $\mathtt{c}_{\mathtt{unv}_{\mathbf{a}}}$, which $\mathcal{S}_4$ has seen during the Outsource task and hence, for which a consistency check suffices. □

Lemma G.26 (Indistinguishability of H_4 and H_5)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_4 and H_5 with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. Indistinguishability here follows from the trapdoor nature of POK. If any environment $\mathcal{Z}$ could distinguish the execution of the protocol when using *crs* created by $crs \leftarrow \mathsf{SetupPoK}$ from *crs* created by $(crs, td_{\mathsf{ext}}) \leftarrow \mathsf{SetupEPoK}$ with probability $\frac{1}{2} + \varepsilon$, we can build a PPT-environment $\mathcal{Z}'$ that breaks the indistinguishability of the dual-mode property of POK, by having $\mathcal{Z}'$ execute the code of all parties in its head. This leads to the same success probability of $\frac{1}{2} + \varepsilon$, thus causing $\varepsilon \in negl(\kappa)$ by requirement of the chosen POK-scheme. □

Lemma G.27 (Indistinguishability of H_5 and H_6)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_5 and H_6 with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. The only difference between H_5 and H_6 is, that $\mathcal{S}_6$ stores additional information in H_6 that was accessible even in H_5. Additionally, this game introduces a new abort-criteria.

For indistinguishability of H_5 and H_6, we thus have to show that those two criteria are, in fact, equivalent. In H_5, $\mathcal{O}$ fetches the key $\mathsf{pk}_{\mathcal{U}}$ belonging to $pid_{\mathcal{U}}$ from $\mathcal{F}_{\mathrm{BB}}$, thus effectively asking (since H_2) $\mathcal{S}_6$ for the key that $pid_{\mathcal{U}}$ registered there. Thus, instead of $\mathcal{S}_6$ simulating $\mathcal{F}_{\mathrm{BB}}$ and giving $\mathcal{O}$ the key $\mathsf{pk}_{\mathcal{U}}$ so $\mathcal{O}$ can verify that $\mathsf{pk}_{\mathcal{U}}$ is the one that was used in Π, $\mathcal{S}_6$ now does the exact same thing, only that, due to the simulation of $\mathcal{F}_{\mathrm{BB}}$, no further interaction is required to obtain $\mathsf{pk}_{\mathcal{U}}$. Hence, aborts in H_5 due to a duplicate or mismatching $\mathsf{pk}_{\mathcal{U}}$ occur if and only if aborts in H_6 happen due to a duplicate or mismatching $\mathsf{pk}_{\mathcal{U}}$. Thus, both distributions from H_5 and H_6 are equivalent. □

Lemma G.28 (Indistinguishability of H_6 and H_7)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_6 and H_7 with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. Extraction is possible due to the different *crs* introduced in H_5 and the fact that each witness only has elements from the target group, so F-extractability of POK suffices to obtain $\mathsf{pk}_{\mathcal{U}}$. The public key $\mathsf{pk}_{\mathcal{U}}$ is contained in every language used throughout the system. The user found by looking up $pid_{\mathcal{U}} \coloneqq f_{\mathrm{ID}}(\mathsf{pk}_{\mathcal{U}})$ is the correct user.

Assume for the sake of contradiction that $\mathcal{S}_7$ aborts in H_7 because $pid_{\mathcal{U}} = \bot$, but $\mathcal{S}_6$ would successfully terminate in H_6. The following behavior by $\mathcal{U}$ could have caused this:

- $\mathcal{U}$ **proved a faulty statement.** The correctness property of POK would cause this to be detected in H_6 by $\mathcal{O}$ with overwhelming probability, thus causing an abort.
- $\mathcal{U}$ **changed the commitment $\mathtt{com}_{id}$ in the statement.** Then the proof of equivalence between $\mathtt{com}_{id}$ and the rerandomized $\widetilde{\mathtt{com}_{id}}$ would have to be forged, thus breaking the statement of Π. This would also cause an abort in H_7, due to the correctness of POK.
- $\mathcal{U}$ **opened $\mathtt{com}_{id}$ to to a different $\mathsf{pk}'_{\mathcal{U}}$.** In this case, the binding-property of the commitment scheme COM would be violated. Since we assumed COM to be unconditionally binding, this cannot occur.
- $\mathcal{U}$ **created its own signature.** In case the user $\mathcal{U}$ created a signature σ on a new logbook λ' containing a public key $\mathsf{pk}'_{\mathcal{U}} \neq \mathsf{pk}_{\mathcal{U}}$ without knowing the operator's secret key $\mathsf{sk}_{\mathcal{O}}$, $\mathcal{U}$ would break the unforgeability of SIG. By requirement on SIG, this is possible only with probability negligible in κ.

The reverse is also true; if $\mathcal{O}$ discards the proof Π in H_6, then he does so too in H_7, as this part has not been changed. □

Lemma G.29 (Indistinguishability of H_7 and H_8)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_7 and H_8 with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. Indistinguishability of those two games trivially follows; the new machine is only accessible by *honest* parties, who follow the protocol. Their protocol description in H_8 does not include any access to $\mathcal{F}_{PUBA}^{(\Delta)}$. The

simulator doesn't access $\mathcal{F}_{PUBA}^{(\Delta)}$ either, so the two distributions of H_7 and H_8 are statistically close and hence, the best any PPT-environment $\mathcal{Z}$ can do is guessing. □

Lemma G.30 (Indistinguishability of H_8 and H_9)**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_8 and H_9 with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. Indistinguishability of the two games directly follows from the facts that (1) both tasks for Init and Sign Function Parameter that contain the TSA $\mathcal{T}$ are executed with an honest operator $\mathcal{O}$, and (2) none of the two aforementioned tasks contain any secret inputs of $\mathcal{T}$. Hence, the role of $\mathcal{T}$ can be played by $\mathcal{S}_9$, who creates the same message distribution. This implies indistinguishability based on the fact that in the view of $\mathcal{Z}$, the two games are equivalent. □

Lemma G.31 (Indistinguishability of H_9 and H_{10})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_9 and H_{10} with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. The rewriting is purely cosmetical, as essentially the same code is executed on different machines. In H_9, the commitment $\mathtt{com}_{fp}$ is computed by $\mathcal{O}$, the signature σ_{fp} is computed by $\mathcal{T}$, and the resp. values are fetched by $\mathcal{O}$ prior to a computation with $\mathcal{F}_{\text{PPA}}$. In H_{10}, all these steps are done by the simulator. Since $\mathcal{T}$ is honest and its key has thus been created by $\mathcal{S}_{10}$ and the relevant information for $\mathtt{com}_{fp}$ and σ_{fp} are leaked by $\mathcal{O}$, the honest code can be executed directly.

Finally, the last change induced is the replaced check with the decommitment information. Since this is essentially a rewriting, this change is purely cosmetical and hence undetectable. □

Lemma G.32 (Indistinguishability of H_{10} and H_{11})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{10} and H_{11} with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. $\mathcal{O}$ is a PPT-machine, which executes code based on the secret input from $\mathcal{Z}$. Knowing this input from the leak, $\mathcal{S}_{11}$ can execute the code of the honest $\mathcal{O}$ by following the protocol. This trivially leads to statistical indistinguishability. □

Lemma G.33 (Indistinguishability of H_{11} and H_{12})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{11} and H_{12} with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. The major change induced in this game hop has the simulator reporting valid zero-commitments instead of valid commitments on Function Parameters *fp*. Let $\mathcal{Z}$ be an environment that distinguishes the two games from H_{11} and H_{12}. We construct an adversary $\mathcal{A}$ that breaks the hiding property of COM', which we model similar to IND-CPA for encryption schemes. We adapt the LR-view, stating that the challenger $\mathcal{C}$ on the hiding game provides us with an oracle that accepts two different inputs, but outputs a valid commitment on a fixed one of them.

The adversary $\mathcal{A}$ can thus create the transcript from H_{11}, but whenever a commitment $\mathtt{com}_{fp}$ on the FPs *fp* is required, $\mathcal{A}$ sends the two messages $(\mathbf{0}, fp)$ to $\mathcal{C}$ and obtains a commitment on one of them.

Note that if the commitment always uses the former entry, $\mathcal{A}$ perfectly simulates H_{12}, and if the commitment always uses the latter entry, $\mathcal{A}$ perfectly simulates H_{11}.

It thus follows that the success probability of $\mathcal{Z}$ in detecting this game hop is limited by the probability of $\mathcal{A}$ to break the hiding game, which is negligible. □

Lemma G.34 (Indistinguishability of H_{12} and H_{13})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{12} and H_{13} with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. The case here is similar to that from Lemma G.32. $\mathcal{S}_{13}$ can execute the code of any honest user $\mathcal{U}$, since honest user reveal their identity to $\mathcal{S}_{13}$; by leaking the secret input, $\mathcal{S}_{13}$ can follow the protocol of $\mathcal{U}$ from H_{12}. Indistinguishability follows. □

Lemma G.35 (Indistinguishability of H_{13} and H_{14})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{13} and H_{14} with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. The proxy itself has no secrets, so no leaks are required here. Hence, all messages of $\mathcal{P}$ depend only on messages it has seen before. Since $\mathcal{S}_{14}$ can see those messages as well and $\mathcal{P}$ is a PPT-machine, $\mathcal{S}_{14}$ can execute the code of honest proxies, thus causing a statistically indistinguishable distribution from H_{13}. □

Lemma G.36 (Indistinguishability of H_{14} and H_{15})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{14}*

and H_{15} *with probability* $1/2 + \varepsilon$. *Then it holds that* $\varepsilon \in negl(\kappa)$.

Proof. Indistinguishability of H_{14} and H_{15} follows from the fact that in both cases the messages depend on exactly the same values. Essentially, the same code is executed, only by different machines; the game hop is only a cosmetical one.

In more detail, the situation looks as follows:

User Registration, $\mathcal{U}$ corrupted, $\mathcal{O}$ honest. Here, the definition of $\mathcal{F}_{PUBA}^{(\Delta)}$ and $\mathcal{S}_{15}$, who calls $\mathcal{F}_{PUBA}^{(\Delta)}$-*User Registration* with the correct inputs $in_{\mathcal{U}}$, implies correct behavior. The input $in_{\mathcal{U}}$ is correct due to the extractability and the pid is correct as a wrong pid would require the user to break the Co-CDH assumption by computing the id *id* of the public key $\mathsf{pk}_{\mathcal{U}}$ stored in $\mathcal{F}_{\text{BB}}$.

Bookkeeping, $\mathcal{U}$ corrupted, $\mathcal{O}$ honest. In H_{14}, the user holds its own logbook λ with the corresponding User History $\mathcal{UH}$. In H_{15}, $\mathcal{F}_{PUBA}^{(\Delta)}$ executes the same function Δ on the inputs to $\mathcal{F}_{PUBA}^{(\Delta)}$, that the simulator computed in H_{14}. In H_{15}, the correct – and latest – input is used by definition of the ideal functionality. The transfer values $(\alpha, \mathbf{s}, \mathbf{a})$ are output to $\mathcal{U}$ and hence visible to $\mathcal{S}_{15}$ and the commitment and decommitment on $\mathbf{a}$ can be computed directly by $\mathcal{S}_{15}$. If this value would have been the same in H_{14} then indistinguishability for any exectution of the Bookkeeping-task trivially follows; the best any PPT-environment $\mathcal{Z}$ could do here is to guess.

So assume that there is some set of inputs, for which $\mathcal{Z}$ can differentiate between H_{14} and H_{15} notably better than guessing based on the Bookkeeping task. Since the computation performed by the simulator in H_{14} is exactly the same as the one $\mathcal{F}_{PUBA}^{(\Delta)}$ does in H_{15} and $\mathcal{Z}$ can not lie about $in_{\mathcal{U}}$ (as it is input into $\mathcal{F}_{\text{PPA}}$), the only way $\mathcal{Z}$ could try to win here is by providing different input $\mathcal{UH}$. There are different ways $\mathcal{Z}$ could achieve that:

- $\mathcal{Z}$ provides a wrong proof regarding σ. Then, we could build an environment that either forges a signature σ and uses an honest witness, or fakes a proof Π_{Val} and uses a false witness. The former would contradict our EUF-CMA requirement for SIG, the latter would contradict the soundness-property required for POK.
- $\mathcal{Z}$ provides the correct information of a different corrupted user $\mathcal{U}'$. We assumed SIG to be EUF-CMA-secure, thus assuring unforgeability of the signature σ on λ. Hence, $\mathcal{S}_{15}$ would extract the id $\mathsf{pk}_{\mathcal{U}}$ of $\mathcal{U}'$. By a lookup from f_{ID}, $\mathcal{S}_{15}$ would get the *pid* of $\mathcal{U}'$ and provide input to $\mathcal{F}_{PUBA}^{(\Delta)}$ in the name of $\mathcal{U}'$. Thus, $\mathcal{F}_{PUBA}^{(\Delta)}$ uses the same $\mathcal{UH}$, that would have been used by the simulator in H_{14}.

Hence, assuming that $\mathcal{F}_{PUBA}^{(\Delta)}$ internally updates $\mathcal{UH}$ correctly (which we will show for the other tasks as well), this change can not be used to increase the chance of $\mathcal{Z}$ to differentiate.

Outsource, $\mathcal{U}$ corrupted, $\mathcal{O}$ and $\mathcal{P}$ honest. Here, too, indistinguishability trivially follows from the correctness of inputs; we merely copied the reconstruction of shares for this scenario from Outsourced Analytics to Outsource. Those are input directly into $\mathcal{F}_{PUBA}^{(\Delta)}$, where they are used later. The shares $\text{sh}_{in_{\mathcal{U}}}^{(\mathcal{P})}$ and $\text{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}$ are either correct, in which case $\mathcal{F}_{PUBA}^{(\Delta)}$ will use them accordingly during Outsourced Analytics. Or they are not, in which case $\mathcal{S}_{15}$ sets $in_{\mathcal{U}} := \perp$; the reconstruction H_{14} would fail during the Outsource-task, which also happens in H_{15}. With the two parties performing Outsourced Analytics being honest, no further problems arise during the subsequent execution. The extraction of $\mathsf{pk}_{\mathcal{U}}$ further removes the ability of $\mathcal{Z}$ to cheat by letting $\mathcal{S}_{15}$ send input to $\mathcal{F}_{PUBA}^{(\Delta)}$ in the name of the wrong $\mathcal{U}$.

Outsource, $\mathcal{U}$ and $\mathcal{O}$ honest, $\mathcal{P}$ corrupted. In this case, it is not possible to cheat without being detected. By knowing the input $\mathcal{Z}$ would have given to $\mathcal{P}$, $\mathcal{S}_{15}$ can mimic the behavior of an honest dummy proxy and forward it into $\mathcal{F}_{PUBA}^{(\Delta)}$.

Outsource, $\mathcal{U}$ and $\mathcal{P}$ corrupted, $\mathcal{O}$ honest. The correct user $\mathcal{U}$ can be determined via extraction of Π, so no $\mathcal{Z}$ cannot use two different users for the two games here. However, in this task, $\mathcal{S}_{15}$ information-theoretically can not determine the correct input $in_{\mathcal{U}}$, as it only sees one part $\text{sh}_{in_{\mathcal{U}}}^{(\mathcal{P})}$ of the additive sharing—the second part, $\text{sh}_{in_{\mathcal{U}}}^{(\mathcal{O})}$, is sent between two corrupted parties and hence not visible for $\mathcal{S}_{15}$. However, in the subsequent Outsourced Analytics execution, $\mathcal{S}_{15}$ learns the shares that $\mathcal{U}$ sent to $\mathcal{P}$ via simulation of $\mathcal{F}_{\text{PPA}}$. There, $\mathcal{S}_{15}$ can reconstruct $in_{\mathcal{U}}$. Note that there is no difference between $\mathcal{S}_{15}$ learning the input during the Outsource task and inserting it into $\mathcal{F}_{PUBA}^{(\Delta)}$ right away and $\mathcal{S}_{15}$ learning it during the Outsourced Analytics task, since $\mathcal{F}_{PUBA}^{(\Delta)}$ allows $\mathcal{S}_{15}$ to update the input $in_{\mathcal{U}}$ for all corrupted users before starting the computation during Outsourced Analytics. Thus, the same input $in_{\mathcal{U}}$ is used in both games.

Outsourced Analytics, $\mathcal{P}$ corrupted, $\mathcal{O}$ honest. Here, $\mathcal{S}_{15}$ still uses the same shares for the operator in both games and receives (and verifies) all proxy-shares input into $\mathcal{F}_{\text{PPA}}$ in both games, there is no direct change here. The only new thing is the equivocation of $in_{\mathcal{U}}$ for corrupted users. This change was already discussed above; the remaining protocol remains equivalent, since $\mathcal{S}_{15}$ only does in H_{15} what honest parties would do in H_{14}.

Update, $\mathcal{U}$ corrupted, $\mathcal{P}$ and $\mathcal{O}$ honest. The values that determine the messages are still equivalent in both games. The values for $f_{\text{UP}}^{(\mathcal{P})}$ and $f_{\text{UP}}^{(\mathcal{O})}$ were honestly kept by $\mathcal{S}_{15}$, so both just follow the same protocol. As nothing is done with the output of $\mathcal{F}_{PUBA}^{(\Delta)}$, distinguishing here is not possible.

Update, $\mathcal{U}$ and $\mathcal{O}$ honest, $\mathcal{P}$ corrupted. The linking number that was sent as leak by $\mathcal{U}$ to the simulator in H_{14} trivially equals $f_{\text{LN}}(ssid)$ that was kept by $\mathcal{S}_{15}$ in H_{15}. During the Outsource task, assuming *lin* is sampled from a sufficiently large space, then the probability that the mapping $lin \to ssid$ is unique becomes overwhelming. During the Update task, this does not change. If this wouldn't be the case, $\mathcal{Z}$ would have to create a duplicate *lin*, which, with Blum coin toss, is possible only with negligible probability. Even then, $\mathcal{S}_{15}$ in H_{15} would use the correct linking number *lin*. Note further that $\mathcal{Z}$ has no way on how to lie about *ssid*.

Update, $\mathcal{U}$ and $\mathcal{P}$ corrupted, $\mathcal{O}$ honest. Here, the simulator only has to ensure that the interaction between $\mathcal{O}$ and $\mathcal{U}$ is canonical. Again, nothing here depends on the output of $\mathcal{F}_{PUBA}^{(\Delta)}$, it is only called to keep the functionality in a consistent space. Hence, the changes induced here provide no advantage to $\mathcal{Z}$ in distinguishing the two games.

□

Lemma G.37 (Indistinguishability of H_{15} and H_{16})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{15} and H_{16} with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. We claim indistinguishability based on the fact that the two distributions from H_{15} and H_{16} are indistinguishable.

First, notice that the binary outcome of a Blum coin toss and a uniformly random bit cannot be differentiated better that by guessing, so this change doesn't provide any distinguishing advantage.

To support our claim also regarding the new map f_{UP}, let's consider all possible corruption cases:

$\mathcal{P}$ honest, $\mathcal{O}$ honest. In case $\mathcal{U}$ is also honest, the functionality $\mathcal{F}_{PUBA}^{(\Delta)}$ directly obtains input from the respective parties during the Outsource task, which causes $\mathcal{F}_{PUBA}^{(\Delta)}$ to load the correct $\mathcal{UH}$ and to use the input $in_{\mathcal{U}}$ provided by $\mathcal{U}$. Hence, the simulator has to only remember *that* honest parties provided input, not *what* these inputs were. Thus, inserting the special symbol $\perp$ to keep the list size consistent suffices, since those values are never used again.

In case $\mathcal{U}$ is corrupted, $\mathcal{Z}$ has to send shares $\text{sh}_{(\cdot)}^{(\mathcal{P})}$ and $\text{sh}_{(\cdot)}^{(\mathcal{O})}$ in the name of $\mathcal{U}$ to $\mathcal{P}$ and $\mathcal{O}$, respectively. Neither of them are corrupted, so $\mathcal{S}_{16}$ can see both messages. Since neither $\mathcal{O}$ nor $\mathcal{P}$ have secret inputs, $\mathcal{S}_{16}$ can follow the honest protocols, thus producing the same distribution.

$\mathcal{P}$ corrupted, $\mathcal{O}$ honest. If the user $\mathcal{U}$ is honest, then $\mathcal{S}_{16}$ has to send messages containing valid shares of $\mathcal{U}$s input, without actually knowing the input. Thus, $\mathcal{S}_{16}$ distributes zero-shares; the environment $\mathcal{Z}$ can see only the part sent to $\mathcal{P}$, not the one sent to $\mathcal{O}$. Obviously, no environment $\mathcal{Z}$ can distinguish its part of the zero-sharing obtained in H_{16} from a valid sharing of the respective user input from H_{15} better than by randomly guessing. Hence, those distributions look equivalent. The simulator follows the protocol of $\mathcal{O}$ regarding f_{OI} honestly, which causes no difference in the distributions.

Against a *corrupted* user $\mathcal{U}$, $\mathcal{S}_{16}$ directly follows the protocol of $\mathcal{O}$, thus causing the same distribution.

Since all possible cases cause indistinguishable distributions, our claim follows. □

Lemma G.38 (Indistinguishability of H_{16} and H_{17})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{16} and H_{17} with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. Again, indistinguishability follows from the fact that the messages sent are still the same. All the leaks sent by honest parties to the simulator in H_{16} are ignored, the messages are independent of any leaks still sent by the semi-dummy parties: the only change between H_{16} and H_{17} is with respect to $in_{\mathcal{O}}$ and $in_{\mathcal{U}}$. Both are input to $\mathcal{F}_{PUBA}^{(\Delta)}$ directly by the respective dummy party and used there accordingly. Hence, the change in-

duced by H_{17} doesn't change the view of $\mathcal{Z}$ at all, which makes indistinguishability trivial. □

Lemma G.39 (Indistinguishability of H_{17} and H_{18})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{17} and H_{18} with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. We generally assume that parties obtain pairwise shared keys at the beginning of each task from $\mathcal{F}_{\text{KE}}$. This yields a key k, which they use for symmetric encryption with the IND-CPA-secure symmetric encryption scheme ENC. Assume for the sake of contradiction that there is a PPT environment $\mathcal{Z}$, which can distinguish between H_{17} and H_{18} with advantage $\frac{1}{2} + \varepsilon$ for $\varepsilon \notin negl(\kappa)$. We show that this implies an adversary $\mathcal{A}$ on the IND-CPA experiment of ENC:

We adapt the LR-view, that provides the reduction algorithm with an algorithm that on input m_0, m_1 either always outputs $\mathsf{ENC.Enc}(\mathsf{sk}, m_0)$, or $\mathsf{ENC.Enc}(\mathsf{sk}, m_1)$.

$\mathcal{Z}$ distinguishes H_{17} and H_{18}, and $\mathcal{A}$ can simulate the views correctly, by providing always the oracle output on input $(m_0, \mathbf{0})$, where m_0 is the honest transcript; If the adversary outputs H_{18}, $\mathcal{A}$ sends 0 to $\mathcal{C}$. If the adversary outputs H_{17}, $\mathcal{A}$ outputs 1 to $\mathcal{C}$.

Note that $\mathcal{A}$ has the same success probability as $\mathcal{Z}$, that is, $\frac{1}{2} + \varepsilon$. Hence, by our assumption, it follows that $\varepsilon \in negl(\kappa)$, which concludes our proof. □

Lemma G.40 (Indistinguishability of H_{18} and H_{19})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{18} and H_{19} with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. By assumption on the group gp, a public key is uniformly distributed in G_1. Any environment $\mathcal{Z}$ distinguishing H_{18} and H_{19} based on $\mathsf{pk}_{\mathcal{U}}$ would violate this assumption. The simulator now lacks knowledge of the corresponding user id *id*. However, note that *id* is never used in H_{18}, so $\mathcal{S}_{19}$ can create a similar distribution in H_{19} independently of *id*. □

Lemma G.41 (Indistinguishability of H_{19} and H_{20})**.** *Let $\mathcal{Z}$ be a PPT-environment. Let $\mathcal{Z}$ distinguish H_{19} and H_{20} with probability $1/2 + \varepsilon$. Then it holds that $\varepsilon \in negl(\kappa)$.*

Proof. Indistinguishability easily follows from the fact that in H_{19}, the only corrupted party is $\mathcal{P}$ and the only value depending on the proxy is *lin*. Hence, π_{Verify} only aborts, iff $\mathcal{P}$ sent a wrong linking number *lin*. This is still the case in H_{20}, thus making indistinguishability trivial. □

Thus, we can now finally prove our final security statement:

Corollary G.42 (System Security)**.** *For all environments $\mathcal{Z}$ who statically corrupted a subset $\mathcal{U}' \subseteq \mathcal{U}$ and the proxy $\mathcal{P}$, it follows that*

$$\pi_{PUBA}^{(\mathcal{F}_{\text{PPA}}, \mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{KE}}, \mathcal{F}_{\text{CRS}})} \geq_{UC} \mathcal{F}_{PUBA}^{(\Delta)}$$

We have shown in Lemma G.23 to Lemma G.41, that for an honest operator, the simulator $\mathcal{S}_{\text{SysSec}}$ acting in the ideal world can provide a view for $\mathcal{Z}$ that is indistinguishable from a real execution of the protocol:

$$view_{\mathcal{Z}, \mathcal{A}, \pi_{PUBA}} \approx^c view_{\mathcal{Z}, \mathcal{S}_{\text{SysSec}}, \mathcal{F}_{PUBA}^{(\Delta)}}$$

Thus, by combining Corollaries G.22 and G.42, our main claim follows:

Corollary G.43 (Security)**.** *Assuming that SIG is an EUF-CMA-secure structure preserving signature scheme, SIG' is an EUF-CMA-secure signature scheme, POK is a trapdoor dual-mode NIZKPoK scheme, $\mathsf{gp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p, g_1, g_2)$ is a pairing-group where the Co-CDH assumption is hard, COM and COM' are unconditionally hiding (and computationally binding) homomorphic commitment scheme, where images of COM can efficiently be inverted, then for all environments $\mathcal{Z}$ who do not corrupt the operator $\mathcal{O}$* and *the proxy $\mathcal{P}$ at the same time, it holds that*

$$\pi_{PUBA}^{(\mathcal{F}_{\text{PPA}}, \mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{KE}}, \mathcal{F}_{\text{CRS}})} \geq_{UC} \mathcal{F}_{PUBA}^{(\Delta)}$$

We thus have proven our protocol π_{PUBA} to be at least as secure as the ideal functionality $\mathcal{F}_{PUBA}^{(\Delta)}$.