Interoperable Private Attribution: A Distributed Attribution and Aggregation Protocol

Benjamin M Case[1], Richa Jain[1], Alex Koshelev[1], Andy Leiserson[2], Daniel Masny[1], Ben Savage[1], Erik Taubeneck[1], Martin Thomson[2], and Taiki Yamaguchi[1]

[1]*Meta*
[2]*Mozilla*

Abstract

Measuring people's interactions that span multiple websites can provide unique insight that enables better products and improves people's experiences, but directly observing people's individual journeys creates privacy risks that conflict with the newly emerging privacy model for the web. We propose a protocol that uses the combination of multi-party computation and differential privacy that enables the processing of peoples' data such that only aggregate measurements are revealed, strictly limiting the information leakage about individual people. Our primary application of this protocol is measuring, in aggregate, the effectiveness of digital advertising without enabling cross-site tracking of individuals. In this paper we formalize our protocol, Interoperable Private Attribution (IPA), and analyze its security. IPA is proposed in the W3C's Private Advertising Technology Community Group (PATCG) [8]. We have implemented our protocol in the malicious honest majority MPC setting for three parties where network costs dominate compute costs. For processing a query with 1M records it uses around 18GiB of network which at $0.08 per GiB leads to a network cost of $1.44.

1 Introduction

Accurate measurements about how complex systems are performing is necessary to keep them running properly and improve them over time. There are many application areas where this is important. Traditionally, measuring many such systems involved collecting and sharing extensive amounts of individual level user data. Much of the time, analysis is done on aggregated data, and individual-level data is collected only to be used in computing aggregates. Privacy can be further improved by producing these aggregates with technical means to prevent exposing or sharing individual-level data. Multi-party com-

putation and differential privacy provide that technical capability in several important applications that have seen large-scale real world deployments:

1. *Health statistics*: The COVID-19 Exposure Notification system developed jointly by Apple and Google includes a Private Analytics system that informs health authorities about how effectively the system is being used [11].

2. *Identifying malicious origins*: Mozilla's Origin Telemetry project helps browser vendors to identify malicious web pages through aggregate measurements without exposing users' browsing history [24].

3. *Advertising measurement*: Meta's private ads measurement products allow a Publisher and Advertiser to privately compute statistics about the effectiveness of advertising campaigns. [5, 7].

The IETF's Privacy-Preserving Measurement working group [8] is working to standardize the MPC behind first two of these two which utilize a particular variety of MPC protocols known as Verifiable Distributed Aggregation Functions (VDAFs) [19, 13, 20]. VDAFs core functionality is aggregation across inputs from many users, enabling a useful but still limited class of functions including sums, mean, standard deviation, linear regression [19], a step of gradient descent [27] and heavy hitters [13]. One of the core differences in this current work is the addition of an attribution step before aggregation. Attribution allows for conditional aggregations based on the inputs from other reports (see below), enabling more expressive functions to be computed.

The MPC protocols for advertising measurement mentioned above use a private join [15, 14, 2, 21] followed by some general purpose MPC [23, 9]. These protocols do support attribution, but this current work differs in two main ways 1) we support inputs coming not just from two parties but from many client devices, and 2) we target a stronger malicious security threat model instead of semi-honest security.

1.1 Attribution Measurement

Attribution measurement is a standard measurement approach to understand how a person's interactions in one context (e.g. website/app) affect their interactions in a different context (e.g. website/app). Attribution measurement can have many applications but we will specifically focus on how it is used in online digital advertising. For a given conversion event (e.g., a purchase) a person has on one website, we aim to attribute that conversion to zero or more ad impressions the person had the opportunity to see in a different context.

Abstractly, we call ad impressions *source events* and conversions *trigger events*. These events are associated with other data, such as an identifier for the person, an identifier for the campaign and/or product, a timestamp, etc. We call the collection of data associated with an event a *report*; an attribution measurement can be constructed as a query over a set of *source reports* and *trigger reports*.

In a typical case, there may be many source and trigger reports associated with the same person. There are various approaches for attributing source reports and trigger reports; for simplicity, in this paper we currently only address last touch attribution, i.e., trigger reports are attributed to the most recent source event in the past from the same person, if one exists. Our protocol could be adapted to allow for more complex attribution logic, but we leave that for future work.

Trigger reports carry with them a trigger value, which will be aggregated across all attributed trigger reports. Source reports include a breakdown key used to specify the granularity of aggregations done after attribution. The end result of attribution measurement is aggregates (sums of the trigger values), grouped by the breakdown keys from the source reports to which each trigger report was attributed.

1.2 Related ads measurement works

Related APIs for private ads measurement include Apple's Private Click Measurement (PCM) [4] and Google's Attribution Reporting API (ARA)[3].

Apple's "Private Click Measurement" system has already been deployed in the Safari browser. In this system, attribution is performed on-device, and information about attributed conversions is sent to websites. There is no MPC or server-side aggregation in this system, nor is there any differential privacy guarantee. Instead, heuristics are employed to reduce the linkability of attributed reports to individuals. These heuristics include adding a random time delay to reports, and reducing the total entropy of these reports.

Google's "Attribution Reporting API with Aggregate Reports" also performs on-device attribution, and emits randomly time-delayed and entropy limited reports. However, in this proposal the reports are encrypted and must be aggregated with a server-side aggregation system. While this aggregation system could be realized with an MPC, the current proposal involves the use of a "Trusted Execution Environment" operated on a cloud provider. This proposal also involves the addition of differential privacy and managing of a per-epoch privacy budget.

For a more detailed comparison of these APIs see [10].

2 IPA overview

Interoperable Private Attribution (IPA) is a proposal introduced by Meta and Mozilla for a new web platform API to support advertising attribution measurement. It is one proposed standard for the **private measurement** work item in the W3C's Private Advertising Technology Community Group (PATCG). While we believe the core MPC protocol for attribution and aggregation could have useful applications beyond advertising measurement, the advertising measurement application is our primary motivation. As such, we do not aim to design a

fully abstract solution, but instead lean into the complexity of fitting this protocol into an overall system design that could support advertising measurement on the web.

An attribution measurement can be viewed as a query over source reports and trigger reports, where these reports are matched according to a specific attribution logic and an aggregation (e.g., a sum across breakdowns) is performed on attributes from the source and trigger reports. Nearly all forms of attribution logic attempt to match source reports and trigger reports conditional on representing events related to the same person. In the past, this was accomplished through technical mechanisms provided by the user agent, specifically 3rd party cookies and device identifiers.

IPA proposes a new mechanism, called a match key, which is never directly revealed by the user agent. Instead, match keys only leave the user agent as encrypted secret-shares, encrypted towards three non-colluding helper parties. We call this collection of three helper parties a *helper party network*. Helper parties in a helper party network are trusted to not collude and only perform a predetermined MPC protocol that produces an aggregate and differentially private attribution measurement result.

2.1 Parties Involved

There are a number of different parties that are involved with the protocol:

1. User Agent (UA): a software application, such as browser or app , that acts on behalf of a user when communicating with a website or server
2. User Agent Vendor (UAV): The distributor of the user agent (e.g. Mozilla)
3. Helper Party: A party who participates in the MPC protocol.
4. Helper Party Network (HPN): A set of three helper parties. We aim to find helper party networks which are trusted by user agent vendors to be non-colluding.
5. Websites/apps: An individual website (identified by its eTLD+1) or an individual app (identified by the mechanism for the OS it operates on.)
6. Match key providers (MKPs): Websites/apps which call the set match key API, typically websites/apps with a large logged-in set of users.
7. Source websites/apps: Websites/apps on which a user produces a source event.
8. Trigger websites/apps: Websites/apps on which a user produces a trigger event.
9. Report collectors (RCs): A specific website/app or a delegate acting on their behalf, that issues queries to the helper party network.

2.2 Match keys

For IPA we assume there will be unique match keys for each person that can be processed under encryption in the MPC. We are currently considering two main methods for setting match keys:

1. set by a Match Key Provider
2. generated randomly by the device.

For using a MKP, a UAV will need to approve certain sites to be MKP who are allowed to call a set matchkey API. The idea is that some sites which people sign into can assign a match key consistently to users on each device or user agent that person signs into. The matchkey is stored by the user agent as a write-only value that cannot be read back in the clear by anyone. The current threat model for MKPs is semi-honest; hence the need to have them specifically approved by a UAV. It is still an ongoing area of research to support MKPs in a malicious threat model. This method of setting a match key supports the greatest level of interoperability across different platforms.

For device generated matchkeys, each device will generate a random matchkey and store it similarly as a value that cannot be read back in the clear by anyone. Interoperability and cross device consistency of matchkey can still be supported if a user has multiple devices that already share a common secret that can be used to derive a consistent matchkey for that user.

When people visit sites each site can request a secret-shared and encrypted copy of the match key from the user agent for use by the IPA MPC protocol. Each match key secret share is encrypted toward two of the three helper parties such that each helper party can decrypt a distinct pair of shares. These helper parties only need to be trusted not to collude. Our protocol assumes an honest majority in a network of three helper parties, which allows us to provide full security with reasonable efficiency. A governance system will need to exist to establish a set of trusted helper party networks. An existing governance model from which we might derive inspiration is the CA/Browser Forum[1]. For the purpose of this paper, we assume that such a helper party network exists.

2.3 Collecting Reports

IPA begins with different websites collecting information on important events. Sites define what events are interesting to them. As sites are able to request an encrypted match key at any point, information about events (e.g. timestamp and event-type) can be stored along with the encrypted match key associated with the person currently visiting the site. Our goal is to aggregate reports from events that occur across different websites/apps. We define a *report collector* as a party that gathers these reports and assembles them into a query. The bulk of the information that is collected into reports for each event is defined by the website/app and is not observed by the user agent.

For gathering reports there are three steps:

1. **Getting the encrypted matchkey from the user agent:** At any point when a person visits a site or using an app, the website/app can call the get encrypted match key API and get back encrypted secret shares of the match key along with some authenticated associated data about the request. The encrypted match key is bound to certain associated data, including a bit indicating source/trigger, which match key provider's match key was requested, the site/app that requested the match key, and an epoch. This associated data will be used by the helper party networks to enforce the differential privacy budgets which we discuss in more detail below.

2. **Constructing reports:** Information known to the website/app about this event includes a timestamp, a conversion value for trigger reports, and a breakdown key for source reports. Source and trigger reports are built by combining these attributes with an encrypted match key. Most of this data also gets secret shared and encrypted under the public keys of the helper party network. Timestamps remain in the clear (authenticated as associated data) as later we want the report collector who submits the query to first sort the reports in a query by timestamp in the clear before submitting the query. Constructing a report from this data could happen on the client or a server.

3. **Sharing reports with other report collectors:** Once source and trigger sites have generated reports, they need to exchange them with a report collector who uses them to issue queries to a helper party network.

2.4 Overview of MPC Security Model

Our goal is to design a system that web platform vendors are willing to implement and will be technically and economically viable to operate. To this end, we've attempted to find the right balance between the assumption in our security model and performance characteristics of the protocol. The balance we are currently proposing is that a web platform vendor which implements this API only needs to trust the following condition in order to have assurance that the API does not enable cross-context tracking:

1. At least two out of three helper parties in the helper party network are honest, meaning that they will not collude with each other, and will run the correct protocol and enforce the DP budgeting system correctly.

We are proposing a 3-party, malicious, honest majority MPC protocol such that even if one of the helper parties actively tries to attack the protocol, they will be unable to learn any of the sensitive inputs and any actively malicious behavior will be detectable. More specifically we have implemented the protocol using replicated secret sharing [17]. This three party, honest majority setting allows for very efficient MPC protocols that can achieve malicious security at a reasonable cost over semi-honest security [17, 22].

2.5 Differential Privacy

Any system that reveals aggregate measurements about people leaks information about those people over time. Our privacy goal is to limit the amount of information that can be learned by a website/app about a single person's interactions over a specific period of time, an epoch (e.g., week). Aggregation or k-anonymity alone are insufficient [18] to guarantee such a bound so we rely on differential privacy to formally analyze and bound how much user information will potentially be revealed.

Below, we describe how to achieve ε, δ differential privacy for an individual query and how to manage a privacy budget across queries over the course of an epoch.

We define two types of queries: *source fan-out queries* with source reports from a single website/app and trigger reports from many websites/apps, and *trigger fan-out queries* with trigger reports from a single website/app and source reports from many websites/apps. Each site is issued a privacy budget which is consumed through these queries. The privacy budget for a source fan-out query is drawn from the budget of the singular source website/app, and trigger fan-out queries draw from the privacy budget of the singular trigger website/app.

2.5.1 Sensitivity Capping

In order to provide differential privacy at the level of people's individual contributions (as identified by match keys), each match key must be limited in the total contribution it can make across all aggregate outputs of a query. This maximum contribution is the sensitivity of the query, and together with the ε, determines the amount of noise required to achieve differential privacy.

We allow for the sensitivity cap to be provided as a parameter to the query. If the total contribution for a given match key exceeds this cap, it will be clipped at the cap. The exact value of what is lost due to this capping will be unknown to all entities involved, including the helper parties and the report collector. For example, a report collector might set a maximum contribution of \$100, but would be unaware how many users (if any) exceeded that cap and by how much the cap was exceeded. The relationship between this sensitivity cap and the differentially private noise added to the aggregate outputs results in a bias-accuracy tradeoff: a higher cap reduces bias introduced by clipping, but reduces accuracy due to larger relative noise added to provide differential privacy. Allowing this as a query parameter allows each report collector to make their own decision with regards to this tradeoff.

Note that because individual contributions are capped, our protocol also provides robustness against malicious inputs over that cap, meaning that a single report cannot corrupt the result of a query by contributing unbounded trigger values.

2.5.2 Differentially Private Noise

The output of each query is a sum per breakdown key. The contribution from a single person is capped across all breakdown keys, but this contribution can be allocated to a single breakdown key or distributed across multiple breakdown keys. Consequently, the sensitivity of the value of each breakdown is determined by the global sensitivity cap. Random noise is added to each breakdown, using ε and the sensitivity to inform the variance of the noise distribution. Noise will be added to each breakdown sum to provide global differential privacy. The exact form $(\varepsilon, \varepsilon - \delta)$, noise distribution (e.g. Laplace, Gaussian) and method of application (in-MPC, by helpers) has not yet been determined. We will need to consider various differential privacy composition theorems for when we reveal multiple outputs in a single query and for multiple queries.

2.5.3 Differential Privacy Budget Management

The previous section focuses on applying differential privacy to individual queries. However, we need to further design a system that is differentially private over all queries issued by report collectors on behalf of a given website/app in an epoch. Specifically, we propose that for a given epoch and website/app (represented by a single report collector), people (represented by match keys) can have a bounded amount of information released impact on the results, as measured by $\varepsilon - \delta$ differential privacy.

In our current approach, we achieve this by providing each report collector with a budget, ε for the given website/app they are querying on behalf of. When report collectors run queries they will specify how much budget to use, ε_i, which will be deducted from their remaining budget. For example, given an epoch limit of ε, a report collector could perform 10 queries, each with global differential privacy applied at $\varepsilon/10$, or a more complicated set of queries such as three with $\varepsilon/5$ and four with $\varepsilon/10$.

The helper party network will need to maintain this budget, per website/app, over the course of an epoch, preventing report collectors from issuing additional queries once that budget is exhausted. At the beginning of the next epoch, every report collector's budget will refresh to ε. Additionally, report collectors will need to be bound to a single helper party network for the duration of an epoch, to prevent double spending. The system for establishing these bindings is beyond the scope of this work. A site will make a commitment to only use a certain MKP through an epoch. This prevents them from running the same events but with different MKPs to increase their budget.

2.5.4 Enforcing DP Budgets

Each site has a certain DP budget per epoch. The tracking and enforcement of the budgets will be done by the Helper Party Networks.

As mentioned above there are two types of queries which can be issued by a report collector:

1. A source fanout query is designed to help a website/app understand the effect that the ads it shows have on outcomes for its advertisers. A source fanout query can only contain source reports from a single source website/app. A source fanout query can include trigger reports from multiple sites, where each report might represent a conversion event.

2. A trigger fanout query helps a site that buys ads to understand how its advertising is helping to drive outcomes on its website/app. A trigger fanout query can only contain trigger reports from a single trigger website/app. A trigger fanout query includes source reports from multiple sites, where each report might represent an ad impression or click.

The budget is associated with the website/app that provides source reports for a source fanout query or the website/app that provides trigger reports for a trigger fanout query. A website/app can provide source reports for trigger fanout queries on any other website/app without expending their budget; similarly, no budget is spent by providing trigger reports to another website/app for source fanout queries.

The information in the associated data with an encrypted matchkey allows the Helper Parties to enforce the checks to make sure queries are well formed according to these constraints. The associated data consists of which MKP's matchkey was requested, was the matchkey requested for a source or trigger report, what site requested the encrypted matchkey, what epoch is it. This information allows the HPN to check that when running a source-fan-out query all the source events were created on the site who is submitting the query and whose budget is being consumed. Similarly, for a trigger-fan-out query all the trigger events must have been created on the site that is submitting the query and whose budget is being consumed.

2.6 Query Overview

To initiate a query, a report collector collects a batch of source and trigger reports into a source fan-out or trigger fan-out query. The RC sorts them by their timestamps before submitting.

The Helper Parties then execute the following main stages of the query phase.

1. **Local Decryption:** Parties decrypt the shares they were sent encrypted under their public keys

2. **Verify:** The Helper Parties ensure that there is DP budget available for the RC submitting the query and that the query is a well formed source fan-out or trigger fan-out query. Helper Parties start the MPC by running a verification protocol to ensure all shares are well formed and that none of the Helpers have modified any of their original replicated shares.

3. **Sort:** Next they compute a stable sort by the value of the match keys using a radix sort in MPC [12]. This groups together all the reports belonging

to the same match key and because the sort is stable each match key's reports are sorted by timestamp.

4. **Attribution:** Next they run a last touch attribution algorithm that assigns the credit for a trigger report's value to first source report that occurred early in time with the same match key. The oblivious algorithms for this attribution step and next capping step use a parallel prefix sum computation similar to the approach used in [25].

5. **Capping:** Capping computes the total contribution of each person to the measurement result across all breakdowns (histogram buckets). If this contribution exceeds the per-person cap, we drop their contribution down to the cap.

6. **Aggregation:** Aggregation sums the attributed trigger values into separate histogram buckets according to the breakdown key of the source report they were attributed to.

7. **DP noise addition:** Finally, we apply DP noise to the aggregate for each breakdown and reveal the result to the RC who submitted the query.

3 Preliminaries

3.1 Notations

For $n \in \mathbb{N}$, we use $[n]$ to denote set $\{1, \ldots, n\}$. For any two elements a, b of a set S, we define operator $(a = b)$ which is 1 if $a = b$ and 0 otherwise.

For a bitstring $s \in \{0,1\}^*$, we define operators $\mathsf{AND}(s) : \{0,1\}^* \to \{0,1\}$, $\mathsf{OR}(s) : \{0,1\}^* \to \{0,1\}$, $\neg(s) : \{0,1\}^* \to \{0,1\}^*$ as the and, or and negation of all bits of s.

We use log to denote the logarithm for basis 2.

3.2 Cryptographic Primitives

Pseudorandom Functions

Definition 1 (Pseudorandom Function)**.** *We call a function* $\mathsf{PRF} : \mathsf{K} \times \mathsf{D} \to \mathsf{I}$ *a secure pseudorandom function if and only if for any* ppt *adversary* A *with query access to* $\mathcal{O}_{\mathsf{PRF}}(\mathsf{k}, \cdot)$ *(*$\mathcal{O}_u(\cdot)$*),*

$$|\Pr[\mathsf{A}^{\mathcal{O}_{\mathsf{PRF}}(\mathsf{k},\cdot)}(1^\kappa) = 1] - \Pr[\mathsf{A}^{\mathcal{O}_u(\cdot)}(1^\kappa) = 1]| \leq \mathsf{negl},$$

where $\mathsf{k} \leftarrow \mathsf{K}$ *and for* $x \in \mathsf{D}$*,* $\mathcal{O}_u$ *outputs a uniform* $y \leftarrow \mathsf{I}$ *whereas* $\mathcal{O}_{\mathsf{PRF}}(\mathsf{k}, \cdot)$ *outputs* $y = \mathsf{PRF}(\mathsf{k}, x)$*, and where* κ *is the security parameter.*

Public Key Encryption

Definition 2 (Public Key Encryption with Authenticated Data)**.** *A public encryption scheme with authenticated data is a triplet of algorithms* $(\mathsf{PKE.KG}, \mathsf{PKE.ENC}, \mathsf{PKE.DEC})$ *with the following syntax:*

- $\mathsf{PKE.KG}(1^\kappa)$*: On input* 1^κ *output a key pair* $(\mathsf{pk}, \mathsf{sk})$*.*
- $\mathsf{PKE.ENC}(\mathsf{pk}, \mathsf{m}, \mathsf{ad})$*: On input* $(\mathsf{pk}, \mathsf{m}, \mathsf{ad})$*, where* ad *is associated data* $\mathsf{PKE.ENC}$ *outputs a ciphertext* ct*.*
- $\mathsf{PKE.DEC}(\mathsf{sk}, \mathsf{ct}, \mathsf{ad})$*: On input* $(\mathsf{sk}, \mathsf{ct}, \mathsf{ad})$*,* $\mathsf{PKE.DEC}$ *outputs a message* m*.*

For correctness, we ask that for any message $\mathsf{m} \in \{0,1\}^*$ *and any associated data* ad*,*

$$\Pr_{(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{PKE.KG}(1^\kappa)}[\mathsf{PKE.DEC}(\mathsf{sk}, \mathsf{PKE.ENC}(\mathsf{pk}, \mathsf{m}, \mathsf{ad}), \mathsf{ad}) = \mathsf{m}] \geq 1 - \mathsf{negl}.$$

Definition 3 (IND-CCA-AD Security)**.** *We call an encryption scheme indistinguishable under chosen ciphertext attacks with authenticated data (IND-CCA-AD secure) if for any* ppt *algorithm* A*,*

$$\left|\Pr[\mathsf{A}^{\mathcal{O}_{\mathsf{PKE.DEC}}(\mathsf{sk},\cdot)}(\mathsf{pk}, \mathsf{ct}_0) = 1] - \Pr[\mathsf{A}^{\mathcal{O}_{\mathsf{PKE.DEC}}(\mathsf{sk},\cdot)}(\mathsf{pk}, \mathsf{ct}_1) = 1]\right| \leq \mathsf{negl},$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KG}(1^\kappa)$*,* $(\mathsf{m}_0, \mathsf{ad}_0, \mathsf{m}_1, \mathsf{ad}_1) \leftarrow \mathsf{A}(\mathsf{pk})$*,* $\forall i \in \{0,1\} : \mathsf{ct}_i \leftarrow \mathsf{PKE.ENC}(\mathsf{pk}, \mathsf{m}_i, \mathsf{ad}_i)$*, where* $\mathcal{O}_{\mathsf{PKE.DEC}}(\mathsf{sk}, \cdot)$ *responds to queries* m, ad *with* $\mathsf{PKE.DEC}(\mathsf{sk}, \mathsf{m}, \mathsf{ad})$ *and* A *does not query* $\mathsf{ct}_b, \mathsf{ad}_0$ *nor* $\mathsf{ct}_b, \mathsf{ad}_1$ *to* $\mathcal{O}_{\mathsf{PKE.DEC}}(\mathsf{sk}, \cdot)$*.*

Similar to the fact that CCA security implies non-malleability, the IND-CCA-AD notion implies that it is hard to manipulate the associated data of a ciphertext since otherwise, adversary A could use the decryption oracle to decrypt the challenge ciphertext together with manipulated associated data.

UC Security For a protocol Π between a set of parties $\mathcal{C} \cup \mathcal{H}$, we use $(\mathcal{C}, \mathcal{H})_\Pi$ to denote the joint output distribution. We omit Π when the protocol is clear from the context. Similarly, we use $(\mathsf{Sim}, \mathcal{F})$ to denote the output distribution of the joint output of algorithm Sim interacting with ideal functionality $\mathcal{F}$ and the outputs of $\mathcal{F}$ to the other parties interacting with $\mathcal{F}$.

For defining UC security, we follow the framework of [16]. For more details and an introduction to UC, we refer to [16].

Definition 4 (UC Security)**.** *A protocol* Π *UC realizes ideal functionality* $\mathcal{F}$ *if for any* ppt *adversary* A *corrupting the parties* $\mathcal{C}$*, there exists a simulator* Sim *such that for any* ppt *environment* D *and any polynomial size auxiliary input* z

$$|\Pr[\mathsf{D}(z, (\mathcal{C}, \mathcal{H})_\Pi) = 1] - \Pr[\mathsf{D}(z, (\mathsf{Sim}, \mathcal{F})) = 1]| = \mathsf{negl},$$

where all algorithms receive input 1^κ *and* $\mathcal{H}$ *is the set of honest parties.*

Replicated Secret Sharing

Definition 5 (Three Party Replicated Secret Sharing)**.** *A* three party replicated secret sharing *scheme consists of a input domain* D_{sh} *with an addition* $\oplus$ *and multiplication* $\odot$ *and the following* ppt *algorithms.*

$\mathsf{share}(x)$**:** *For input* $x \in \mathsf{D}_{\mathsf{sh}}$*, sample* $\mathsf{sh}_1, \mathsf{sh}_2, \mathsf{sh}_3 \leftarrow \mathsf{D}_{\mathsf{sh}}$ *under the constraint that* $x = \mathsf{sh}_1 \oplus \mathsf{sh}_2 \oplus \mathsf{sh}_3$*. Define shares* $\mathsf{SH}_1 := (\mathsf{sh}_1, \mathsf{sh}_2)$*,* $\mathsf{SH}_2 := (\mathsf{sh}_2, \mathsf{sh}_3)$*,* $\mathsf{SH}_3 := (\mathsf{sh}_3, \mathsf{sh}_1)$ *and output* $(\mathsf{SH}_1, \mathsf{SH}_2, \mathsf{SH}_3)$*.*

$\mathsf{share}(x, i, \mathsf{SH}_i)$**:** *For input* $x \in \mathsf{D}_{\mathsf{sh}}$ *and* $\mathsf{SH}_i = (\mathsf{sh}_i, \mathsf{sh}_{i+1 \bmod 3})$*, compute* $\mathsf{sh}_{i-1 \bmod 3} := x \ominus \mathsf{sh}_i \ominus \mathsf{sh}_{i+1 \bmod 3}$*. Output* $(\mathsf{SH}_1, \mathsf{SH}_2, \mathsf{SH}_3)$*. We sometimes abbreviate* $\mathsf{share}(x, i, \mathsf{SH}_i)$ *as* $\mathsf{share}(x, \mathsf{SH}_i)$*.*

$\mathsf{combine}(i, j, \mathsf{SH}_i, \mathsf{SH}_j)$**:** *If* $i \neq j$*, compute* $x := \mathsf{sh}_1 \oplus \mathsf{sh}_2 \oplus \mathsf{sh}_3$ *and output* x*. Otherwise, output* $\perp$*.*

4 Interoperable Private Attribution

4.1 Model

Parties We consider the following parties.

MKP**:** A match key provider sets up each user agent with a user specific match key.

UA**:** A user agent generates source and trigger events and sends them in encrypted form to a report collector. The events include a match key.

Examples of user agents are web browsers and smart phone applications.

RC**:** A report collector collects encrypted events, called reports, from user agents and stores them together with meta-information, which includes timestamp and campaign ID. Report collectors might send reports to other report collectors.

A RC can make a query to the MPC helper parties by submitting a list of encrypted events that are sorted by the timestamp.

Examples of report collectors are advertiser (for trigger events) and publisher (for source events).

MPC Helper Parties: After receiving a query from a RC, the helper parties process the query in MPC and send the outcome to RC. There are three helper parties which we denote with party 1, 2 and 3.

Network Topology We assume secure, i.e. confidential and authenticated, point-to-point channels between match key provider and user agents, user agents and report collectors, different report collectors, report collectors and MPC helper parties, and different MPC helper parties. We can establish such channels using HTTPS, TLS and PKI.

Security under Corruptions Security holds under the following corruption model. An adversary is allowed to statically corrupt any amount of match key providers, user agents and report collectors. He is allowed to statically corrupt at most one out of the three MPC helper parties. Corrupted parties might not follow the protocol description (malicious security, active security). We use $\mathcal{H}$ to denote the set of honest parties and $\mathcal{C}$ the set of corrupted parties. $\mathsf{UA} \in \mathcal{C}$ denotes a corrupted user agent, and likewise for MKP and RC. $\mathcal{H}$ and $\mathcal{C}$ may also contain the integers 1, 2 and 3, referring to the MPC helper parties. We use $i \in \mathcal{H}$ to refer to all honest MPC helper parties. We use $i \in \mathcal{C}$ accordingly.

We consider security under aborts. If any party aborts, the protocol execution stops. In the ideal world, we allow the simulator or the functionality to abort as well. Further, if any subroutine of an ideal functionality or protocol aborts, the ideal functionality and protocol abort as well. If an abort happens during a subsession (ssid), only the subsession is aborted, the main session (sid) continues.

4.2 IPA Related Definitions

Source and Trigger Events In IPA, there are source and trigger events. When processing the events in MPC, we use the same format such that when events are secret shared, trigger and source events cannot be distinguished. We define the event datatype as follows.

Definition 6 (Events)**.** *An* event evt *consists of the following variables.*

epid**:** *The epoch identifier specifies the epoch of an event. An event is excluded in queries under DP budget* $\mathsf{dpbgt}_{\mathsf{epid}'}$ *for epoch* epid' *iff* $\mathsf{epid} \notin \mathsf{dprange}(\mathsf{epid}')$*, where range* $\mathsf{dprange}$ *is defined for each epoch and determines from what passed epochs events can be used in cross-epoch queries.*

mk**:** *The match key which is unique for each user and used to group user events.*

ac**:** *The attribution constraint is an additional constraint for the attribution process. To be eligible for attribution two events must have the same* mk *and same* ac*.*

ts**:** *Timestamp of the event.*

tb**:** *The trigger bit determines whether the event is a trigger, i.e.* 1 *or source event, i.e.* 0*.*

bk**:** *Breakdown key of the event which determines the aggregation category of the event. We use* BK *to denote the set of all breakdown keys.*

tv**:** *The trigger value which is* 0 *for source events.*

The following variables are appended when processing events in IPA.

hb**:** *The helper bit determines whether the previous event (IPA events are processed in a sorted list) has the same match key and attribution constraint.*

sb: *The stop bit determines whether attribution process of an event has been finalized.*

cr: *The credit of a source event is the sum of trigger values that are attributed to this source event.*

minf: *Metadata information that is sent from the* UA *to a* RC *in a session where the* RC *also requests an encrypted* mk*. This information would be used by the* RC *to assign* tv *or* bk*.*

eid: *An event id which is used by the ideal functionality.*

In IPA, we process events in lists. We define a list of events together with list operators as follows.

Definition 7 (List of Events)**.** *A list of events* L *consists of events* evt*. Further, we use* $\mathsf{L.evt}$ *to access a specific event* $\mathsf{L.(evt}+t)$ *to access a following event that is* $t-1$ *events apart of* evt*.* $\mathsf{L.(evt}-t)$ *is defined accordingly. We use* $\mathsf{L}[i]$ *to access the element on position* i *in* L*.*

Definition 8 (Special symbols)**.** *Various protocols use values* $x \in \mathsf{D_{sh}} \cup \{\emptyset, \mathsf{flag}\}$*. The symbols* $\emptyset$ *and* flag *are applied when reports fail decryption and have inconsistent shares, respectively.*

4.3 The Ideal Functionality of IPA

We present the ideal functionality of IPA $\mathcal{F}_{\mathsf{IPA}}$ in Figure 1. In this functionality, a user agent can set up a match key. If no match key is set up, it is picked at random during the event generation. Reports of generated events are shared with a report collector. Additionally, report collectors are allowed to share events with other report collectors. At any time, a report collector is allowed to spend his privacy budget and submit a query to the IPA functionality on a subset of his reports. $\mathcal{F}_{\mathsf{IPA}}$ returns the differentially private aggregated result to the report collector.

Note that in UC Security proofs the ideal functionality carries out operations that are enforced by the protocol. But certain operations which are prescribed for correctness but not enforced (or needed for privacy) are not included in the ideal functionality. An example of this is presorting reports based on ts by RC before submitting a query.

5 The IPA Protocol

We present the IPA protocol in a more modular fashion, separated by initialization, match key generation, event generation, sharing events and query even though the different parts only together realize functionality $\mathcal{F}_{\mathsf{IPA}}$. We focus on presenting the case that includes a MKP as that is more complex than device generated matchkeys.

Functionality $\mathcal{F}_{\mathsf{IPA}}$:

Match Key Generation:

1. Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \mathsf{request}, \mathsf{MKP})$ from UA, send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{UA})$ to MKP.
2. Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \mathsf{mk})$ from MKP, send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{mk})$ to UA.

Event Generation:

- Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \mathsf{epid}, \mathsf{mk}, \mathsf{minf}, \mathsf{RC})$ from UA, generate eid. Send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{eid}, \mathsf{epid}, \mathsf{minf})$ to RC and store $(\mathsf{eid}, \mathsf{epid}, \mathsf{mk})$ in L_{RC}.

If $\mathsf{RC} \in \mathcal{C}$, it can also make the following query:

- Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \mathsf{mk}, \mathsf{epid})$ from RC, generate eid. Store $(\mathsf{eid}, \mathsf{epid}, \mathsf{mk})$ in L_{RC} and send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{eid})$ to RC.

Sharing Reports:

- Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \mathsf{eid}, \mathsf{ac}, \mathsf{ts}, \mathsf{tb}, \mathsf{bk}, \mathsf{tv}, \mathsf{RC}')$ from RC, look up $(\mathsf{eid}, \mathsf{epid}, \mathsf{mk}) \in \mathsf{L}_{\mathsf{RC}}$, generate eid' store $(\mathsf{eid}', \mathsf{epid}, \mathsf{mk}, \mathsf{ac}, \mathsf{ts}, \mathsf{tb}, \mathsf{bk}, \mathsf{tv})$ in $\mathsf{LS}_{\mathsf{RC}'}$ and send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{eid}', \mathsf{epid}, \mathsf{ts})$ to RC'. (It might be that $\mathsf{RC} = \mathsf{RC}'$).

Query: (no concurrent queries by same RC*)*

1. Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \mathsf{L}_{\mathsf{eid}}, \mathsf{epid}, \mathsf{dpsp}, \mathsf{dpcap})$ from RC, send $\mathsf{L}_{\mathsf{eid}}, \mathsf{dpsp}, \mathsf{dpcap}$ to parties 1, 2 and 3. Abort if $\mathsf{dpbgt}_{\mathsf{RC},\mathsf{epid}} < \mathsf{dpsp}$. Set $\mathsf{dpbgt}_{\mathsf{RC},\mathsf{epid}} := \mathsf{dpbgt}_{\mathsf{RC},\mathsf{epid}} - \mathsf{dpsp}$. For $\mathsf{eid} \in \mathsf{L}_{\mathsf{eid}}$, if $(\mathsf{eid}, \mathsf{epid}, \mathsf{mk}) \in \mathsf{L}_{\mathsf{RC}}$, request $(\mathsf{ac}, \mathsf{ts}, \mathsf{tb}, \mathsf{bk}, \mathsf{tv})$ from RC and otherwise, look up $(\mathsf{eid}, \mathsf{epid}, \mathsf{mk}, \mathsf{ac}, \mathsf{ts}, \mathsf{tb}, \mathsf{bk}, \mathsf{tv}) \in \mathsf{LS}_{\mathsf{RC}}$. Add $(\mathsf{eid}, \mathsf{epid}, \mathsf{mk}, \mathsf{ac}, \mathsf{ts}, \mathsf{tb}, \mathsf{bk}, \mathsf{tv})$ to L. Remove $\mathsf{evt} \in \mathsf{L}$ from L with $\mathsf{evt.epid} \notin \mathsf{dprange}(\mathsf{epid})$ or $\mathsf{evt.mk} = \emptyset$ (send eid of removed elements to adversary).
2. Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \mathsf{sssid}, \mathsf{eid})$ from $i \in \mathcal{C}$, delete element with eid from L. ($i \in \mathcal{C}$ can issue multiple queries).
3. Stable sort L according to $(\mathsf{evt.mk}, \mathsf{evt.ac})$.
4. Perform a last touch attribution on L: Append $\mathsf{evt.cr}$ to each event and initialize it with 0. For every event, compute $(\mathsf{evt}+i).\mathsf{cr} += \mathsf{evt.tb} \cdot \mathsf{evt.tv}$ where $(\mathsf{evt}+i)$ is the closest preceding source event, i.e. $(\mathsf{evt}+i).\mathsf{tb} = 0$, for which $\mathsf{evt.mk} = (\mathsf{evt}+i).\mathsf{mk}$ and $\mathsf{evt.ac} = (\mathsf{evt}+i).\mathsf{ac}$. If no such event exists, ignore this event.
5. Perform DP capping: Set $\mathsf{cr} := 0$ for all trigger events. $\forall i \in [|\mathsf{L}|]$, set $(\mathsf{evt}+i).crt = \min((\mathsf{evt}+i).crt, \max(0, \mathsf{dpcap} - \sum_{j \in [i]}((\mathsf{evt}+i).\mathsf{mk} = (\mathsf{evt}+j).\mathsf{mk}) \cdot (\mathsf{evt}+j).\mathsf{cr}))$
6. Perform an aggregation on L: For each $\mathsf{bk} \in \mathsf{BK}$, compute $\mathsf{AG}_{\mathsf{bk}} := \sum_{\mathsf{evt} \in \mathsf{L}}(\mathsf{bk} = \mathsf{evt.bk}) \cdot \mathsf{evt.cr}$.
7. Sample and add DP-noise: For each $\mathsf{bk} \in \mathsf{BK}$, sample $e_{\mathsf{bk}} \leftarrow \mathcal{D}_{\mathsf{dpsp},\mathsf{dpcap}}$ and compute $\mathsf{DPAG}_{\mathsf{bk}} := \mathsf{AG}_{\mathsf{bk}} + e_{\mathsf{bk}}$.
8. Send $(\mathsf{sid}, \mathsf{ssid}, \{\mathsf{bk}, \mathsf{DPAG}_{\mathsf{bk}}\}_{\mathsf{bk} \in \mathsf{BK}})$ to RC.

Figure 1: The ideal functionality $\mathcal{F}_{\mathsf{IPA}}$. Match key generation, event generation, sharing events and query can be queried concurrently (as indicated by giving them a subsession id ssid).

5.1 Protocol Overview

In Figure 2, we give a brief overview of the IPA protocol and its different sub-protocols.

Protocol for IPA (sub-protocols):

- Initialization
- Match Key Generation
- Event Generation
- Sharing Reports
- Query

Figure 2: The overview of the protocol and its sub-protocols realizing IPA.

Initialization During the initialization, we specify the privacy budgets for all the report collectors and epochs. Different from $\mathcal{F}_{\mathsf{IPA}}$, the real protocol performs additional steps to set up the other protocol procedures. This includes initializing secret and public keys for an IND-CCA-AD secure PKE and distributing them. We show the details in Figure 3. In an implementation, the user agent

Protocol for Initialization:

1. For $i \in [3]$, party i generates $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{PKE.KG}(1^\kappa)$.
2. For $i \in [3]$, party i sends pk_i to all UA and RC.

Figure 3: The protocol realizing the initialization.

vendors will distributed the keys to the user agents.

Match Key Generation The protocol for the match key generation is straightforward since a match key provider can simply send the match key to the user agent over the point to point channel. We describe the protocol formally in

Protocol for Match Key Generation:

1. UA sends $(\mathsf{sid}, \mathsf{ssid}, \mathsf{request})$ to MKP.
2. MKP sends with $(\mathsf{sid}, \mathsf{ssid}, \mathsf{mk})$ to UA.

Figure 4: The protocol realizing the match key generation.

Figure 4.

Protocol for Event Generation:

1. UA samples $(\mathsf{SH}_1, \mathsf{SH}_2, \mathsf{SH}_3) \leftarrow \mathsf{share}(\mathsf{mk})$
2. UA defines $\mathsf{ad} := \mathsf{epid}$ and computes $\mathsf{ct}_i \leftarrow \mathsf{PKE.ENC}(\mathsf{pk}_i, \mathsf{SH}_i, \mathsf{ad})$ $\forall i \in [3]$.
3. UA sends $\mathsf{epid}, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3$ and minf to RC.
4. RC samples $(\mathsf{SHRC}_1, \mathsf{SHRC}_2, \mathsf{SHRC}_3) \leftarrow \mathsf{share}((\mathsf{ac}, \mathsf{ts}, \mathsf{tb}, \mathsf{bk}, \mathsf{tv}))$ and computes $\mathsf{ctRC}_i \leftarrow \mathsf{PKE.ENC}(\mathsf{pk}_i, \mathsf{SHRC}_i, \emptyset)$ $\forall i \in [3]$.
5. RC stores report $\mathsf{rp} := (\mathsf{epid}, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{ts}, \mathsf{ctRC}_1, \mathsf{ctRC}_2, \mathsf{ctRC}_3)$.

Figure 5: The protocol realizing the event generation.

Event Generation We describe the protocol for the event generation in Figure 5. For the event generation, we use an IND-CCA-AD secure PKE scheme together with a replicated secret sharing scheme. We do not specify the domain D_{sh} in this section and emphasize that it could be implemented using different domains, depending on the data type of the variable and downstream processing of the events.

Sharing Reports A report collector can share his reports with any other report collector by simply sending him the reports over the point to point channel. Figure 6 shows the protocol.

Protocol for Sharing Reports:

1. RC sends $(\mathsf{sid}, \mathsf{ssid}, \mathsf{rp})$ to RC'.

Figure 6: The protocol realizing sharing reports between report collectors.

Query We present the protocol for IPA queries in a modular fashion by abstracting the individual steps using ideal functionalities. This allows to replace them individually with different realizations without impacting the security of IPA. We present the protocol for IPA queries in Figure 7. The protocol uses the following ideal functionalities:

$\mathcal{F}_{\mathsf{VRFY}}$**:** Verifies that the shares are well-formed.

$\mathcal{F}_{\mathsf{Sort}}$**:** Performs a stable sort according to $(\mathsf{evt.mk}, \mathsf{evt.ac})$ on the secret shared list.

$\mathcal{F}_{\mathsf{LTA}}$**:** Performs a last touch attribution on the secret shared list.

$\mathcal{F}_{\mathsf{CAP}}$**:** Performs a DP capping on the secret shared list.

$\mathcal{F}_{\mathsf{AGR}}$**:** Performs an aggregation according to $\mathsf{evt.bk}$ on the secret shared list and outputs a secret shared aggregate for each breakdown key.

$\mathcal{F}_{\mathsf{DPNoise}}$**:** Generates DP noise and adds it to each secret shared aggregate.

Protocol for Query:

1. RC sorts $\mathsf{L} = \{\mathsf{rp}_j\}_{j\in[m]}$ for $m \in \mathbb{N}$ according to ts and sends $(\mathsf{dpsp}, \mathsf{dpcap}, \mathsf{epid}, \{\mathsf{ad}_j, \mathsf{ct}_{j,i}, \mathsf{ctRC}_{j,i}\}_{j\in[m]})$ to helper party i for each $i \in [3]$.
2. The parties $i \in [3]$ send $\mathsf{dpsp}, \mathsf{dpcap}, \mathsf{epid}$ to each other and abort if it is inconsistent or $\mathsf{dpbgt}_{\mathsf{RC},\mathsf{epid}} < \mathsf{dpsp}$.
3. The parties $i \in [3]$ remove all shares for which $\mathsf{epid}_j \notin \mathsf{dprange}(\mathsf{epid})$ (by informing the other parties). They compute $\mathsf{SH}_{j,i} := \mathsf{PKE.DEC}(\mathsf{sk}_i, \mathsf{ct}_{j,i}, \mathsf{ad}_j)$, $\mathsf{SHRC}_{j,i} := \mathsf{PKE.DEC}(\mathsf{sk}_i, \mathsf{ctRC}_{j,i}, \emptyset)$ for all remaining j. If a decryption for $j \in [m]$ fails, the party informs the other parties and all parties remove the shares for j. They submit $\mathsf{SH}_{j,i}, \mathsf{SHRC}_{j,i}$ to $\mathcal{F}_{\mathsf{VRFY}}$ for all remaining j. If $\mathcal{F}_{\mathsf{VRFY}}$ outputs 0, they remove the shares for j. If the lists have inconsistent sizes, the parties call $\mathcal{F}_{\mathsf{VRFY}}$ for empty shares. We use $n \in \mathbb{N}$ to denote the remaining amount of reports. They define $\mathsf{L}_i := \{\mathsf{SH}_{j,i}, \mathsf{SHRC}_{j,i}\}_{j\in[n]}$.
4. Party i for $i \in \mathcal{H}$ submits L_i to $\mathcal{F}_{\mathsf{Sort}}$ and replaces L_i with the response of $\mathcal{F}_{\mathsf{Sort}}$.
5. Party i for $i \in \mathcal{H}$ submits L_i to $\mathcal{F}_{\mathsf{LTA}}$ and replaces L_i with the response of $\mathcal{F}_{\mathsf{LTA}}$.
6. Party i for $i \in \mathcal{H}$ submits L_i to $\mathcal{F}_{\mathsf{CAP}}$ and replaces L_i with the response of $\mathcal{F}_{\mathsf{CAP}}$.
7. Party i for $i \in \mathcal{H}$ submits L_i to $\mathcal{F}_{\mathsf{AGR}}$ and replaces L_i with the response of $\mathcal{F}_{\mathsf{AGR}}$.
8. Party i for $i \in \mathcal{H}$ submits $\mathsf{L}_i, \mathsf{dpsp}, \mathsf{dpcap}$ to $\mathcal{F}_{\mathsf{DPNoise}}$ and replaces L_i with the response of $\mathcal{F}_{\mathsf{DPNoise}}$.
9. Party i submits $\{\mathsf{bk}, \mathsf{e}_{\mathsf{bk},i}\}_{\mathsf{bk}\in\mathsf{BK}}$ to RC.
10. RC recombines $\mathsf{bk}, \mathsf{e}_{\mathsf{bk},i}$ to $\mathsf{DPAG}_{\mathsf{bk}}$. He aborts if reconstruction fails (inconsistent shares). Otherwise, he outputs $\{\mathsf{DPAG}_{\mathsf{bk}}\}_{\mathsf{bk}\in\mathsf{BK}}$.

Figure 7: The protocol realizing the query using ideal functionalities $\mathcal{F}_{\mathsf{VRFY}}$, $\mathcal{F}_{\mathsf{Sort}}$, $\mathcal{F}_{\mathsf{LTA}}$, $\mathcal{F}_{\mathsf{CAP}}$, $\mathcal{F}_{\mathsf{AGR}}$ and $\mathcal{F}_{\mathsf{DPNoise}}$.

We emphasize that Step 3 of the query step does not abort the protocol but deletes specific events in order to provide robustness against malicious user agents. Aborting the protocol because of a single ill-formed event would allow denial of service attacks and render the IPA protocol as useless in practice. Step 3 is not intended to provide security against a malicious helper party who would not be able to charge a report collector for its services in case of a protocol abort.

5.2 Ideal Functionalities during Query

Verify We define verification functionality $\mathcal{F}_{\mathsf{VRFY}}$ in Figure 8. We realize the

Functionality $\mathcal{F}_{\mathsf{VRFY}}$:

1. Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \mathsf{SH}_i)$ from party $i \in \mathcal{H}$, define $(\mathsf{sh}_i, , \mathsf{sh}'_{i+1 \bmod 3}) := \mathsf{SH}_i$. Send sh'_j to $j \in \mathcal{C}$, receive $\mathsf{sh}'_{j+1 \bmod 3}$ from $j \in \mathcal{C}$. If for any $i \in [3]$, $\mathsf{sh}_i = \emptyset$, immediately abort.
2. Send $(\mathsf{sh}_i = \mathsf{sh}'_i)$, $(\mathsf{sh}_k = \mathsf{sh}'_k)$ for $i, k \in \mathcal{H}$ to Party $j \in \mathcal{C}$. Receive $b \in \{0, 1\}$ from Party $j \in \mathcal{C}$. If there is no $j \in \mathcal{C}$, define $b := (\mathsf{sh}_j = \mathsf{sh}'_j)$. Send $b \wedge (\mathsf{sh}_k = \mathsf{sh}'_k) \wedge (\mathsf{sh}_i = \mathsf{sh}'_i)$ to Party i for $i \in \mathcal{H}$.

Figure 8: The ideal functionality $\mathcal{F}_{\mathsf{VRFY}}$.

share verification functionality $\mathcal{F}_{\mathsf{VRFY}}$ with protocol in Figure 9.

Protocol realizing $\mathcal{F}_{\mathsf{VRFY}}$:

1. For all $i \in [3]$, Party i define $(\mathsf{sh}_i, \mathsf{sh}_{i+1 \bmod 3}) := \mathsf{SH}_i$ and sends $\mathsf{sh}_{i+1 \bmod 3}$ to Party $i + 1 \bmod 3$. If for any $i \in [3]$, $\mathsf{sh}_i = \emptyset$, immediately abort.
2. For all $i \in [3]$, Party i receives sh'_i and checks whether $\mathsf{sh}_i = \mathsf{sh}'_i$. If this is the case, it sends 1 to both other parties, otherwise 0.
3. Party i outputs 0 if any sent or received message is 0, otherwise output 1. Further it shares the message received from Party $i - 1 \bmod 3$ with Party $i + 1 \bmod 3$. If the messages are inconsistent for any Party, the Party aborts.

Figure 9: The protocol realizing ideal functionality $\mathcal{F}_{\mathsf{VRFY}}$.

Lemma 1. *The protocol in Figure 9 realizes ideal functionality $\mathcal{F}_{\mathsf{VRFY}}$ when at most one helper party is corrupted.*

Proof. We construct the simulator Sim as follows. Sim receives sh'_j and forwards it to the adversary who responds with $\mathsf{sh}'_{j+1 \bmod 3}$. Sim sends $\mathsf{sh}'_{j+1 \bmod 3}$ to $\mathcal{F}_{\mathsf{VRFY}}$ which responds with b_i, b_k for $i, k \in \mathcal{H}$. Sim forwards both bits to the adversary who responds with b'_i for $i \in \mathcal{H}$. If $b'_i \neq b'_k$ for $i, k \in \mathcal{H}$ with $i \neq k$, Sim sends abort to $\mathcal{F}_{\mathsf{VRFY}}$. Sim forwards b'_i for $i \in [3] \setminus \{j\}$ to $\mathcal{F}_{\mathsf{VRFY}}$. Sim outputs the adversaries output.

The messages that Sim sends to the adversary are identical to the messages that the adversary receives during the real protocol. Further, if Sim does not abort, $\mathcal{F}_{\mathsf{VRFY}}$ produces the same outputs than the honest parties during the real protocol execution.

We need to show that if Sim aborts, at least one of the real parties in $\mathcal{H}$ would also abort. Sim aborts if the adversary sends inconsistent messages to the honest parties. In the real protocol, at least one of the honest parties shares this message with the other honest party which would then detect the inconsistency and abort. □

Secure Sorting We define the ideal functionality $\mathcal{F}_{\mathsf{Sort}}$ in Figure 10. [12] uses

Functionality $\mathcal{F}_{\mathsf{Sort}}$:

1. Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \mathsf{L}_i)$ from party $i \in \mathcal{H}$, reconstruct L.
2. Stable sort L according to $(\mathsf{evt.mk}, \mathsf{evt.ac})$.
3. Receive L_j for $j \in \mathcal{C}$ from Party j. Generate $(\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3) \leftarrow \mathsf{share}(\mathsf{L}, j, \mathsf{L}_j)$.
4. Send L_i to Party $i \in \mathcal{H}$.

Figure 10: The ideal functionality $\mathcal{F}_{\mathsf{Sort}}$.

a Radix sorting based approach for secure sorting. The advantage of this sort is that it does not use secure comparison operations which are usually performed using Boolean circuits. To upgrade Boolean circuits to malicious security by using MACs is very expensive since every AND gate necessary for semi-honest security results in $1+\kappa$ many AND gates for malicious security. One of the best performing comparison based sorting is Quicksort. Unfortunately, Quicksort reveals the amount of identical events which would allow to track the total amount of events of a user which would violate our security goals. This can be fixed by appending a counter to each match key. However, the counter size would be $\log(|\mathsf{L}|)$, which increases the costs for comparisons by another $\log(|\mathsf{L}|)$ factor. For these reasons, a Radix sorting based approach seems to be better suited for this work. Further, at the current state of art, it is more dollar cost efficient in terms of network and computation costs.

Last Touch Attribution We define the ideal functionality $\mathcal{F}_{\mathsf{LTA}}$ in Figure 11. We show how to instantiate this functionality in Section 9.

DP Capping We define the ideal functionality $\mathcal{F}_{\mathsf{CAP}}$ in Figure 12. We show how to instantiate this functionality in Section 9.

Aggregation We define the ideal functionality $\mathcal{F}_{\mathsf{AGR}}$ in Figure 13. We show how to instantiate this functionality in Section 9.

Functionality $\mathcal{F}_{\mathsf{LTA}}$:

1. Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \mathsf{L}_i)$ from party $i \in \mathcal{H}$, reconstruct L. Send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{L}_j)$ to $j \in \mathcal{C}$.
2. Perform a last touch attribution on L: Append $\mathsf{evt.cr}$ to each event and initialize it with $\mathsf{evt.tv}$. For every event $\mathsf{evt} \in \mathsf{L}$ define $I \in [|\mathsf{L}|]$ s.t. $(\mathsf{evt} - I)$ is the closest preceding source event for which $\mathsf{evt.mk} = (\mathsf{evt} - I).\mathsf{mk}$ and $\mathsf{evt.ac} = (\mathsf{evt} - I).\mathsf{ac}$. For every $i \in [I]$, compute $(\mathsf{evt} - i).\mathsf{cr} = (\mathsf{evt} - i).\mathsf{cr} + \mathsf{evt.tv}$. If no such event exists, continue with next event in L.
3. For every $\mathsf{evt} \in \mathsf{L}$, receive $\mathsf{evt}_j.\mathsf{SH}_{\mathsf{cr},j}$ for $j \in \mathcal{C}$ from Party j. Generate $\{\mathsf{evt}_i.\mathsf{SH}_{\mathsf{cr},i}\}_{i\in[3]} \leftarrow \mathsf{share}(\mathsf{evt.cr}, \mathsf{evt}_j.\mathsf{SH}_{\mathsf{cr},j})$. Append $\mathsf{evt}_i.\mathsf{SH}_{\mathsf{cr},i}$ to list L_i.
4. Send L_i to Party $i \in \mathcal{H}$.

Figure 11: The ideal functionality $\mathcal{F}_{\mathsf{LTA}}$.

Functionality $\mathcal{F}_{\mathsf{CAP}}$:

1. Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \mathsf{L}_i)$ from party $i \in \mathcal{H}$, reconstruct L. Send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{L}_j)$ to $j \in \mathcal{C}$.
2. Set $\mathsf{cr} := 0$ for all events with $\mathsf{tb} = 1$.
3. Perform DP capping: $\forall r \in \{|\mathsf{L}|, \ldots, 1\}$, i.e. iterate in reverse order, define the currently used budget for event $\mathsf{L}[r]$ as $c_r := \sum_{j\in\{r,\ldots|\mathsf{L}|\}}(\mathsf{L}[r].\mathsf{mk} = \mathsf{L}[j].\mathsf{mk}) \cdot \mathsf{L}[j].\mathsf{cr}$ Set $\mathsf{L}[r].\mathsf{cr} := \min\big(\mathsf{L}[r].\mathsf{cr}, \max\big(0, \mathsf{dpcap} - c_r\big)\big)$
4. For every $\mathsf{evt} \in \mathsf{L}$, receive $\mathsf{evt}_j.\mathsf{SH}_{\mathsf{cr},j}$ for $j \in \mathcal{C}$ from Party j. Generate $\{\mathsf{evt}_i.\mathsf{SH}_{\mathsf{cr},i}\}_{i\in[3]} \leftarrow \mathsf{share}(\mathsf{evt.cr}, \mathsf{evt}_j.\mathsf{SH}_{\mathsf{cr},j})$. Update $\mathsf{evt}_i.\mathsf{SH}_{\mathsf{cr},i}$ in list L_i.
5. Send L_i to Party $i \in \mathcal{H}$.

Figure 12: The ideal functionality $\mathcal{F}_{\mathsf{CAP}}$.

Functionality $\mathcal{F}_{\mathsf{AGR}}$:

1. Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \mathsf{L}_i)$ from party $i \in \mathcal{H}$, reconstruct L. Send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{L}_j)$ to $j \in \mathcal{C}$.
2. Perform an aggregation on L: For each $\mathsf{bk} \in \mathsf{BK}$, compute $\mathsf{AG}_{\mathsf{bk}} := \sum_{\mathsf{evt}\in\mathsf{L}}(\mathsf{bk} = \mathsf{evt.bk}) \cdot \mathsf{evt.cr}$.
3. For each $\mathsf{bk} \in \mathsf{BK}$, Receive $\mathsf{SH}_{\mathsf{bk},j}$ for $j \in \mathcal{C}$ from Party j. Generate $\{\mathsf{SH}_{\mathsf{bk},i}\}_{i\in[3]} \leftarrow \mathsf{share}(\mathsf{AG}_{\mathsf{bk}}, \mathsf{SH}_{\mathsf{bk},j})$.
4. Send $\{\mathsf{SH}_{\mathsf{bk},i}\}_{\mathsf{bk}\in\mathsf{BK}}$ to Party $i \in \mathcal{H}$.

Figure 13: The ideal functionality $\mathcal{F}_{\mathsf{AGR}}$.

Functionality $\mathcal{F}_{\mathsf{DPNoise}}$:

1. Upon receiving $(\mathsf{sid}, \mathsf{ssid}, \{\mathsf{SH}_{\mathsf{bk},i}\}_{\mathsf{bk}\in\mathsf{BK}})$ from party $i \in \mathcal{H}$, reconstruct $\mathsf{AG}_{\mathsf{bk}}$. Send $(\mathsf{sid}, \mathsf{ssid}, \{\mathsf{SH}_{\mathsf{bk},j}\}_{\mathsf{bk}\in\mathsf{BK}})$ to $j \in \mathcal{C}$.
2. Sample and add DP-noise: For each $\mathsf{bk} \in \mathsf{BK}$, sample $e_{\mathsf{bk}} \leftarrow \mathcal{D}_{\mathsf{dpsp},\mathsf{dpcap}}$ and compute $\mathsf{DPAG}_{\mathsf{bk}} := \mathsf{AG}_{\mathsf{bk}} + e_{\mathsf{bk}}$.
3. Receive $\{\widehat{\mathsf{SH}}_{\mathsf{bk},j}\}_{\mathsf{bk}\in\mathsf{BK}}$ for $j \in \mathcal{C}$ from Party j. Generate for each $\mathsf{bk} \in \mathsf{BK}$: $\{\widehat{\mathsf{SH}}_{\mathsf{bk},i}\}_{i\in[3]} \leftarrow \mathsf{share}(\mathsf{DPAG}_{\mathsf{bk}}, \widehat{\mathsf{SH}}_{\mathsf{bk},j})$.
4. Send $\{\widehat{\mathsf{SH}}_{\mathsf{bk},i}\}_{\mathsf{bk}\in\mathsf{BK}}$ to Party $i \in \mathcal{H}$.

Figure 14: The ideal functionality $\mathcal{F}_{\mathsf{DPNoise}}$.

DP Noise We define the ideal functionality $\mathcal{F}_{\mathsf{DPNoise}}$ in Figure 14. To generating the DP noise securely, we can use a Box-Muller transform based approach as described in [26]. Since we use a global DP setting, the noise generation process is independent of the amount of events, i.e. $|\mathsf{L}|$. Therefore, this process is not performance sensitive and has minimal impact on the overall dollar cost of the IPA protocol.

6 Security of the IPA protocol

Theorem 1. *Let* PKE *be an IND-CCA-AD secure PKE. Then, the IPA protocol defined in Figure 3, 4, 5, 6 and 7 securely realizes the ideal IPA functionality defined in Figure 1 under at most one corrupted helper party and any amount of corrupted match key provider, user agents and report collectors in the* $\mathcal{F}_{\mathsf{VRFY}}$, $\mathcal{F}_{\mathsf{Sort}}$, $\mathcal{F}_{\mathsf{LTA}}$, $\mathcal{F}_{\mathsf{CAP}}$, $\mathcal{F}_{\mathsf{AGR}}$, $\mathcal{F}_{\mathsf{DPNoise}}$ *hybrid model.*

Proof. We construct the simulator Sim as follows.

Initialize: For each $i \in \mathcal{H}$, Sim generates $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{PKE.KG}(1^\kappa)$. Further it receives pk_i for $i \in \mathcal{C}$ from the adversary. Sim sends pk_1, pk_2 and pk_3 to all user agents UA and report collectors RC.

Match Key Generation: Sim simply forwards any message from or to a corrupted UA or corrupted MKP.

Event Generation: If $\mathsf{RC}, \mathsf{UA} \in \mathcal{H}$, do nothing. Otherwise, do the following. If $\mathsf{UA} \in \mathcal{C}$, receive $\mathsf{rp} := (\mathsf{epid}, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{minf})$ from UA. Use $\mathsf{sk}_i, \mathsf{sk}_j$ for $i, j \in \mathcal{H}$ with $i \neq j$, decrypt $\mathsf{ct}_i, \mathsf{ct}_j$ to obtain $\mathsf{SH}_i, \mathsf{SH}_j$ and use them to reconstruct mk (for failed decryption $\mathsf{mk} := \emptyset$, for inconsistent shares $\mathsf{mk} := \mathsf{flag}$). Submit $(\mathsf{sid}, \mathsf{ssid}, \mathsf{epid}, \mathsf{mk}, \mathsf{minf})$ to $\mathcal{F}_{\mathsf{IPA}}$ and the report to RC. If $\mathsf{UA} \in \mathcal{H}$ (and therefore $\mathsf{RC} \in \mathcal{C}$), receive $(\mathsf{sid}, \mathsf{ssid}, \mathsf{epid}, \mathsf{minf}, \mathsf{RC})$ from $\mathcal{F}_{\mathsf{IPA}}$. generate $(\mathsf{SH}_1, \mathsf{SH}_2, \mathsf{SH}_3) \leftarrow \mathsf{share}(0)$, compute $\mathsf{ct}_i \leftarrow \mathsf{PKE.ENC}(\mathsf{pk}_i, \mathsf{SH}_i, \mathsf{epid})$. If $\mathsf{RC} \in \mathcal{C}$, send $(\mathsf{epid}, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{minf})$ to RC. Store $(\mathsf{eid}, \mathsf{rp}, \mathsf{mk}, \{\mathsf{sh}_j, \mathsf{sh}_{j+1 \bmod 3}\}_{j\in\mathcal{C}})$.

Sharing Reports: We treat this case identically to the event generation where RC takes the role of UA and RC′ of RC with the following difference. Look

up report that matches $\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3$ among stored reports to get eid. If $\mathsf{RC} \in \mathcal{C}$ use sk_i, sk_j to decrypt ctRC_i, ctRC_j to obtain $\mathsf{SHRC}_i, \mathsf{SHRC}_j$, reconstruct and submit $(\mathsf{sid}, \mathsf{ssid}, \mathsf{eid}, \mathsf{ac}, \mathsf{ts}, \mathsf{tb}, \mathsf{bk}, \mathsf{tv})$ to $\mathcal{F}_{\mathsf{IPA}}$ (for failed decryption $\mathsf{mk} := \emptyset$, for inconsistent shares $\mathsf{mk} := \mathsf{flag}$). If $\mathsf{RC} \in \mathcal{H}$, generate $(\mathsf{SHRC}_1, \mathsf{SHRC}_2, \mathsf{SHRC}_3) \leftarrow \mathsf{share}(0)$ and encrypt them to ctRC_1, ctRC_2 and ctRC_3. Store $(\mathsf{eid}', \mathsf{mk}, \mathsf{rp}, \mathsf{rp}')$, where rp is the initial report sent by RC and rp' is the looked up report.

Query: If $\mathsf{RC} \in \mathcal{H}$ and there is a $j \in \mathcal{C}$, receive $\mathsf{L}_{\mathsf{eid}}, \mathsf{dpsp}, \mathsf{dpcap}$ from $\mathcal{F}_{\mathsf{IPA}}$. For every $\mathsf{eid} \in \mathsf{L}_{\mathsf{eid}}$, look up the matching report among the stored reports. If there is a match, use the associated report rp (eid is also matched against (eid,eid') of shared reports). Otherwise (report was generated by $\mathsf{UA} \in \mathcal{H}$), Sim generates an event by generating $(\mathsf{SH}_1, \mathsf{SH}_2, \mathsf{SH}_3) \leftarrow \mathsf{share}(0)$, $(\mathsf{SHRC}_1, \mathsf{SHRC}_2, \mathsf{SHRC}_3) \leftarrow \mathsf{share}(0)$, encrypting them to $\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3$ and $\mathsf{ctRC}_1, \mathsf{ctRC}_2, \mathsf{ctRC}_3$. Sim sends $(\mathsf{epid}, \mathsf{ct}_j, \mathsf{ctRC}_j)$ for all reports together with $\mathsf{dpsp}, \mathsf{dpcap}$ to Party j.

If $\mathsf{RC} \in \mathcal{C}$, Sim receives the reports for each helper party from RC and merges the reports by using $\mathsf{epid}, \mathsf{dpsp}, \mathsf{dpcap}$ of one of the honest parties to $(\mathsf{dpsp}, \mathsf{dpcap}, \mathsf{epid}, \{\mathsf{ad}_j, \mathsf{ct}_{j,i}, \mathsf{ctRC}_{j,i}\}_{j \in [m]})$. Sim compares $(\mathsf{ct}_{j,1}, \mathsf{ct}_{j,2}, \mathsf{ct}_{j,3})$ with other ciphertexts of stored reports to obtain a corresponding eid which he then adds to $\mathsf{L}_{\mathsf{eid}}$. If there is no matching ciphertext, Sim uses $\mathsf{sk}_i, \mathsf{sk}_k$ for $i, k \in \mathcal{H}$ to decrypt $\mathsf{ct}_{j,i}, \mathsf{ct}_{j,k}$ and the shares to reconstruct mk (for failed decryption set $\mathsf{mk} := \emptyset$, for inconsistent shares set $\mathsf{mk} := \mathsf{flag}$). He then issues an Event Generation query for corrupted RC, submitting $(\mathsf{sid}, \mathsf{ssid}, \mathsf{mk}, \mathsf{epid}_j)$ to $\mathcal{F}_{\mathsf{IPA}}$. $\mathcal{F}_{\mathsf{IPA}}$ responds with eid_j which Sim adds to $\mathsf{L}_{\mathsf{eid}}$. After adding one eid to $\mathsf{L}_{\mathsf{eid}}$ for each report, Sim issues a query to $\mathcal{F}_{\mathsf{IPA}}$. $\mathcal{F}_{\mathsf{IPA}}$ might request $(\mathsf{ac}, \mathsf{ts}, \mathsf{tb}, \mathsf{bk}, \mathsf{tv})$ which Sim obtains by using $\mathsf{sk}_i, \mathsf{sk}_k$ for $i, k \in \mathcal{H}$ to decrypt $\mathsf{ctRC}_{j,i}, \mathsf{ctRC}_{j,k}$ and using the shares for reconstructing $(\mathsf{ac}, \mathsf{ts}, \mathsf{tb}, \mathsf{bk}, \mathsf{tv})$ (for reports that fail decryption we set $\mathsf{mk} := \emptyset$, for inconsistent shares $\mathsf{mk} := \mathsf{flag}$).

Sim sends $\mathsf{dpsp}, \mathsf{dpcap}, \mathsf{epid}$, which he either gets from $\mathcal{F}_{\mathsf{IPA}}$ or $\mathsf{RC} \in \mathcal{C}$, for both honest parties to $j \in \mathcal{C}$. He aborts if $\mathsf{dpsp}, \mathsf{dpcap}, \mathsf{epid}$ are inconsistent or if $j \in \mathcal{C}$ sends inconsistent values. $\mathcal{F}_{\mathsf{IPA}}$ sends all eid of reports that need to be removed. Inform Party j that those reports need to be removed (includes any report with $\mathsf{mk} = \emptyset$). Party $j \in \mathcal{C}$ might request to remove reports as well. For each of those reports, issue a delete element query to $\mathcal{F}_{\mathsf{IPA}}$. If Party j only requests one of the other helper parties to remove the report, Sim sets $\mathsf{mk} := \mathsf{flag}$ for that report (but does not issue a delete query to $\mathcal{F}_{\mathsf{IPA}}$ for that report).

For all remaining events, Sim forwards the messages and outputs between Party $j \in \mathcal{C}$ and $\mathcal{F}_{\mathsf{VRFY}}$. Sim also submits the shares of the honest parties, which are shares of 0. To generate the share of 0, Sim looks up $\mathsf{SH}_j = (\mathsf{sh}_j, \mathsf{sh}_{j+1 \bmod 3})$ for $j \in \mathcal{C}$ among the stored reports. For reports with $\mathsf{mk} = \mathsf{flag}$, send the inconsistent shares which will cause them to be removed. Sim receives the output for $i \in \mathcal{H}$ from $\mathcal{F}_{\mathsf{VRFY}}$. If it is a 0, Sim

issues a delete query to $\mathcal{F}_{\mathsf{IPA}}$ to delete the report for that event from L.

Sim simulates the steps involving $\mathcal{F}_{\mathsf{Sort}}$, $\mathcal{F}_{\mathsf{LTA}}$, $\mathcal{F}_{\mathsf{CAP}}$, $\mathcal{F}_{\mathsf{AGR}}$ and $\mathcal{F}_{\mathsf{DPNoise}}$ as follows. He forwards the messages and outputs between Party $j \in \mathcal{C}$ and $\mathcal{F}_{\mathsf{Sort}}$, $\mathcal{F}_{\mathsf{LTA}}$, $\mathcal{F}_{\mathsf{CAP}}$, $\mathcal{F}_{\mathsf{AGR}}$ and $\mathcal{F}_{\mathsf{DPNoise}}$. Instead of submitting actual shares, Sim generates shares of 0 and submits the shares on behalf of the honest parties to $\mathcal{F}_{\mathsf{Sort}}$, $\mathcal{F}_{\mathsf{LTA}}$, $\mathcal{F}_{\mathsf{CAP}}$, $\mathcal{F}_{\mathsf{AGR}}$ and $\mathcal{F}_{\mathsf{DPNoise}}$ as in case of $\mathcal{F}_{\mathsf{VRFY}}$.

If $\mathsf{RC} \in \mathcal{C}$, Sim receives the output $\{\mathsf{bk}, \mathsf{DPAG}_{\mathsf{bk}}\}_{\mathsf{bk}\in\mathsf{BK}}$ from $\mathcal{F}_{\mathsf{IPA}}$. Sim uses the shares $\mathsf{L}_j := \{\mathsf{e}_{\mathsf{bk},j}\}_{\mathsf{bk}\in\mathsf{BK}}$ submitted by $j \in \mathcal{C}$ to $\mathcal{F}_{\mathsf{DPNoise}}$ and generates $(\{\mathsf{e}_{\mathsf{bk},1}, \mathsf{e}_{\mathsf{bk},2}, \mathsf{e}_{\mathsf{bk},3}\}_{\mathsf{bk}\in\mathsf{BK}}) \leftarrow \mathsf{share}(\{\mathsf{DPAG}_{\mathsf{bk}}\}_{\mathsf{bk}\in\mathsf{BK}}, j, \{\mathsf{e}_{\mathsf{bk},j}\}_{\mathsf{bk}\in\mathsf{BK}})$. Sim sends $\{\mathsf{bk}, \mathsf{e}_{\mathsf{bk},i}\}_{\mathsf{bk}\in\mathsf{BK}}$ for each $i \in \mathcal{H}$ to RC. Sim forwards the message of Party $j \in \mathcal{C}$ to RC as well.

After the simulator Sim has simulated all subsessions, he outputs the output of the adversary.

We now show that the joint output distribution of Sim and $\mathcal{F}_{\mathsf{IPA}}$ are indistinguishable from the joint output distribution of the adversary and the honest parties (interacting with the adversary). We show this in a sequence of hybrids that we define as follows.

Hybrid_1: Is the real protocol execution between the honest parties and the adversary. It produces the joint output distribution of the adversary and the honest parties (interacting with the adversary).

Hybrid_2: The same as Hybrid_1 except that Hybrid_2 interacts with $\mathcal{F}_{\mathsf{IPA}}$ during the Match Key Generation as follows. When $\mathsf{UA} \in \mathcal{C}$ requests a match key from MKP, Hybrid_2 submits a request $(\mathsf{sid}, \mathsf{ssid}, \mathsf{request}, \mathsf{MKP})$ to $\mathcal{F}_{\mathsf{IPA}}$ rather than submitting it directly to MKP. Hybrid_2 forwards the response of $\mathcal{F}_{\mathsf{IPA}}$ to UA. Further for $\mathsf{MKP} \in \mathcal{C}$, Hybrid_2 receives $(\mathsf{sid}, \mathsf{ssid}, \mathsf{UA})$ from $\mathcal{F}_{\mathsf{IPA}}$, forwards it to MKP. MKP responds with $(\mathsf{sid}, \mathsf{ssid}, \mathsf{mk})$ which Hybrid_2 forwards to $\mathcal{F}_{\mathsf{IPA}}$.

Hybrid_3: The same as Hybrid_2 except that Hybrid_3 interacts with $\mathcal{F}_{\mathsf{IPA}}$ during the Event Generation as follows. Whenever $\mathsf{UA} \in \mathcal{C}$ sends or $\mathsf{RC} \in \mathcal{C}$ receives (from $\mathsf{UA} \in \mathcal{H}$) $\mathsf{rp} := (\mathsf{epid}, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{minf})$, use the secret keys $\mathsf{sk}_i, \mathsf{sk}_k$ for $i, k \in \mathcal{H}$ to decrypt ct_i, ct_k. If decryption fails, set $\mathsf{mk} = \emptyset$. If the shares are inconsistent set $\mathsf{mk} = \mathsf{flag}$. Otherwise, reconstruct mk. If $\mathsf{UA} \in \mathcal{C}$, send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{epid}, \mathsf{mk}, \mathsf{minf}, \mathsf{RC})$ to $\mathcal{F}_{\mathsf{IPA}}$. In any case, receive $(\mathsf{sid}, \mathsf{ssid}, \mathsf{eid}, \mathsf{epid}, \mathsf{minf})$ and store $(\mathsf{eid}, \mathsf{rp}, \mathsf{mk}, \{\mathsf{sh}_j, \mathsf{sh}_{j+1 \bmod 3}\}_{j\in\mathcal{C}})$, where the shares are the decrypted shares of mk. Still forward $(\mathsf{epid}, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{minf})$ to $\mathsf{RC} \in \mathcal{C}$.

Hybrid_4: The same as Hybrid_3 except that Hybrid_4 interacts with $\mathcal{F}_{\mathsf{IPA}}$ during Sharing Reports as follows. Whenever $\mathsf{RC} \in \mathcal{C}$ sends or $\mathsf{RC}' \in \mathcal{C}$ receives (from $\mathsf{RC} \in \mathcal{H}$) $\mathsf{rp} := (\mathsf{epid}, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{ts}, \mathsf{ctRC}_1, \mathsf{ctRC}_2, \mathsf{ctRC}_3)$, use the secret keys $\mathsf{sk}_i, \mathsf{sk}_k$ for $i, k \in \mathcal{H}$ to decrypt ct_i, ct_k, ctRC_i, ctRC_k. If decryption fails, set $\mathsf{mk} = \emptyset$. If the shares are inconsistent set $\mathsf{mk} = \mathsf{flag}$. Otherwise, reconstruct $\mathsf{mk}, \mathsf{ac}, \mathsf{ts}, \mathsf{tb}, \mathsf{bk}, \mathsf{tv}$. If $\mathsf{UA} \in \mathcal{C}$, look up eid

that matches $\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3$ among stored events. (if it doesn't exist, issue event generation query for $\mathsf{RC} \in \mathcal{C}$). Let stored event be rp'. Send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{eid}, \mathsf{ac}, \mathsf{ts}, \mathsf{tb}, \mathsf{bk}, \mathsf{tv}, \mathsf{RC}')$ to $\mathcal{F}_{\mathsf{IPA}}$. In any case, receive $(\mathsf{sid}, \mathsf{ssid}, \mathsf{eid}', \mathsf{epid})$ and store $(\mathsf{eid}', \mathsf{mk}, \mathsf{rp}, \mathsf{rp}')$. Still forward $(\mathsf{epid}, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{ts}, \mathsf{ctRC}_1, \mathsf{ctRC}_2, \mathsf{ctRC}_3)$ to $\mathsf{RC}' \in \mathcal{C}$.

Hybrid_5**:** The same as Hybrid_4 except that Hybrid_5 interacts with $\mathcal{F}_{\mathsf{IPA}}$ as follows. If $\mathsf{RC} \in \mathcal{H}$ and there is a $j \in \mathcal{C}$, receive $\mathsf{L}_{\mathsf{eid}}, \mathsf{dpsp}, \mathsf{dpcap}$ from $\mathcal{F}_{\mathsf{IPA}}$. For every $\mathsf{eid} \in \mathsf{L}_{\mathsf{eid}}$, look up the matching report among the stored reports. If there is a match, use the associated report rp (eid is also matched against (eid,eid') of shared reports). Otherwise (report was generated by $\mathsf{UA} \in \mathcal{H}$), Sim generates an event by generating $(\mathsf{SH}_1, \mathsf{SH}_2, \mathsf{SH}_3) \leftarrow \mathsf{share}(0)$, $(\mathsf{SHRC}_1, \mathsf{SHRC}_2, \mathsf{SHRC}_3) \leftarrow \mathsf{share}(0)$, encrypting them to $\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3$ and $\mathsf{ctRC}_1, \mathsf{ctRC}_2, \mathsf{ctRC}_3$. Sim sends $(\mathsf{epid}, \mathsf{ct}_j, \mathsf{ctRC}_j)$ for all reports together with $\mathsf{dpsp}, \mathsf{dpcap}$ to Party j.

If $\mathsf{RC} \in \mathcal{C}$, receive the reports for each helper party from RC and merge the reports by using $\mathsf{epid}, \mathsf{dpsp}, \mathsf{dpcap}$ of one of the honest parties to $(\mathsf{dpsp}, \mathsf{dpcap}, \mathsf{epid}, \{\mathsf{ad}_j, \mathsf{ct}_{j,i}, \mathsf{ctRC}_{j,i}\}_{j \in [m]})$. Compare $(\mathsf{ct}_{j,1}, \mathsf{ct}_{j,2}, \mathsf{ct}_{j,3})$ with other ciphertexts of stored reports to obtain a corresponding eid which is added to $\mathsf{L}_{\mathsf{eid}}$. If there is no matching ciphertext, use $\mathsf{sk}_i, \mathsf{sk}_k$ for $i, k \in \mathcal{H}$ to decrypt $\mathsf{ct}_{j,i}, \mathsf{ct}_{j,k}$ and the shares to reconstruct mk (for failed decryption set $\mathsf{mk} := \emptyset$, for inconsistent shares set $\mathsf{mk} := \mathsf{flag}$). Issue an Event Generation query for corrupted RC, submitting $(\mathsf{sid}, \mathsf{ssid}, \mathsf{mk}, \mathsf{epid}_j)$ to $\mathcal{F}_{\mathsf{IPA}}$. $\mathcal{F}_{\mathsf{IPA}}$ responds with eid_j which is added to $\mathsf{L}_{\mathsf{eid}}$. After adding one eid to $\mathsf{L}_{\mathsf{eid}}$ for each report, Issue a query to $\mathcal{F}_{\mathsf{IPA}}$. $\mathcal{F}_{\mathsf{IPA}}$ might request $(\mathsf{ac}, \mathsf{ts}, \mathsf{tb}, \mathsf{bk}, \mathsf{tv})$ which is obtained by using $\mathsf{sk}_i, \mathsf{sk}_k$ for $i, k \in \mathcal{H}$ to decrypt $\mathsf{ctRC}_{j,i}, \mathsf{ctRC}_{j,k}$ and using the shares for reconstructing $(\mathsf{ac}, \mathsf{ts}, \mathsf{tb}, \mathsf{bk}, \mathsf{tv})$ (for reports that fail decryption we set $\mathsf{mk} := \emptyset$, for inconsistent shares $\mathsf{mk} := \mathsf{flag}$).

Send $\mathsf{dpsp}, \mathsf{dpcap}, \mathsf{epid}$, which is either obtained from $\mathcal{F}_{\mathsf{IPA}}$ or $\mathsf{RC} \in \mathcal{C}$, for both honest parties to $j \in \mathcal{C}$. Abort if $\mathsf{dpsp}, \mathsf{dpcap}, \mathsf{epid}$ are inconsistent or if $j \in \mathcal{C}$ sends inconsistent values. $\mathcal{F}_{\mathsf{IPA}}$ sends all eid of reports that need to be removed. Inform Party j that those reports need to be removed (includes any report with $\mathsf{mk} = \emptyset$). Party $j \in \mathcal{C}$ might request to remove reports as well. For each of those reports, issue a delete element query to $\mathcal{F}_{\mathsf{IPA}}$. If Party j only requests one of the other helper parties to remove the report, Sim sets $\mathsf{mk} := \mathsf{flag}$ for that report (but does not issue a delete query to $\mathcal{F}_{\mathsf{IPA}}$ for that report).

For all remaining events, forward the messages and outputs between Party $j \in \mathcal{C}$ and $\mathcal{F}_{\mathsf{VRFY}}$. For reports with $\mathsf{mk} = \mathsf{flag}$, send the inconsistent shares (such that they are consistent with the adversaries view) which causes them to be removed. Receive the output for $i \in \mathcal{H}$ from $\mathcal{F}_{\mathsf{VRFY}}$. If it is a 0, Issue a delete query to $\mathcal{F}_{\mathsf{IPA}}$ to delete the report for that event from L.

Hybrid_5 aborts if the list held by $\mathcal{F}_{\mathsf{IPA}}$ contains different events than the list that is secret shared between the parties $i \in \mathcal{H}$.

Hybrid_6**:** The same as Hybrid_5 except that for all subsessions of Query with $\mathsf{RC} \in \mathcal{C}$, parties $i \in \mathcal{H}$ send $\{\mathsf{bk}, \mathsf{e}_{\mathsf{bk},i}\}_{\mathsf{bk}\in\mathsf{BK}}$ to RC as the final output, where $(\{\mathsf{e}_{\mathsf{bk},1}, \mathsf{e}_{\mathsf{bk},2}, \mathsf{e}_{\mathsf{bk},3}\}_{\mathsf{bk}\in\mathsf{BK}}) \leftarrow \mathsf{share}(\{\mathsf{DPAG}_{\mathsf{bk}}\}_{\mathsf{bk}\in\mathsf{BK}}, j, \{\mathsf{e}_{\mathsf{bk},j}\}_{\mathsf{bk}\in\mathsf{BK}})$ and $\{\mathsf{e}_{\mathsf{bk},j}\}_{\mathsf{bk}\in\mathsf{BK}}$ is the share sent by Party $j \in \mathcal{C}$ to $\mathcal{F}_{\mathsf{DPNoise}}$. $\{\mathsf{DPAG}_{\mathsf{bk}}\}_{\mathsf{bk}\in\mathsf{BK}}$ is the output received by Hybrid_6 from $\mathcal{F}_{\mathsf{IPA}}$.

Hybrid_7**:** The same as Hybrid_6 except that the honest parties always submit random shares of 0 to the functionalities $\mathcal{F}_{\mathsf{Sort}}$, $\mathcal{F}_{\mathsf{LTA}}$, $\mathcal{F}_{\mathsf{CAP}}$, $\mathcal{F}_{\mathsf{AGR}}$ and $\mathcal{F}_{\mathsf{DPNoise}}$.

Hybrid_8**:** The same as Hybrid_7 except that the honest parties always submit shares of 0 to the functionalities $\mathcal{F}_{\mathsf{VRFY}}$. To generate the share of 0, look up $\mathsf{SH}_j = (\mathsf{sh}_j, \mathsf{sh}_{j+1 \bmod 3})$ for $j \in \mathcal{C}$. Sample $(\mathsf{SH}_1, \mathsf{SH}_2, \mathsf{SH}_3) \leftarrow \mathsf{share}(0, j, \mathsf{sh}_j)$ or if the decrypted shares are inconsistent (empty), pick inconsistent (empty) shares $\mathsf{SH}_i, \mathsf{SH}_k$ (which only happens if the ciphertext was created by an $\mathsf{UA} \in \mathcal{C}$ or $\mathsf{RC} \in \mathcal{C}$) while still being consistent with sh_j.

Hybrid_9**:** The same as Hybrid_8 except that Hybrid_9 does the following. Let $j \in \mathcal{C}$. Whenever a new report for an honest party is for the first time sent to a corrupted party (either during Event Generation from $\mathsf{UA} \in \mathcal{H}$ to $\mathsf{RC} \in \mathcal{C}$ or during Sharing Reports from $\mathsf{RC} \in \mathcal{H}$ to $\mathsf{RC} \in \mathcal{C}$) instead of sending an encryption of $\mathsf{sh}_{j-1 \bmod 3}$ of the match key (the share that is not in possession of $j \in \mathcal{C}$) under pk_i and pk_k for $i, k \in \mathcal{H}$, send an encryption of $\mathsf{sh}'_{j-1 \bmod 3}$ such that $\mathsf{sh}_j \oplus \mathsf{sh}'_{j-1 \bmod 3} \oplus \mathsf{sh}_{j+1 \bmod 3} = 0$.

Claim 1. Hybrid_1 *and* Hybrid_2 *are indistinguishable.*

Proof. $\mathcal{F}_{\mathsf{IPA}}$ only forwards messages during the Match Key Generation. Therefore it does not change their distribution. Hence, Hybrid_1 and Hybrid_2 send identically distributed messages to the adversary. □

Claim 2. Hybrid_2 *and* Hybrid_3 *are indistinguishable.*

Proof. As in the previous claim, Hybrid_3 does not change the distribution of messages to corrupted parties. Therefore, Hybrid_2 and Hybrid_3 are indistinguishable. □

Claim 3. *Then* Hybrid_3 *and* Hybrid_4 *are indistinguishable.*

Proof. As in the previous claim, Hybrid_4 does not change the distribution of messages to corrupted parties. Therefore, Hybrid_3 and Hybrid_4 are indistinguishable. □

Claim 4. Hybrid_4 *and* Hybrid_5 *are indistinguishable.*

Proof. For $\mathsf{RC} \in \mathcal{C}$, Hybrid_5 receives the reports for each helper party and merges them. If $\mathsf{epid}, \mathsf{dpsp}, \mathsf{dpcap}$ are inconsistent, Hybrid_4 aborts whenever the honest helper parties Hybrid_4 would abort because of inconsistent values. It looks up matching eid for each report that has been submitted. If there is no such report,

Hybrid_4 issues an Event Generation for $\mathsf{RC} \in \mathcal{C}$ to cause $\mathcal{F}_{\mathsf{IPA}}$ to generate a new event for this report. It uses the secret keys of the honest parties to decrypt the ciphertexts. It also uses the keys for request for $\mathsf{ac}, \mathsf{ts}, \mathsf{tb}, \mathsf{bk}, \mathsf{tv}$ from $\mathcal{F}_{\mathsf{IPA}}$. If decryption fails, $\mathsf{mk} = \emptyset$, or if the shares are inconsistent, $\mathsf{mk} = \mathsf{flag}$. If there is no abort, Hybrid_5 submits $\mathsf{epid}, \mathsf{dpsp}, \mathsf{dpcap}$ together with the eid of the reports to $\mathcal{F}_{\mathsf{IPA}}$. $\mathcal{F}_{\mathsf{IPA}}$ will abort if $\mathsf{dpbgt}_{\mathsf{RC},\mathsf{epid}} < \mathsf{dpsp}$. This is also the case in Hybrid_4 where both of the helper parties will abort. Since the values are consistent among honest helper parties (otherwise they abort), it will not happen that one honest helper party will abort but not $\mathcal{F}_{\mathsf{IPA}}$ by triggering the $\mathsf{dpbgt}_{\mathsf{RC},\mathsf{epid}} < \mathsf{dpsp}$ condition. Further, due to this abort condition, $\mathsf{dpbgt}_{\mathsf{RC},\mathsf{epid}}$ remains consistent as long as no RC queries Query concurrently.

Let $\mathsf{RC} \in \mathcal{H}$. In this case, $\mathcal{F}_{\mathsf{IPA}}$ sends $\mathsf{L}_{\mathsf{eid}}, \mathsf{dpsp}, \mathsf{dpcap}$. Similar to Sim, Hybrid_5 looks up the reports that match $\mathsf{eid} \in \mathsf{L}_{\mathsf{eid}}$. If there is no match, it creates the reports as encryptions of shares for 0 (same as Sim). Any report generated by a malicious party will be stored during Event Generation or Sharing Reports. Therefore, they will be matched and the correct report is sent to $j \in \mathcal{C}$ (i.e. consistent with $\mathcal{F}_{\mathsf{IPA}}$). For reports that have not been matched must be reports generated by $\mathsf{UA} \in \mathcal{H}$ and only shared with $\mathsf{RC} \in \mathcal{H}$. Since we only send share SH_j in encrypted form to $j \in \mathcal{C}$, the message $(\mathsf{epid}, \mathsf{ct}_j, \mathsf{ctRC}_j)$ is indistinguishable from any message encrypting one secret share of any different value.

Hybrid_5 then sends the generated reports in case of $\mathsf{RC} \in \mathcal{H}$ or forwards the reports in case of $\mathsf{RC} \in \mathcal{C}$ to $j \in \mathcal{C}$. As argued above, $\mathsf{RC} \in \mathcal{H}$ the distribution of the reports for party j does not differ between Hybrid_4 and Hybrid_5. In case $\mathsf{RC} \in \mathcal{C}$, the reports are just forwarded and therefore the same holds.

Hybrid_5 removes any report with $\mathsf{mk} =$ using a delete report query to Query (Step 2). This will delete all events in L for which ciphertexts cannot be decrypted by at least one honest party. For such events, the honest party in the actual protocol will inform the other honest parties who then will also delete them. Further, Hybrid_5 uses a delete report query to Query (Step 2) for each report that $j \in \mathcal{C}$ requests to be deleted to both honest parties. If just one party is requested to delete it, we flag the report by setting $\mathsf{mk} := \mathsf{flag}$.

Hybrid_5 now forwards messages between $\mathcal{F}_{\mathsf{VRFY}}$ and parties $i \in [3]$. For reports that are flagged with $\mathsf{mk} = \mathsf{flag}$, Hybrid_5 submits inconsistent shares for the honest parties. Notice that only reports are flagged with $\mathsf{mk} = \mathsf{flag}$ when they contained inconsistent or deleted reports to start with. Therefore the inconsistent shares can be generated such that they are consistent with the adversaries view. Hybrid_5 receives the output of $\mathcal{F}_{\mathsf{VRFY}}$ for each report. If the output is 0, i.e. the actual parties have removed it, Hybrid_5 issues a delete query to remove it from $\mathcal{F}_{\mathsf{IPA}}$'s internal list of the Query subsession.

When submitting the initial list, Hybrid_5 ensures that it contains all report IDs eid that represent the reports in the actual query. Missing eid are generated by issuing additional queries and decrypting the corresponding ciphertexts. Hybrid_5 flags bad reports, i.e. reports that fail decryption or contain inconsistent shares and then issues delete queries to remove them from the $\mathcal{F}_{\mathsf{IPA}}$ internal list. The real protocol removes such reports by having the parties interact. During

this interaction the honest parties remove all such bad reports. In this process, $j \in \mathcal{C}$ might remove additional reports for which Hybrid_5 also issues delete queries. Therefore, Hybrid_5 does not abort because of inconsistent lists and hence Hybrid_4 and Hybrid_5 are indistinguishable. □

Claim 5. Hybrid_5 *and* Hybrid_6 *are indistinguishable.*

Proof. By the definition of $\mathcal{F}_{\mathsf{DPNoise}}$, it also generates the shares $\{\mathsf{bk}, \mathsf{e}_{\mathsf{bk},i}\}_{\mathsf{bk}\in\mathsf{BK}}$ for $i \in \mathcal{H}$ in Hybrid_1 by computing

$$(\{\mathsf{e}_{\mathsf{bk},1}, \mathsf{e}_{\mathsf{bk},2}, \mathsf{e}_{\mathsf{bk},3}\}_{\mathsf{bk}\in\mathsf{BK}}) \leftarrow \mathsf{share}(\{\mathsf{DPAG}_{\mathsf{bk}}\}_{\mathsf{bk}\in\mathsf{BK}}, j, \{\mathsf{e}_{\mathsf{bk},j}\}_{\mathsf{bk}\in\mathsf{BK}}).$$

It remains to show that $\{\mathsf{DPAG}_{\mathsf{bk}}\}_{\mathsf{bk}\in\mathsf{BK}}$ generated in Hybrid_6 is identical to the recombination of the shares $\{\mathsf{bk}, \mathsf{e}_{\mathsf{bk},i}\}_{\mathsf{bk}\in\mathsf{BK}}$ for $i \in \mathcal{H}$ in Hybrid_5. This is due to the fact that the internal list during Query of $\mathcal{F}_{\mathsf{IPA}}$ contains the same events as secret shared between the helper parties. Therefore, the same events are contained in the aggregated output. Further, by the definition of $\mathcal{F}_{\mathsf{Sort}}$, $\mathcal{F}_{\mathsf{LTA}}$, $\mathcal{F}_{\mathsf{CAP}}$, $\mathcal{F}_{\mathsf{AGR}}$ and $\mathcal{F}_{\mathsf{DPNoise}}$, the actual protocol helper parties (see Figure 7) process list L identically to $\mathcal{F}_{\mathsf{IPA}}$ (see Figure 1). Therefore, the outcome $\{\mathsf{DPAG}_{\mathsf{bk}}\}_{\mathsf{bk}\in\mathsf{BK}}$ is also identical. □

Claim 6. Hybrid_6 *and* Hybrid_7 *are indistinguishable.*

Proof. The functionalities $\mathcal{F}_{\mathsf{Sort}}$, $\mathcal{F}_{\mathsf{LTA}}$, $\mathcal{F}_{\mathsf{CAP}}$, $\mathcal{F}_{\mathsf{AGR}}$ and $\mathcal{F}_{\mathsf{DPNoise}}$ have the same interaction pattern with the adversary. Namely, the adversary sends its shares to these functionalities and does not receive anything in return. Therefore, having the honest parties sending shares of 0 instead is indistinguishable from the point of view of the adversary. Further, the outputs of the functionalities $\mathcal{F}_{\mathsf{Sort}}$, $\mathcal{F}_{\mathsf{LTA}}$, $\mathcal{F}_{\mathsf{CAP}}$, $\mathcal{F}_{\mathsf{AGR}}$ and $\mathcal{F}_{\mathsf{DPNoise}}$ in Hybrid_6 and Hybrid_7 are independent of the output $\{\mathsf{DPAG}_{\mathsf{bk}}\}_{\mathsf{bk}\in\mathsf{BK}}$ or its shares of the Query procedure. □

Claim 7. Hybrid_7 *and* Hybrid_8 *are indistinguishable.*

Proof. By the definition of $\mathcal{F}_{\mathsf{VRFY}}$, Party $j \in \mathcal{C}$ receives share sh_j. sh_j has the same distribution in Hybrid_7 and Hybrid_8. Further, Party $j \in \mathcal{C}$ receives the outcome of the comparison $\mathsf{sh}_i = \mathsf{sh}'_i$ and $\mathsf{sh}_k = \mathsf{sh}'_k$ for $i, k \in \mathcal{H}$ from $\mathcal{F}_{\mathsf{VRFY}}$. If the shares of the honest parties are consistent in Hybrid_8, then they are also consistent in Hybrid_7. Therefore, the adversary receives the same outputs from $\mathcal{F}_{\mathsf{VRFY}}$ in Hybrid_7 and Hybrid_8. □

Claim 8. *Let* PKE *be IND-CCA-AD secure. Then,* Hybrid_8 *and* Hybrid_9 *are computationally indistinguishable.*

Proof. We use a hybrid argument over both public keys, pk_i and pk_k for $i, k \in \mathcal{H}$ and over every honest event submitted to a corrupted report collector. During each hybrid step, we use the following reduction. Let $\mathsf{sh}_j, \mathsf{sh}_{j+1 \bmod 3}, \mathsf{sh}_{j-1 \bmod 3}$ be the match key shares. We define $\mathsf{sh}'_{j-1 \bmod 3} := 0 \ominus \mathsf{sh}_j \ominus \mathsf{sh}_{j+1 \bmod 3}$ and depending on the hybrid $\mathsf{m}_0 = (\mathsf{sh}_{j-1 \bmod 3}, \mathsf{sh}_j)$, $\mathsf{m}_1 = (\mathsf{sh}'_{j-1 \bmod 3}, \mathsf{sh}_j)$ or $\mathsf{m}_0 = (\mathsf{sh}_{j+1 \bmod 3}, \mathsf{sh}_{j-1 \bmod 3})$, $\mathsf{m}_1 = (\mathsf{sh}_{j-1 \bmod 3}, \mathsf{sh}'_{j-1 \bmod 3})$. We submit

$\mathsf{m}_0, \mathsf{m}_1$ together with ad to the IND-CCA-AD challenge game after receiving pk and depending on the hybrid, setting $\mathsf{pk}_i := \mathsf{pk}$ or $\mathsf{pk}_j := \mathsf{pk}$. Since we lose access to sk_i or sk_j, we use the decryption oracle to decrypt other ciphertexts. Notice that in Hybrid_8, we do not need access to $\mathsf{sh}_{j-1 \bmod 3}$ anymore other than for generating honest user agent events and challenge ciphertexts. For simulating the remaining parts of Hybrid_8, sh_j, $\mathsf{sh}_{j+1 \bmod 3}$ are sufficient which can be always obtained (unless the decryption fails in which case they do not need to be obtained). There is one subtlety here. Namely, a corrupted RC could use the challenge ciphertext for a different epid and submit it during a different subsession as report to Query or Sharing Reports. $\mathcal{F}_{\mathsf{IPA}}$ only allows to generate a new report by either querying Sharing Reports, in which case we do not need to know mk, however it only works generating a report for the same epid or by querying Event Generation as $\mathsf{RC} \in \mathcal{C}$. This however requires mk and hence we would need to decrypt the callenge ciphertext. Since we are using an IND-CCA-AD secure scheme rather than IND-CCA secure, we are allowed to query the decryption oracle for a decryption of the challenge ciphertext as long as epid is different.

Now if an adversary can distinguish two consecutive intermediate hybrid steps, he can break the IND-CCA-AD security. Therefore Hybrid_8 and Hybrid_9 cannot be dististinguished when using an IND-CCA-AD secure PKE. □

We can conclude the theorem by observing that Hybrid_9 resembles Sim and is completely independent of share $\mathsf{sh}_{j-1 \bmod 3}$ of any match key submitted through an honest user agent. Further, the output distribution of Sim is indistinguishable from Hybrid_1 which is the actual protocol. This suffices for concluding the proof. □

7 Basic Functionalities

Open Functionality In Figure 15 we show the ideal functionality to securely

Ideal Functionality $\mathcal{F}_{\mathsf{Open}}$:
1. Receive SH_i from $i \in \mathcal{H}$. Recombine $x = \mathsf{combine}(\{\mathsf{SH}_i\}_{i \in \mathcal{H}})$.
2. Send x to Party $i \in [3]$.

Figure 15: The ideal functionality $\mathcal{F}_{\mathsf{Open}}$ to open secret shares.

open secret shared values.

Multiplication of Shares We define the $\mathcal{F}_{\mathsf{mult}}$ for multiplying secret shared valuse in Figure 16. In Figure 17, we show a protocol realizing $\mathcal{F}_{\mathsf{mult}}$.

Lemma 2. *Let* PRF *be a secure pseudorandom function. Then, the protocol in Figure 17 securely realizes* $\mathcal{F}_{\mathsf{mult}}$ *in the three party setting with at most one corruption.*

Functionality $\mathcal{F}_{\mathsf{mult}}$:

1. Receive $\#g, \mathsf{SH}_{x,i}, \mathsf{SH}_{y,i}, \mathsf{K}_i$ from party $i \in \mathcal{H}$
2. Compute $x := \mathsf{combine}(\{\mathsf{SH}_{x,i}\}_{i\in\mathcal{H}})$, $y := \mathsf{combine}(\{\mathsf{SH}_{y,i}\}_{i\in\mathcal{H}})$ and derive $\mathsf{SH}_{x,j}$, $\mathsf{SH}_{y,j}$ for $j \in \mathcal{C}$ from $\{\mathsf{SH}_{x,i}\}_{i\in\mathcal{H}}$, $\{\mathsf{SH}_{y,i}\}_{i\in\mathcal{H}}$.
3. Send $\mathsf{SH}_{x,j}$, $\mathsf{SH}_{y,j}$, K_j and receive $\mathsf{SH}_{z,j}$ and d from party j for $j \in \mathcal{C}$.
4. $\{\mathsf{SH}_{z,i}\}_{i\in[3]} := \mathsf{share}(x \odot y \oplus d, \mathsf{SH}_{z,j})$
5. Send $\mathsf{SH}_{z,i}$ to party $i \in \mathcal{H}$

Figure 16: The ideal functionality $\mathcal{F}_{\mathsf{mult}}$.

Protocol realizing $\mathcal{F}_{\mathsf{mult}}$:

1. On input $\#g$, $\mathsf{SH}_{x,i} = (\mathsf{sh}_{x,i}, \mathsf{sh}_{x,i+1 \bmod 3})$, $\mathsf{SH}_{y,i} = (\mathsf{sh}_{x,i}, \mathsf{sh}_{x,i+1 \bmod 3})$, k_i, $\mathsf{k}_{i+1 \bmod 3}$ compute $\mathsf{sh}_{z,i} := \mathsf{sh}_{x,i} \odot \mathsf{sh}_{y,i} \oplus \mathsf{sh}_{x,i} \odot \mathsf{sh}_{y,i+1 \bmod 3} \oplus \mathsf{sh}_{x,i+1 \bmod 3} \odot \mathsf{sh}_{y,i} \oplus \mathsf{PRF}(\mathsf{k}_i, \#g) \ominus \mathsf{PRF}(\mathsf{k}_{i+1 \bmod 3}, \#g)$
2. Send $(\#g, \mathsf{sh}_{z,i})$ to party $i-1 \bmod 3$ and receive $(\#g, \mathsf{sh}_{z,i+1 \bmod 3})$ from party $i+1 \bmod 3$.
3. Output share $\mathsf{SH}_{z,i} := (\mathsf{sh}_{z,i}, \mathsf{sh}_{z,i+1 \bmod 3})$.

Figure 17: The protocol realizing $\mathcal{F}_{\mathsf{mult}}$. The gate number $\#g$ is unique for each call to any protocol and ideal functionality. Party $i \in \mathcal{H}$ has access to k_i, $\mathsf{k}_{i+1 \bmod 3}$ which can be used across multiple sessions

Proof. Let $j \in [3]$ be the index of the corrupted party in the protocol of Figure 17. We construct the following simulator that produces the same output distribution as Party j but interacts with ideal functionality $\mathcal{F}_{\mathsf{mult}}$. The simulator receives as input the gate number and PRF keys $\#g, \mathsf{k}_j, \mathsf{k}_{j+1 \bmod 3}$. It also receives $\mathsf{SH}_{x,j} = (\mathsf{sh}_{x,j}, \mathsf{sh}_{x,j+1 \bmod 3})$, $\mathsf{SH}_{y,j} = (\mathsf{sh}_{y,j}, \mathsf{sh}_{y,j+1 \bmod 3})$ from $\mathcal{F}_{\mathsf{mult}}$. It then invokes corrupted party with $\#g, \mathsf{k}_j, \mathsf{k}_{j+1 \bmod 3}, \mathsf{SH}_{x,j}, \mathsf{SH}_{y,j}$ as inputs. Afterwards, it samples a random share $\mathsf{sh}_{z,j+1 \bmod 3} \leftarrow \mathsf{D}_{\mathsf{sh}}$ and sends it to Party j. Party j responds with $\mathsf{sh}_{z,j}$. The simulator then defines $\mathsf{SH}_{z,j} := (\mathsf{sh}_{z,j}, \mathsf{sh}_{z,j+1 \bmod 3})$ and $d := \mathsf{sh}_{z,j} \ominus \mathsf{PRF}(\mathsf{k}_j, \#g) \oplus \mathsf{PRF}(\mathsf{k}_{j+1 \bmod 3}, \#g) \ominus \mathsf{sh}_{x,j} \odot \mathsf{sh}_{y,j} \ominus \mathsf{sh}_{x,j} \odot \mathsf{sh}_{y,j+1 \bmod 3} \ominus \mathsf{sh}_{x,j+1 \bmod 3} \odot \mathsf{sh}_{y,j}$. Finally, the simulator send $\mathsf{SH}_{z,j}$ and d to $\mathcal{F}_{\mathsf{mult}}$ and outputs the output of corrupted Party j.

We now prove that the output distributions are indistinguishable for the environment. We use a sequence of hybrid argument.

Hybrid_1**:** Identical to the real protocol execution. The corrupted party interacts with the honest parties.

Hybrid_2**:** We replace $\mathsf{PRF}(\mathsf{k}_{j-1 \bmod 3}, \#g)$ with $u \leftarrow \mathsf{I}_{\mathsf{PRF}}$ for all computations of all parties, where j is the index of the corrupted party.

Hybrid_3**:** The corrupted party interacts with the simulator. The outputs to the environment are generated by the simulator and $\mathcal{F}_{\mathsf{mult}}$.

Hybrid_1 and Hybrid_2 are indistinguishable by the pseudorandomness of the PRF. Notice that the corrupted party does not have access to PRF key $\mathsf{k}_{j-1 \bmod 3}$. Therefore the reduction is straightforward such that distinguishing Hybrid_1 from Hybrid_2 allows to distinguish $\mathsf{PRF}(\mathsf{k}_{j-1 \bmod 3}, \#g)$ from $u \leftarrow \mathsf{I}_{\mathsf{PRF}}$ and therefore break the pseudorandomness of PRF. The distinguishing probability is therefore upper bounded by a negligible probability.

The main difference between our simulator and an honest party is that the former sends a random share $\mathsf{sh}_{z,j+1 \bmod 3}$ to Party j while the latter sends $\mathsf{sh}_{z,j+1 \bmod 3} := \mathsf{sh}_{x,j+1 \bmod 3} \odot \mathsf{sh}_{y,j+1 \bmod 3} \oplus \mathsf{sh}_{x,j+1 \bmod 3} \odot \mathsf{sh}_{y,j-1 \bmod 3} \oplus \mathsf{sh}_{x,j-1 \bmod 3} \odot \mathsf{sh}_{y,j+1 \bmod 3} \oplus \mathsf{PRF}(\mathsf{k}_{j+1 \bmod 3}, \#g) \ominus \mathsf{PRF}(\mathsf{k}_{j-1 \bmod 3}, \#g)$. In Hybrid_2, $\mathsf{PRF}(\mathsf{k}_{j-1 \bmod 3}, \#g)$ has been replaced with a uniform u. Therefore, the distribution of $\mathsf{sh}_{z,j+1 \bmod 3}$ in Hybrid_2 and Hybrid_3 is identical and cannot be distinguished.

The output of $\mathcal{F}_{\mathsf{mult}}$ is a sharing of $x \odot y \oplus d$ for $d = \mathsf{sh}_{z,j} \ominus \mathsf{PRF}(\mathsf{k}_j, \#g) \oplus \mathsf{PRF}(\mathsf{k}_{j+1 \bmod 3}, \#g) \ominus \mathsf{sh}_{x,j} \odot \mathsf{sh}_{y,j} \ominus \mathsf{sh}_{x,j} \odot \mathsf{sh}_{y,j+1 \bmod 3} \ominus \mathsf{sh}_{x,j+1 \bmod 3} \odot \mathsf{sh}_{y,j}$. Further, the sharing is uniquely defined by $x \odot y \oplus d$ and $\mathsf{SH}_{z,j}$. Therefore, the shares output by $\mathcal{F}_{\mathsf{mult}}$ are identical to the shares generated by the parties in Hybrid_2, which are $\mathsf{sh}_{z,j}, \mathsf{sh}_{z,j+1 \bmod 3}, \mathsf{sh}_{z,j-1 \bmod 3}$ and

$$
\begin{aligned}
& \mathsf{sh}_{z,j} \oplus \mathsf{sh}_{z,j+1 \bmod 3} \oplus \mathsf{sh}_{z,j-1 \bmod 3} \\
= \quad & \mathsf{sh}_{z,j} \oplus \mathsf{sh}_{x,j+1 \bmod 3} \odot \mathsf{sh}_{y,j+1 \bmod 3} \oplus \mathsf{sh}_{x,j+1 \bmod 3} \odot \mathsf{sh}_{y,j-1 \bmod 3} \\
& \oplus \mathsf{sh}_{x,j-1 \bmod 3} \odot \mathsf{sh}_{y,j+1 \bmod 3} \oplus \mathsf{PRF}(\mathsf{k}_{j+1 \bmod 3}, \#g) \ominus u \\
& \oplus \mathsf{sh}_{x,j-1 \bmod 3} \odot \mathsf{sh}_{y,j-1 \bmod 3} \oplus \mathsf{sh}_{x,j-1 \bmod 3} \odot \mathsf{sh}_{y,j} \\
& \oplus \mathsf{sh}_{x,j} \odot \mathsf{sh}_{y,j-1 \bmod 3} \oplus u \ominus \mathsf{PRF}(\mathsf{k}_{j-1 \bmod 3}, \#g) \\
= \quad & d \oplus \mathsf{sh}_{x,j} \odot \mathsf{sh}_{y,j} \oplus \mathsf{sh}_{x,j} \odot \mathsf{sh}_{y,j+1 \bmod 3} \oplus \mathsf{sh}_{x,j+1 \bmod 3} \odot \mathsf{sh}_{y,j} \\
& \oplus \mathsf{sh}_{x,j+1 \bmod 3} \odot \mathsf{sh}_{y,j+1 \bmod 3} \oplus \mathsf{sh}_{x,j+1 \bmod 3} \odot \mathsf{sh}_{y,j-1 \bmod 3} \\
& \oplus \mathsf{sh}_{x,j-1 \bmod 3} \odot \mathsf{sh}_{y,j+1 \bmod 3} \\
& \oplus \mathsf{sh}_{x,j-1 \bmod 3} \odot \mathsf{sh}_{y,j-1 \bmod 3} \oplus \mathsf{sh}_{x,j-1 \bmod 3} \odot \mathsf{sh}_{y,j} \\
& \oplus \mathsf{sh}_{x,j} \odot \mathsf{sh}_{y,j-1 \bmod 3} \\
= \quad & d \oplus (\mathsf{sh}_{x,1} \oplus \mathsf{sh}_{x,2} \oplus \mathsf{sh}_{x,3}) \odot (\mathsf{sh}_{y,1} \oplus \mathsf{sh}_{y,2} \oplus \mathsf{sh}_{y,3}). \\
= \quad & d \oplus x \odot y.
\end{aligned}
$$

□

PRF Key Generation When using $\mathcal{F}_{\mathsf{mult}}$, we need as secret shared PRF Key between the helper parties. We assume that their is a setup phase, that establishes such a PRF Key. The parties can use the protocol that is defined in Figure 18.

Protocol for Generating a Secret Shared PRF Key:

1. $\forall i \in [3]$, Party i samples $\mathsf{k}_i \leftarrow \mathsf{K}$ and sends k_i to Party $i-1 \bmod 3$.

Figure 18: The protocol to generate a secret shared PRF key.

8 Authenticated Secret Sharing

Some functionalities are not secure against additive attack such as $\mathcal{F}_{\mathsf{mult}}$. Further, any functionality in the $\mathcal{F}_{\mathsf{mult}}$ hybrid model also suffers from additive attacks. Nevertheless, we can prevent additive attacks by using a Message Authentication Code (MAC) based authentication mechanism. Such a MAC needs to be verified before secret shared values are opened to ensure that they have not been altered.

This section is closely aligned with the compiler proposed in [17]. Unfortunately, we cannot use their compiler directly since they use a different setup. This is especially true for the share generation. Their compiler is also less modular, while in IPA, we want to have a high level of flexibility. In this section, we therefore frame authenticated secret sharing closer to UC functionalities, that easily allow to replace subprotocols. It also allows to open secret shared values within the protocol, whereas [17] is more geared to open them at the end of the protocol.

Message Authentication Codes We use information-theoretically secure Message Authentication Codes (MAC) to ensure security against malicious behavior. We use $\mu \in \mathbb{N}$ to denote the dimension of the MAC in the context of secret shares over domain D_{sh} and security parameter κ. μ ensures that the MAC provides enough security for security parameter κ. μ is defined as $\mu := \lfloor \frac{\kappa}{\log |\mathsf{D}_{\mathsf{sh}}|} \rfloor$. We overload notation and use μ over different D_{sh}, the associated D_{sh} will be clear from the context.

A MAC for a value $x \in \mathsf{D}_{\mathsf{sh}}$ has typically the form $(\tau^1, \ldots, \tau^\mu) := \tau$ and for $\ell \in [\mu]$, $\tau^\ell := \mathsf{k}_\tau^\ell \odot x$ for a secret key $\mathsf{k}_\tau := (\mathsf{k}_\tau^1, \ldots, \mathsf{k}_\tau^\mu) \leftarrow \mathsf{D}_{\mathsf{sh}}^\mu$. For simplicity, we use $\mathsf{k}_\tau \cdot x$ to denote $(\mathsf{k}_\tau^1 \odot x, \ldots, \mathsf{k}_\tau^\mu \odot x)$. During our protocols, we use a global MAC secret key that is the same for all shares over a specific D_{sh}. For each domain D_{sh}, we use a different key. For $\mathsf{D}_{\mathsf{sh}} = \{0,1\}$, we denote k_τ with k_b and for $\mathsf{D}_{\mathsf{sh}} = \mathbb{Z}_p$, we use k_p.

Similar to x, τ is processed in secret shared form, i.e. for $i \in [3]$, $\mathsf{SM}_i := (\mathsf{SM}_i^1, \ldots, \mathsf{SM}_i^\mu)$ and for $\ell \in [\mu]$, $(\mathsf{SM}_1^\ell, \mathsf{SM}_2^\ell, \mathsf{SM}_3^\ell) \leftarrow \mathsf{share}(\tau^\ell)$, where for $i \in [3]$ $\mathsf{SM}_i^\ell := (\mathsf{sm}_i^\ell, \mathsf{sm}_{i+1 \bmod 3}^\ell)$. For the ease of notation we will sometimes use $(\mathsf{SM}_1, \mathsf{SM}_2, \mathsf{SM}_3) \leftarrow \mathsf{share}(\tau)$ or $\mathsf{SM}_i = (\mathsf{sm}_i, \mathsf{sm}_{i+1 \bmod 3})$.

Similar to the MAC τ, we also process the secret key k_τ in secret shared form, i.e. for $\ell \in [\mu]$, $(\mathsf{SKM}_1^\ell, \mathsf{SKM}_2^\ell, \mathsf{SKM}_3^\ell) \leftarrow \mathsf{share}(\mathsf{k}_\tau^\ell)$, where for $i \in [3]$, $\mathsf{SKM}_i^\ell := (\mathsf{skm}_i^\ell, \mathsf{skm}_{i+1 \bmod 3}^\ell)$. For the ease of notation, we also use $(\mathsf{SKM}_1, \mathsf{SKM}_2, \mathsf{SKM}_3) \leftarrow \mathsf{share}(\mathsf{k}_\tau)$ or $\mathsf{SKM}_i = (\mathsf{skm}_i, \mathsf{skm}_{i+1 \bmod 3})$. For k_b we use $\mathsf{SKB}, \mathsf{skb}$ instead of $\mathsf{SKM}, \mathsf{skm}$ and for k_p we use $\mathsf{SKP}, \mathsf{skp}$.

Protection Against Additive Attacks We show the following useful lemma that shows that shares authenticated with an information theoretic Mac are protected against additive attacks. During an additive attack, an adversary is able to add two offsets d_1, d_2 to a value, Mac tuple x, τ such that the new secret shared values are $x' := x + d_1, \tau' := \tau + d_2$. The next lemma states that such a tuple would not pass the MAC verification procedure that asks that $\tau' = \mathsf{k}_\tau \cdot x'$.

Lemma 3. *For any algorithm* A*, any* $x \in \mathsf{D}_{\mathsf{sh}}$ *and any* $i \in [3]$

$$\Pr[\tau + d_1 = \mathsf{k}_\tau \cdot (x + d_2) \mid (d_1, d_2) \leftarrow \mathsf{A}(\mathsf{SKM}_i, \mathsf{SH}_i, \mathsf{SM}_i), (d_1, d_2) \neq (0, 0)] = \mathsf{negl},$$

where the randomness is taken over $\mathsf{k}_\tau \leftarrow \mathsf{D}^\mu_{\mathsf{sh}}$, $\tau := \mathsf{k}_\tau \cdot x$, $\{\mathsf{SM}_i\}_{i\in[3]} \leftarrow \mathsf{share}(\tau)$, $\{\mathsf{SKM}_i\}_{i\in[3]} \leftarrow \mathsf{share}(\mathsf{k}_\tau)$ *and* $\{\mathsf{SH}_i\}_{i\in[3]} \leftarrow \mathsf{share}(x)$.

Proof. Since $\tau = \mathsf{k}_\tau \cdot x$, $\tau + d_1 = \mathsf{k}_\tau \cdot (x + d_2)$ is equivalent to $d_1 = \mathsf{k}_\tau \cdot d_2$. This equation holds if either $\mathsf{k}_\tau = 0$ which happens with probability $\frac{1}{|\mathsf{D}^\mu_{\mathsf{sh}}|} \leq 2^{-\kappa}$ or $d_2 = 0$ or $\mathsf{k}_\tau = \frac{d_1}{d_2}$. If $d_2 = 0$ it follows that $d_1 = 0$ which violates $(d_1, d_2) \neq (0, 0)$. Further, since $\mathsf{SKM}_i, \mathsf{SH}_i, \mathsf{SM}_i$ are independent of k_τ the probability that $\mathsf{k}_\tau = \frac{d_1}{d_2}$ is identical to the probability that when first picking an element $d_3 = \frac{d_1}{d_2}$, then sampling a $\mathsf{k}_\tau \leftarrow \mathsf{D}^\mu_{\mathsf{sh}}$, k_τ is the same as d_3. This happens with probability $\frac{1}{|\mathsf{D}^\mu_{\mathsf{sh}}|} \leq 2^{-\kappa}$. Therefore, the probability of A picking d_1, d_2 such that the equation holds and $(d_1, d_2) \neq (0, 0)$ is negligible. □

Generation of a MAC Key $\mathcal{F}_{\mathsf{GenMACKey}}$ In Figure 19, we show the ideal

Ideal Functionality $\mathcal{F}_{\mathsf{GenMACKey}}$:
1. Receive SKM_j from $j \in \mathcal{C}$.
2. Sample $\mathsf{k}_\tau \leftarrow \mathsf{D}^\mu_{\mathsf{sh}}$
3. Compute $\{\mathsf{SKM}_i\}_{i\in[3]} \leftarrow \mathsf{share}(\mathsf{k}_\tau, \mathsf{SKM}_j)$.
4. Send SKM_i to $i \in \mathcal{H}$.

Figure 19: The ideal functionality $\mathcal{F}_{\mathsf{GenMACKey}}$ to generate a secret shared MAC key.

functionality that allows to generate a secret shared MAC key.

Generation of Authenticated Shares In Figure 20, we show the ideal functionality that allows to generate authenticated shares.

Verification of Authenticated Shares In Figure 21, we present the functionality that allows to securely verify MACs. If $\mathcal{F}_{\mathsf{MACVrfy}}$ on input LMAC would output 1, we say that the list LMAC *verifies* and denote it with $\mathsf{vrfy}(\mathsf{LMAC})$. We realize $\mathcal{F}_{\mathsf{MACVrfy}}$ by using the protocols in [17] of the verification stage of their compiler.

Ideal Functionality $\mathcal{F}_{\mathsf{GenAShares}}$:

1. Receive SH_i from Party $i \in \mathcal{H}$ as well as SKM_i. Send SH_j, SKM_j to $j \in \mathcal{C}$. Receive SM_j and d from $j \in \mathcal{C}$.
2. Reconstruct $\mathsf{k}_\tau := \mathsf{combine}(\{\mathsf{SKM}_i\}_{i\in\mathcal{H}})$ and $x := \mathsf{combine}(\{\mathsf{SH}_i\}_{i\in\mathcal{H}})$
3. Compute $\tau := x \cdot \mathsf{k}_\tau + d$ and $\{\mathsf{SM}_i\}_{i\in[3]} \leftarrow \mathsf{share}(\tau, \mathsf{SM}_j)$.
4. Send SM_i to $i \in \mathcal{H}$.

Figure 20: The ideal functionality $\mathcal{F}_{\mathsf{GenAShares}}$ to generate authenticated shares up to additive attacks.

Ideal Functionality $\mathcal{F}_{\mathsf{MACVrfy}}$:

1. Receive $\mathsf{LMAC}_{i,\mathsf{k}_\tau}$ from Party $i \in \mathcal{H}$ as well as SKM_i.
2. Reconstruct $\mathsf{k}_\tau := \mathsf{combine}(\{\mathsf{SKM}_i\}_{i\in\mathcal{H}})$.
3. For each row $(\mathsf{SH}_i, \mathsf{SM}_i)_{i\in\mathcal{H}}$ in $(\mathsf{LMAC}_{\mathsf{k}_\tau,i})_{i\in\mathcal{H}}$ reconstruct the shares to $x := \mathsf{combine}(\{\mathsf{SH}_i\}_{i\in\mathcal{H}})$ and $\tau := \mathsf{combine}(\{\mathsf{SM}_i\}_{i\in\mathcal{H}})$. Output 0 and stop iff $\tau \neq \mathsf{k}_\tau \cdot x$.
4. Output 1.

Figure 21: The ideal functionality $\mathcal{F}_{\mathsf{MACVrfy}}$ to verify authenticated shares.

Protocols with Authenticated Shares We formalize the concept of protocols with authenticated shares in Definition 9. Intuitively, we use this concept to capture protocols that are on its own, not sufficient to satisfy UC security and realize an ideal functionality. However, if the shares are actually verified in a following procedure, they are sufficient to UC realize an ideal functionality.

Definition 9 (Protocol with Authenticated Shares)**.** *A protocol* Π *is called a* protocol with authenticated shares *if a subset of the secret shares defined within* Π *are authenticated via a secret shared MAC. This subset is denoted with* LMAC *and contains elements of the form* $(\mathsf{SH}_1, \mathsf{SH}_2, \mathsf{SH}_3, \mathsf{SM}_1, \mathsf{SM}_2, \mathsf{SM}_3)$*. Within* LMAC*, we distinguish between shares that are inputs to* Π *and shares that are outputs or the result of intermediate computations.*

Conditional Realization For protocols with authenticated shares, we weaken the concept of UC realizing an ideal functionality. Different from UC Security, we only ask that the environment cannot distinguish real and ideal world conditioned on LMAC that verifies, i.e. there are no shares in LMAC that fail to authenticate. Further, we ask that the environment cannot distinguish real and ideal world when not given the output of the honest parties, even when the shares in LMAC do not verify. This is a useful notion, since in the later case, the shares in LMAC can actually be verified and the protocol aborted if they do not verify. In that case, the environment does not learn the output of the honest parties.

In this paragraph, we focus on the weakening of UC realizing a functionality,

which we call conditionally realizing a functionality. We emphasize that conditionally realizing a functionality is not sufficient for UC security, but in the following paragraph, we show how to upgrade it to UC security. We give the formal definition of conditional realization in Definition 10.

Definition 10 (Conditional Realization)**.** *A protocol* Π *with list* LMAC *conditionally realizes an ideal functionality* $\mathcal{F}$ *if for any* ppt *adversary* A *corrupting the parties* $\mathcal{C}$*, there exists a simulator* Sim *such that for any* ppt *environment* D *and any polynomial size auxiliary input* z

$$|\Pr[\mathsf{D}(z,(\mathcal{C},\mathcal{H})_\Pi)=1 \mid \mathsf{vrfy}(\mathsf{LMAC})] - \Pr[\mathsf{D}(z,(\mathsf{Sim},\mathcal{F}))=1]| = \mathsf{negl},$$

where all algorithms receive input 1^κ *and* $\mathcal{H}$ *is the set of honest parties. Further, we require that*

$$|\Pr[\mathsf{D}(z,(\mathcal{C} \mid \mathcal{H})_\Pi)=1] - \Pr[\mathsf{D}(z,(\mathsf{Sim} \mid \mathcal{F}))=1]| = \mathsf{negl},$$

where $(\mathcal{C} \mid \mathcal{H})_\Pi$ *is the joint view of the corrupted parties in* $\mathcal{C}$ *during the protocol execution.* $(\mathsf{Sim} \mid \mathcal{F})$ *is the view generated by the simulator when interacting with* $\mathcal{F}$*, but not including the output of* $\mathcal{F}$ *to the honest parties.*

We remark that typically, an ideal functionality $\mathcal{F}$ that opens shares cannot be conditionally realized by a protocol with an LMAC that contains the opened shares unless the protocol UC realizes $\mathcal{F}$. However, if the functionality instead outputs the shares to the parties, it can be. As we will see in the following paragraph, these shares can be opened by the parties after a successful verification of the shares in LMAC.

Universal Composability of Protocols with Authenticated Shares As mentioned before, the intuition behind protocols with authenticated shares is that authenticated shares can be opened if they verify. In that case, they do not leak any sensitive information as captured by the first requirement of Definition 10. If they do not verify, the protocol or functionality is still safe to execute, as captured by the second requirement of Definition 10. However, the parties will need to abort before any of the shares is opened. This can be ensured by actual verifying the shares and aborting if needed. We can therefore translate authenticated functionalities into UC functionalities by adding a verification procedure for the shares in LMAC and aborting if the verification fails. We depict the procedure in Figure 22.

Theorem 2. *Let protocol* Π^{LMAC} *with authenticated shares* LMAC *conditionally realize an ideal functionality* $\mathcal{F}$*. Then, the protocol in Figure 22 UC realizes* $\mathcal{F}$*.*

Proof. The proof follows from the fact that conditionally realization is equivalent to UC realization when LMAC verifies. Further, the corrupted parties are due to Lemma 3 not able to run an additive attack against $\mathcal{F}_{\mathsf{GenAShares}}$ since with overwhelming probability the shares would not verify. This ensures the well-formedness of the initial input shares for Π^{LMAC}.

Securely Compute Π^{LMAC}:

1. Invoke $\mathcal{F}_{\mathsf{GenMACKey}}$ to generate SKM_i for Party i.
2. Invoke $\mathcal{F}_{\mathsf{GenAShares}}$ for all input shares $(\mathsf{SH}_1, \mathsf{SH}_2, \mathsf{SH}_3) \in \mathsf{LMAC}$ and $(\mathsf{SKM}_1, \mathsf{SKM}_2, \mathsf{SKM}_3)$ to generate the corresponding secret shared MAC $(\mathsf{SM}_1, \mathsf{SM}_2, \mathsf{SM}_3)$.
3. Invoke Π^{LMAC}
4. Invoke $\mathcal{F}_{\mathsf{MACVrfy}}$ on input LMAC. Abort if $\mathcal{F}_{\mathsf{MACVrfy}}$ outputs 0.

Figure 22: The secure computation of a protocol Π^{LMAC} with authenticated shares LMAC.

When LMAC does not verify, the environment cannot distinguish the view of the corrupted party in the real word from the view generated from the simulator. However, for UC security, we need that this holds even when the environment receives the output of the honest parties or in the ideal world, the output of the ideal functionality to the honest parties. Since the protocol in Figure 22 aborts when LMAC does not verify before this output is sent to/generated by the honest parties, the environment only receives failure symbol $\perp$ which does allow the environment to distinguish the real world from the ideal world. □

Chaining Protocols with Authenticated Shares We extend the capacity of Theorem 2 by showing that the composition of protocols with authenticated shares is also a protocol with authenticated shares.

Lemma 4. *Let* Π_1^{LMAC} *and* $\Pi_2^{\overline{\mathsf{LMAC}}}$ *be two protocols with authenticated shares* LMAC *and* $\overline{\mathsf{LMAC}}$*. Then the parallel execution of the protocols as well as the composition of the protocols is again a protocol with authenticated shares* $\mathsf{LMAC} \cup \overline{\mathsf{LMAC}}$*. Further, if they conditionally realize functionalities* $\mathcal{F}_1$*,* $\mathcal{F}_2$*, then the parallel execution conditionally realizes the parallel execution of* $\mathcal{F}_1$ *and* $\mathcal{F}_2$ *and the composition conditionally realizes the composition of* $\mathcal{F}_1$ *and* $\mathcal{F}_2$*.*

Proof. The proof follows straightforwardly from the fact that Definition 9 only asks for a list of authenticated shares which is $\mathsf{LMAC} \cup \overline{\mathsf{LMAC}}$ since we do not need to verify the same shares multiple times. Further, conditional realization holds since we can execute the simulators of Π_1^{LMAC} and $\Pi_2^{\overline{\mathsf{LMAC}}}$ in parallel as well as compose them. □

9 Protocols with Authenticated Shares

9.1 Helper Functionalities

For our protocols with authenticated shares, we use the following helper functionalities. For the sake of simplicity, we do not mention the secret shared MACs in the ideal functionality. However, any computation on a secret shared value also needs to be applied to the secret shared MAC of that value. When using

a helper functionality in our protocol, the protocol will also send the secret shared MAC as an input and expect a secret shared MAC of the output value. Similarly, when conditionally realizing the helper functionality via a protocol, we will also explicitly mention the secret shared MACs.

Functionality for Computing Helper Bits The ideal functionality $\mathcal{F}_{\mathsf{HB}}$ for computing helper bits is defined in Figure 23. For the DP capping, we

Ideal Functionality $\mathcal{F}_{\mathsf{HB}}$:

1. $\mathcal{F}_{\mathsf{SHC}}$ receives input L_i from $i \in \mathcal{H}$. Compute $\mathsf{L} := \mathsf{combine}(\{\mathsf{L}_i\}_{i\in\mathcal{H}})$. Send L_j to $j \in \mathcal{C}$.
2. For every $\mathsf{evt} \in \mathsf{L}$, define $\mathsf{evt.hb} := (\mathsf{evt.mk} = (\mathsf{evt}-1).\mathsf{mk}) \wedge (\mathsf{evt.ac} = ((\mathsf{evt}-1).\mathsf{ac}))$. Further, receive $\mathsf{SH}_{\mathsf{evt.hb},j}$ from $j \in \mathcal{C}$, sample $\{\mathsf{SH}_{\mathsf{evt.hb},i}\}_{i\in[3]} \leftarrow \mathsf{share}(\mathsf{evt.hb}, \mathsf{SH}_{\mathsf{evt.hb},j})$. Add $\{\mathsf{SH}_{\mathsf{evt.hb},i}\}$ to L_i.
3. Send L_i to $i \in \mathcal{H}$.

Figure 23: The ideal functionality $\mathcal{F}_{\mathsf{HB}}$ for computing helper bits.

to compute helper bits that ignore the attribution constraint. We call this functionality $\mathcal{F}_{\mathsf{HBC}}$ and define it in in Figure 24. Both functionalities are very

Ideal Functionality $\mathcal{F}_{\mathsf{HBC}}$:

1. $\mathcal{F}_{\mathsf{SHC}}$ receives input L_i from $i \in \mathcal{H}$. Compute $\mathsf{L} := \mathsf{combine}(\{\mathsf{L}_i\}_{i\in\mathcal{H}})$. Send L_j to $j \in \mathcal{C}$.
2. For every $\mathsf{evt} \in \mathsf{L}$, define $\mathsf{evt.hb} := (\mathsf{evt.mk} = (\mathsf{evt}-1).\mathsf{mk})$. Further, receive $\mathsf{SH}_{\mathsf{evt.hb},j}$ from $j \in \mathcal{C}$, sample $\{\mathsf{SH}_{\mathsf{evt.hb},i}\}_{i\in[3]} \leftarrow \mathsf{share}(\mathsf{evt.hb}, \mathsf{SH}_{\mathsf{evt.hb},j})$. Add $\{\mathsf{SH}_{\mathsf{evt.hb},i}\}$ to L_i.
3. Send L_i to $i \in \mathcal{H}$.

Figure 24: The ideal functionality $\mathcal{F}_{\mathsf{HB}}$ for computing helper bits ignoring the attribution constraint.

similar and can be realized using a protocol for a secure comparison.

Functionality for Share Conversion The ideal functionality for share conversion is defined in Figure 25. For the share conversion from $\mathsf{D}^1_{\mathsf{sh}} := \{0,1\}$ to $\mathsf{D}^2_{\mathsf{sh}} := \mathbb{Z}_q$ we use the protocol defined in Figure 26 that we will use as a subroutine in other protocols.

Lemma 5. *The protocol in Figure 26 conditionally realizes $\mathcal{F}_{\mathsf{SHC}}$ in the $\mathcal{F}_{\mathsf{mult}}$ hybrid model for at most one corrupted helper party.*

Proof. The second requirement from Definition 10 trivially holds, because the protocol only outputs secret shared values to the corrupted party which are independent from any actual sensitive value. For the rest of the proof, we focus

Ideal Functionality $\mathcal{F}_{\mathsf{SHC}}$ from $\{0,1\}$ to $\mathbb{Z}_q$:

1. $\mathcal{F}_{\mathsf{SHC}}$ receives input $\mathsf{SH}_{b,i}$ from $i \in \mathcal{H}$. Compute $b \in \{0,1\}$ via $b := \mathsf{combine}(\{\mathsf{SH}_{b,i}\}_{i\in\mathcal{H}})$. Send $\mathsf{SH}_{b,j}$ to $j \in \mathcal{C}$.
2. Convert $b \in \{0,1\}$ to $b' \in \mathbb{Z}_q$, s.t. $0 \in \{0,1\}$ is mapped to $0 \in \mathbb{Z}_q$ and $1 \in \{0,1\}$ is mapped to $1 \in \mathbb{Z}_q$.
3. Receive $\mathsf{SH}_{b',j}$ from $j \in \mathcal{C}$. Sample $(\mathsf{SH}_{b',1}, \mathsf{SH}_{b',2}, \mathsf{SH}_{b',3}) \leftarrow \mathsf{share}(b', j, \mathsf{SH}_{b',j})$.
4. Send $\mathsf{SH}_{b',i}$ to $i \in \mathcal{H}$.

Figure 25: The ideal share conversion functionality $\mathcal{F}_{\mathsf{SHC}}$.

Protocol for Share Conversion from $\{0,1\}$ to $\mathbb{Z}_q$:

1. Party i receives input $\{\mathsf{SH}_{b,i}, \mathsf{SM}_{b,i}\}$ and defines $\mathsf{SH}_{i,i} := (\mathsf{sh}_{b,i}, 0)$, $\mathsf{SH}_{i+1 \bmod 3,i} := (0, \mathsf{sh}_{b,i+1 \bmod 3})$, $\mathsf{SH}_{i-1 \bmod 3,i} := (0,0)$.
2. $\forall i \in [3]$, Party i sends $\mathsf{SH}_{1,i}$ and $\mathsf{SH}_{2,i}$ to $\mathcal{F}_{\mathsf{mult}}$ and receives $\mathsf{SH}_{x,i}$. Further, it sends $\mathsf{SM}_{1,i}$ and $\mathsf{SM}_{2,i}$ to $\mathcal{F}_{\mathsf{mult}}$ and receives $\mathsf{SM}_{x,i}$. Add $(\mathsf{SH}_{x,1}, \mathsf{SH}_{x,2}, \mathsf{SH}_{x,3}, \mathsf{SM}_{x,1}, \mathsf{SM}_{x,2}, \mathsf{SM}_{x,3})$ to LMAC.
3. $\forall i \in [3]$, Party i defines $\mathsf{SH}_{y,i} := \mathsf{SH}_{1,i} \oplus \mathsf{SH}_{2,i} \ominus 2 \odot \mathsf{SH}_{x,i}$ and $\mathsf{SM}_{y,i} := \mathsf{SM}_{1,i} \oplus \mathsf{SM}_{2,i} \ominus 2 \odot \mathsf{SM}_{x,i}$.
4. $\forall i \in [3]$, Party i sends $\mathsf{SH}_{y,i}$ and $\mathsf{SH}_{3,i}$ to $\mathcal{F}_{\mathsf{mult}}$ and receives $\mathsf{SH}_{z,i}$. Further, it sends $\mathsf{SM}_{y,i}$ and $\mathsf{SM}_{3,i}$ to $\mathcal{F}_{\mathsf{mult}}$ and receives $\mathsf{SM}_{z,i}$. Add $(\mathsf{SH}_{z,1}, \mathsf{SH}_{z,2}, \mathsf{SH}_{z,3}, \mathsf{SM}_{z,1}, \mathsf{SM}_{z,2}, \mathsf{SM}_{z,3})$ to LMAC.
5. $\forall i \in [3]$, Party i outputs $\mathsf{SH}_{b',i} := \ominus 2 \odot \mathsf{SH}_{z,i} \oplus \mathsf{SH}_{3,i} \oplus \mathsf{SH}_{y,i}$ and $\mathsf{SM}_{b',i} := \ominus 2 \odot \mathsf{SM}_{z,i} \oplus \mathsf{SM}_{3,i} \oplus \mathsf{SM}_{y,i}$.

Figure 26: The protocol for share conversion in the $\mathcal{F}_{\mathsf{mult}}$ hybrid model.

on the case that LMAC verifies. In that case, we can invoke Lemma 3 to prove that $d = 0$ whenever $\mathcal{F}_{\mathsf{mult}}$ is queried.

We construct the simulator as follows. For the sake of simplicity, we ignore SM in this description and let the simulator do any action on SH also on SM in the same fashion. Sim receives $\mathsf{SH}_{b,j}$ from $\mathcal{F}_{\mathsf{SHC}}$. Whenever $j \in \mathcal{C}$ sends a message to $\mathcal{F}_{\mathsf{mult}}$, forward the message to $\mathcal{F}_{\mathsf{mult}}$. During the interactions between $j \in \mathcal{C}$ and $\mathcal{F}_{\mathsf{mult}}$, Sim learns $\mathsf{SH}_{1,j}, \mathsf{SH}_{2,j}$ from $\mathcal{F}_{\mathsf{mult}}$, $\mathsf{SH}_{x,i}$ from $j \in \mathcal{C}$ during the first multiplication and $\mathsf{SH}_{y,j}, \mathsf{SH}_{3,j}$ from $\mathcal{F}_{\mathsf{mult}}$, $\mathsf{SH}_{z,i}$ from $j \in \mathcal{C}$ during the second multiplication. Since $d = 0$, there is no additive attack. Therefore, $\mathsf{SH}_{1,j}, \mathsf{SH}_{2,j}, \mathsf{SH}_{3,j}, \mathsf{SH}_{y,j}$ are consistent with $\mathsf{SH}_{b,j}$ and follow the protocol distribution. Sim uses the shares to compute $\mathsf{SH}_{b',j}$ according to protocol and submits it to $\mathcal{F}_{\mathsf{SHC}}$. Sim outputs the output of $j \in \mathcal{C}$.

Since Sim is only forwarding messages between $\mathcal{F}_{\mathsf{mult}}$ and $j \in \mathcal{C}$, the output distribution of $j \in \mathcal{C}$ is identical in the real protocol execution and the simulated one.

It remains to show that the honest parties generate the same output as $\mathcal{F}_{\mathsf{SHC}}$. Since there are no additive attacks, the honest parties compute

$$\mathsf{SH}_{b',i} := \ominus 2 \odot \mathsf{SH}_{3,i} \odot (\mathsf{SH}_{1,i} \oplus \mathsf{SH}_{2,i} \ominus 2 \odot \mathsf{SH}_{x,i}) \oplus \mathsf{SH}_{3,i} \oplus \mathsf{SH}_{y,i}$$

Which corresponds to

$$\begin{aligned} b' &= -2 \cdot \mathsf{sh}_{b,3} \cdot (\mathsf{sh}_{b,1} + \mathsf{sh}_{b,2} - 2 \cdot \mathsf{sh}_{b,1} \cdot \mathsf{sh}_{b,2}) \\ &\quad + \mathsf{sh}_{b,3} + \mathsf{sh}_{b,1} + \mathsf{sh}_{b,2} - 2 \cdot \mathsf{sh}_{b,1} \cdot \mathsf{sh}_{b,2} \mod p \\ &= \mathsf{sh}_{b,1}\mathsf{sh}_{b,2}\mathsf{sh}_{b,3} + \mathsf{sh}_{b,1}(1 - \mathsf{sh}_{b,2})(1 - \mathsf{sh}_{b,3}) \\ &\quad + (1 - \mathsf{sh}_{b,1})\mathsf{sh}_{b,2}(1 - \mathsf{sh}_{b,3}) + (1 - \mathsf{sh}_{b,1})(1 - \mathsf{sh}_{b,2})\mathsf{sh}_{b,3} \mod p. \end{aligned}$$

Therefore, b' is 1 if and only if b is 1 and otherwise 0. When $\mathcal{F}_{\mathsf{mult}}$ interacts with Sim it also creates identical shares. Therefore the joint outputs in the real protocol execution are indistinguishable from the ones in the ideal world. □

Functionality for Comparison In Figure 27, we define functionality $\mathcal{F}_{\mathsf{CE}}$ for comparing a secret shared value against a constant.

Ideal Functionality $\mathcal{F}_{\mathsf{CE}}$:

1. $\mathcal{F}_{\mathsf{CE}}$ receives input $\mathsf{SH}_{x,i}$, y from $i \in \mathcal{H}$. Compute $x := \mathsf{combine}(\{\mathsf{SH}_{x,i}\}_{i \in \mathcal{H}})$. Send $\mathsf{SH}_{x,j}$ to $j \in \mathcal{C}$.
2. Set $b = 1 \in \mathbb{Z}_q$ if $x \leq y$ and $b = 0 \in \mathbb{Z}_q$ otherwise. Receive $\mathsf{SH}_{b,j}$ from $j \in \mathcal{C}$, sample $\{\mathsf{SH}_{b,i}\}_{i \in [3]} \leftarrow \mathsf{share}(b, \mathsf{SH}_{b,j})$.
3. Send $\mathsf{SH}_{b,i}$ to $i \in \mathcal{H}$.

Figure 27: The ideal functionality $\mathcal{F}_{\mathsf{CE}}$ for comparing two values where one is a constant in $\mathbb{Z}_q$.

9.2 Protocols during the Query Phase

Oblivious Last Touch Attribution For the Last Touch Attribution, we use an approach similar to the approach used in [25]. This approach allows to compute a prefix sum in $O(\log(|\mathsf{L}|))$ parallel time. The oblivious attribution uses a tree-like structure to compute a prefix sum. The accumulation of sum elements halts whenever a new match or attribution constraint is reached. It also halts when a source event is reached. By using this strategy, the credit value for each source event is the sum of the following trigger events until another source event follows. This computes a last-touch attribution.

In more detail, we use a helper bit indicating whether the previous event has a different match key or attribution constraint. Further, we use a stop bit indicating whether the accumulation is supposed to stop. We iterate in $\lceil \log N \rceil$ many steps and for each step, we iterate over all list elements. More precisely, we use the following approach.

- For $t = 0$, $t < \log(|\mathsf{L}|) + 1$, $t = t + 1$
  - For $r = |\mathsf{L}| - 2^t$, $r > 0$, $r = r - 1$
    If $(\mathsf{L}[r].\mathsf{sb} = 1$ and $\mathsf{L}[r + 2^t].\mathsf{hb} = 1$ and $\mathsf{L}[r + 2^t].\mathsf{tb} = 1)$ then
    $\mathsf{L}[r].\mathsf{cr} = \mathsf{L}[r].\mathsf{cr} + \mathsf{L}[r + 2^t].\mathsf{cr}$
    $\mathsf{L}[r].\mathsf{sb} = \mathsf{L}[r + 2^t].\mathsf{sb}$
    else
    $\mathsf{L}[r].\mathsf{sb} = 0$

The inner part of the two for loops translates into the following arithmetic computation:

$$\mathsf{L}[r].\mathsf{cr} = \mathsf{L}[r].\mathsf{cr} + \mathsf{L}[r].\mathsf{sb} \cdot \mathsf{L}[r + 2^t].\mathsf{hb} \cdot \mathsf{L}[r + 2^t].\mathsf{tb} \cdot \mathsf{L}[r + 2^t].\mathsf{cr}$$

and

$$\mathsf{L}[r].\mathsf{sb} = \mathsf{L}[r + 2^t].\mathsf{sb} \cdot \mathsf{L}[r].\mathsf{sb} \cdot \mathsf{L}[r + 2^t].\mathsf{hb} \cdot \mathsf{L}[r + 2^t].\mathsf{tb}$$

This can be simplified by precomputing $x = \mathsf{L}[r+2^t].\mathsf{hb} \cdot \mathsf{L}[r+2^t].\mathsf{tb}$ and reusing $y = \mathsf{L}[r].\mathsf{sb} \cdot x$ during the computation. In Figure 28, we use this approach to compute a last touch attribution based on secret shared value. The protocol description in the figure does not overwrite variables and instead introduces new variables that are indexed with the iteration counter t. This is helpful for the security analysis and also does not lead to errors when the for inner for loop of the protocol is executed in parallel (due to overwriting values that still need to be read).

Lemma 6. *The protocol in Figure 28 conditionally realizes $\mathcal{F}_{\mathsf{LTA}}$ in the $\mathcal{F}_{\mathsf{mult}}$, $\mathcal{F}_{\mathsf{HB}}$ hybrid model.*

Proof. The second requirement from Definition 10 trivially holds, because the protocol only outputs secret shared values to the corrupted party which are independent from any actual sensitive value. For the rest of the proof, we focus

Protocol conditionally realizing $\mathcal{F}_{\mathsf{LTA}}$:

1. Invoke $\mathcal{F}_{\mathsf{HB}}$ on input L_i from Party $i \in [3]$.
2. For every $r \in [|\mathsf{L}|]$, invoke $\mathcal{F}_{\mathsf{SHC}}$ on input $\mathsf{L}[r].\mathsf{SH}_{\mathsf{hb},i} \in \{0,1\}$ and define $\mathsf{L}[r].\mathsf{SH}_{\mathsf{hb},i} \in \mathbb{Z}_q$ as the output. Repeat for $\mathsf{L}[r].\mathsf{SH}_{\mathsf{tb},i} \in \{0,1\}$.
3. For every $r \in [|\mathsf{L}|]$, initialize $\mathsf{sh}_{\mathsf{sb},r,0,1} := 1 \in \mathbb{Z}_q$, $\mathsf{sh}_{\mathsf{sb},r,0,2} := 0 \in \mathbb{Z}_q$, $\mathsf{sh}_{\mathsf{sb},r,0,3} := 0 \in \mathbb{Z}_q$ and $\mathsf{SH}_{\mathsf{cr},r,0,i} := \mathsf{L}[r].\mathsf{SH}_{\mathsf{tv},i} \in \mathbb{Z}_q$.
4. For every $r \in [|\mathsf{L}|]$, send $\mathsf{L}[r].\mathsf{SH}_{\mathsf{tb},i}$, $\mathsf{L}[r].\mathsf{SH}_{\mathsf{hb},i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{x,r,i}$. Repeat for $\mathsf{L}[r].\mathsf{SH}_{\mathsf{tb},i}$, $\mathsf{L}[r].\mathsf{SM}_{\mathsf{hb},i}$. Add $\{\mathsf{SH}_{x,r,i}, \mathsf{SM}_{x,r,i}\}_{i\in[3]}$ to LMAC.
5. For $(t := 0,\ t < \log(|\mathsf{L}|) + 1,\ t := t + 1)$ compute:
   - For $(r := 1,\ r \le |\mathsf{L}|,\ r := r + 1)$ compute:
     - If $r + 2^t \ge |\mathsf{L}|$
       - Set $\mathsf{SH}_{\mathsf{cr},r,t+1,i} := \mathsf{SH}_{\mathsf{cr},r,t,i}$, $\mathsf{SM}_{\mathsf{cr},r,t+1,i} := \mathsf{SM}_{\mathsf{cr},r,t,i}$.
       - Set $\mathsf{SH}_{\mathsf{sb},r,t+1,i} := 0$, $\mathsf{SM}_{\mathsf{sb},r,t+1,i} := 0$
     - else
       - Send $\mathsf{SH}_{\mathsf{sb},r,t,i}$, $\mathsf{SH}_{x,r+2^t,i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{y,r,t,i}$. Repeat for $\mathsf{SH}_{\mathsf{sb},r,t,i}$, $\mathsf{SM}_{x,r+2^t,i}$. Add $\{\mathsf{SH}_{y,r,t,i}, \mathsf{SM}_{y,r,t,i}\}_{i\in[3]}$ to LMAC.
       - Send $\mathsf{SH}_{y,r,t,i}$, $\mathsf{SH}_{\mathsf{cr},r+2^t,t,i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{z,r,t,i}$. Repeat for $\mathsf{SM}_{y,r,t,i}$, $\mathsf{SH}_{\mathsf{cr},r+2^t,t,i}$. Add $\{\mathsf{SH}_{z,r,t,i}, \mathsf{SM}_{z,r,t,i}\}_{i\in[3]}$ to LMAC.
       - Set $\mathsf{SH}_{\mathsf{cr},r,t+1,i} := \mathsf{SH}_{\mathsf{cr},r,t,i} + \mathsf{SH}_{z,r,t,i}$, $\mathsf{SM}_{\mathsf{cr},r,t+1,i} := \mathsf{SM}_{\mathsf{cr},r,t,i} + \mathsf{SM}_{z,r,t,i}$.
       - Send $\mathsf{SH}_{y,r,t,i}$, $\mathsf{SH}_{\mathsf{sb},r+2^t,t,i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{\mathsf{sb},r,t+1,i}$. Repeat for $\mathsf{SM}_{y,r,t,i}$, $\mathsf{SH}_{\mathsf{sb},r+2^t,t,i}$. Add $\{\mathsf{SH}_{\mathsf{sb},r,t+1,i}, \mathsf{SM}_{\mathsf{sb},r,t+1,i}\}_{i\in[3]}$ to LMAC.
6. For every $r \in [|\mathsf{L}|]$, set $\mathsf{L}[r].\mathsf{SH}_{\mathsf{cr},i} := \mathsf{SH}_{\mathsf{cr},r,\lceil\log(|\mathsf{L}|)\rceil,i}$. Append $\mathsf{L}[r].\mathsf{SH}_{\mathsf{cr},i}$ to L_i.

Figure 28: The protocol for the last touch attribution in the $\mathcal{F}_{\mathsf{mult}}$, $\mathcal{F}_{\mathsf{HB}}$ hybrid model.

on the case that LMAC verifies. In that case, we can invoke Lemma 3 to prove that $d = 0$ whenever $\mathcal{F}_{\mathsf{mult}}$ is queried.

We construct the simulator Sim as follows. It receives L_j from $\mathcal{F}_{\mathsf{LTA}}$. During the computation of $\mathcal{F}_{\mathsf{HB}}$, it sends L_j, to A who responds with $\mathsf{L}[r].\mathsf{SH}_{\mathsf{hb},j}$. Sim updates L_j using $\mathsf{L}[r].\mathsf{SH}_{\mathsf{hb},j}$. It uses the same strategy when $\mathcal{F}_{\mathsf{SHC}}$ is invoked. Sim follows the protocol for any local computation of any shares for $j \in \mathcal{C}$ and whenever $\mathcal{F}_{\mathsf{mult}}$ is invoked, it sends L_j to A and receives the result of $\mathcal{F}_{\mathsf{mult}}$ from A in forms of shares for $j \in \mathcal{C}$. It uses these shares to compute $\mathsf{L}[r].\mathsf{SH}_{\mathsf{cr},i}$ which it then submits to $\mathcal{F}_{\mathsf{LTA}}$. Sim outputs the output of A.

Since Sim follows the protocol description and only forwards messages between A and the hybrid functionalities, the view of A when interacting with Sim is indistinguishable from its view when interacting with the honest parties during the real protocol. We still need to show that the output of the honest parties is consistent with the output of $\mathcal{F}_{\mathsf{LTA}}$. $\mathcal{F}_{\mathsf{LTA}}$ outputs a secret shared list L that is identical to the input list except that it appends the credits cr for each event. According to $\mathcal{F}_{\mathsf{LTA}}$, cr is defined as the sum of trigger values tv of each following events until an event is a source event or has a different match key or attribution constraint. In the latter case, its helper bit is set to zero. We need to show, that in the real protocol execution results, the honest parties produce the same output. Since $d = 0$ for all multiplications, i.e. calls to $\mathcal{F}_{\mathsf{mult}}$), the honest parties will correctly multiply their shares and A is not able to manipulate any of the secret shared values of the honest parties.

By expanding the tree structure, for an event $\mathsf{L}[r]$ and Party i, we obtain (where $d = 0$ for all multiplications, i.e. calls to $\mathcal{F}_{\mathsf{mult}}$):

$$\begin{aligned}
\mathsf{L}[r].\mathsf{SH}_{\mathsf{cr},i} &= \mathsf{SH}_{\mathsf{cr},r,\lceil\log(|\mathsf{L}|)\rceil,i} \\
&= \mathsf{SH}_{\mathsf{cr},r,\lceil\log(|\mathsf{L}|)\rceil-1,i} + \mathsf{SH}_{z,r,\lceil\log(|\mathsf{L}|)\rceil-1,i} \\
&= \mathsf{L}[r].\mathsf{SH}_{\mathsf{tv},i} + \sum_{t=0}^{\lceil\log(|\mathsf{L}|)\rceil-1} \mathsf{SH}_{z,r,t,i} \\
&= \mathsf{L}[r].\mathsf{SH}_{\mathsf{tv},i} + \sum_{t=0}^{\lceil\log(|\mathsf{L}|)\rceil-1} \mathsf{SH}_{y,r,t,i} \cdot \mathsf{SH}_{\mathsf{cr},r+2^t,t,i}
\end{aligned}$$

where $\mathsf{SH}_{y,r,t,i} = \mathsf{SH}_{\mathsf{sb},r,t,i} \cdot \mathsf{L}[r + 2^t].\mathsf{SH}_{\mathsf{tb},i} \cdot \mathsf{L}[r + 2^t].\mathsf{SH}_{\mathsf{hb},i}$ and for $r + 2^t < 1$, $\mathsf{SH}_{\mathsf{cr},r+2^t,t,i} := 0$, $\mathsf{SH}_{y,r,t+1,i} := 0$, $\mathsf{SH}_{z,r,t+1,i} := 0$. For simplicity, we recombine the shares and show that the underlying values have the correct distribution. Let $\mathsf{cr}_{r,t} := \mathsf{combine}(\{\mathsf{SH}_{\mathsf{cr},r,t,i}\}_{i\in\mathcal{H}})$, $y_{r,t} := \mathsf{combine}(\{\mathsf{SH}_{y,r,t,i}\}_{i\in\mathcal{H}})$ and $\mathsf{sb}_{r,t} := \mathsf{combine}(\{\mathsf{SH}_{\mathsf{sb},r,t,i}\}_{i\in\mathcal{H}})$.

We now use a complete induction to show that we obtain the correct sum. For $t = 0$ and all $r \in [|\mathsf{L}|]$, $\mathsf{cr}_{r,t} := \mathsf{L}[r].\mathsf{tv}$ via definition. Now let for all $t < T$ and all r, $\mathsf{cr}_{r,t}$ be the sum of previous events until there is a source event, i.e. $\mathsf{evt.tb} = 0$ or there is different match key or attribution constraint, i.e. $\mathsf{hb} = 0$. We need to show that this also holds for $\mathsf{cr}_{r,T}$. By expanding $\mathsf{cr}_{r,T}$, we obtain $\mathsf{cr}_{r,T} = \mathsf{L}[r].\mathsf{tv} + \sum_{t=0}^{T-1} y_{r,t} \cdot \mathsf{cr}_{r+2^t,t}$. By assumption, all the $\mathsf{cr}_{r+2^t,t}$ components of the sum have the right distribution. By the definition of $y_{r,t}$,

i.e. $y_{r,t} := \mathsf{sb}_{r,t} \cdot \mathsf{L}[r + 2^t].\mathsf{tb} \cdot \mathsf{L}[r + 2^t].\mathsf{hb}$ holds. Therefore, $y_{r,t} = 0$, whenever event $\mathsf{L}[r]$ is a source event, i.e. $\mathsf{L}[r + 2^t].\mathsf{tb} = 0$ or has a different match key or attribution constraint, i.e. $\mathsf{L}[r + 2^t].\mathsf{hb} = 0$.

There are two things that we need to show. First, once a sum is zeroed out, i.e. $y_{r,t} = 0$, all following sums are also zeroed out, i.e. $y_{r,t+s} = 0$ for all $s > 0$. This is easy to show since it is sufficient to show that $y_{r,t} = 0$ implies that $y_{r,t+1} = 0$. Further, since $\mathsf{sb}_{r,t+1}$ is a factor of $y_{r,t+1}$, it is sufficient for us to show that $y_{r,t} = 0$ implies that $\mathsf{sb}_{r,t+1} = 0$. By definition, $\mathsf{sb}_{r,t+1} := y_{r,t} \cdot \mathsf{sb}_{r+2^t,t}$ and therefore the first statement follows.

Second, it could be that there is an event r' that is a source event or has a different match key, but doesn't have a power of two distance to $\mathsf{L}[r]$. In that case the credits $\mathsf{cr}_{r+2^t,t}$ for $t \in \{0, \ldots, T\}$ are still correct per induction assumption. Let $\mathsf{cr}_{r+2^t,t}$ be the sum that includes $\mathsf{L}[r'].\mathsf{tv}$. We need to ensure that all following summands, i.e. $\mathsf{cr}_{r+2^{t'},t'}$ for $t' > t$, are zeroed out, i.e. $y_{r,t'} = 0$. By the first statement, it is sufficient to show this for $t' = t+1$. We can leverage, that some y terms involved in the computation of $\mathsf{cr}_{r+2^t,t}$ are zero to argue that $y_{r,t+1} = 0$. Again, by the first statement, $y_{r'',t} = 0$ for "lower" order term (meaning a term computed during a comparably early iteration t) $y_{r'',t}$ implies that $y_{r'',t+s} = 0$ for any "higher" order term (computed during a comparably late iteration $t+s$) $y_{r'',t+s}$ with $s > 0$. Therefore, there is at least one "highest" order term involved in the computation of $\mathsf{cr}_{r+2^t,t}$ that is zero. By definition,

$$\mathsf{cr}_{r+2^t,t} := \mathsf{L}[r + 2^t].\mathsf{tv} + \sum_{i=0}^{t-1} y_{r+2^t,i} \cdot \mathsf{cr}_{r+2^t+2^i,i}.$$

By expanding the sum for the "highest" order terms (e.g. $i = t - 1$ and then $i = t - 2, \ldots$) we can see that these "highest" order terms are

$$y_{r+2^t,t-1}; y_{r+2^t+2^{t-1},t-2}; \ldots; y_{r+\sum_{i=0}^{s} 2^{t-i},t-1-s}; \ldots.$$

We need to show that if any of these "highest" terms is zero, then $y_{r,t+1} = 0$. Or more formal, for any $s \geq 0$, $y_{r+\sum_{i=0}^{s} 2^{t-i},t-1-s} = 0$ implies $y_{r,t+1} = 0$. By definition, $y_{r,t+1} := \mathsf{sb}_{r,t+1} \cdot \mathsf{L}[r + 2^{t+1}].\mathsf{tb} \cdot \mathsf{L}[r + 2^{t+1}].\mathsf{hb}$. We need to show that $\mathsf{sb}_{r,t+1} = 0$. Again, by definition $\mathsf{sb}_{r,t+1} := y_{r,t} \cdot \mathsf{sb}_{r+2^t,t}$. We can expand $\mathsf{sb}_{r,t+1}$ as follows

$$\mathsf{sb}_{r,t+1} = y_{r,t} \cdot y_{r+2^t,t-1} \cdot \mathsf{sb}_{r+2^t+2^{t-1},t-1} = y_{r,t} \cdot \prod_{s=0} y_{r+\sum_{i=0}^{s} 2^{t-i},t-1-s}.$$

Since any "highest" order term $y_{r+\sum_{i=0}^{s} 2^{t-i},t-1-s}$ is a factor of $\mathsf{sb}_{r,t+1}$, for any $s \geq 0$, $y_{r+\sum_{i=0}^{s} 2^{t-i},t-1-s} = 0$ implies $\mathsf{sb}_{r,t+1} = 0$ and therefore $y_{r,t+1} = 0$. This concludes our complete induction argument. We have shown that the underlying values of the secret shares during the real protocol result in the correct computation of the Last Touch Attribution. Therefore, the secret shares of the hones parties also have the correct distribution and we have shown the lemma. □

DP Capping We define the protocol conditionally realizing $\mathcal{F}_{\mathsf{CAP}}$ in Figure 29. The DP Capping ensures that the contribution of each match key owner is limited to be at most dpcap. This allows to achieve differential privacy guarantees. Further, it also ensures robustness since even if a user agent submits an artificially large trigger value, it will be capped during the DP Capping.

The protocol invokes two sub-protocols. The first one is called Compute Currently Used Budget. This sub-protocol is defined in Figure 30. Compute Currently Used Budget computes a prefix sum over the list. This prefix sum is defined for each event and represents the currently used budget for all source events associated with the same match key. The algorithm to compute the prefix sum is very similar to the algorithm used for computing the Last Touch Attribution. Its computational complexity is $O(\log(|\mathsf{L}|))$ parallel time.

The second protocol is called Overflow Protection and is defined in Figure 31. The Overflow protection ensures that even if the prefix sum representing the currently spend budget exceeds q, i.e. it wraps around and decreases, the budget is still considered spent. Again, we use a similar algorithm as the Last Touch Attribution. It takes $O(\log(|\mathsf{L}|))$ parallel time.

Lemma 7. *Let $2 \cdot \mathsf{dpcap} < q$. Then, the protocol in Figure 29 conditionally realizes $\mathcal{F}_{\mathsf{CAP}}$ in the $\mathcal{F}_{\mathsf{mult}}$, $\mathcal{F}_{\mathsf{HB}}$, $\mathcal{F}_{\mathsf{SHC}}$, $\mathcal{F}_{\mathsf{CE}}$ hybrid model.*

Proof. Similar to the previous proofs, the second requirement from Definition 10 trivially holds, because the protocol only outputs secret shared values to the corrupted party which are independent from any actual sensitive value. For the rest of the proof, we focus on the case that LMAC verifies. In that case, we can invoke Lemma 3 to prove that $d = 0$ whenever $\mathcal{F}_{\mathsf{mult}}$ is queried.

We construct the simulator Sim as follows. It receives L_j from $\mathcal{F}_{\mathsf{CAP}}$. Sim follows the protocol for any local computation of any shares for $j \in \mathcal{C}$ and whenever a hybrid functionality is invoked, it uses L_j to send A the correct shares and receives the results of the hybrid functionality from A in forms of shares for $j \in \mathcal{C}$. It uses these shares to update L_j and any other stored secret shares according to protocol. After all computation is done, Sim submits the shares $\mathsf{L}[r].\mathsf{SH}_{\mathsf{cr},j}$ for all $r \in [|\mathsf{L}|]$ to $\mathcal{F}_{\mathsf{LTA}}$. Sim outputs the output of A.

As before, since Sim follows the protocol description and only forwards messages between A and the hybrid functionalities, the view of A when interacting with Sim is indistinguishable from its view when interacting with the honest parties during the real protocol. It remains to show that the output of the honest parties during the real world protocol execution is indistinguishable from the output of $\mathcal{F}_{\mathsf{CAP}}$ interacting with Sim. In the following, we argue that the underlying values that are secret shared have the correct distribution. It follows that then the secret shares also have the correct distribution, since the secret shares of the honest parties are determined by the shares of $j \in \mathcal{C}$ and the actual values.

As defined in Figure 12, $\mathcal{F}_{\mathsf{CAP}}$ starts with setting the credit of all trigger events to zero. It then iterates through L, starting with the last element in list L. It keeps track of the currently spend budget for each match key. If the budget is spent for event $\mathsf{L}[r]$, the credit for all $\mathsf{L}[r']$ with $r' < r$ is set to zero.

Protocol conditionally realizing $\mathcal{F}_{\mathsf{CAP}}$:

1. Invoke $\mathcal{F}_{\mathsf{HBC}}$ on input L_i from Party $i \in [3]$. For every $r \in [|\mathsf{L}|]$, invoke $\mathcal{F}_{\mathsf{SHC}}$ on input $\mathsf{L}[r].\mathsf{SH}_{\mathsf{hb},i} \in \{0,1\}$ and define $\mathsf{L}[r].\mathsf{SH}_{\mathsf{hb},i} \in \mathbb{Z}_q$ as the output. Repeat for $\mathsf{L}[r].\mathsf{SH}_{\mathsf{tb},i} \in \{0,1\}$.
2. For every $r \in [|\mathsf{L}|]$, compute $1 - \mathsf{tb}$ by setting $\mathsf{L}[r].\mathsf{sh}'_{\mathsf{tb},1} := 1 - \mathsf{L}[r].\mathsf{sh}_{\mathsf{tb},1}$ and $\forall i \in \{2,3\}$, $\mathsf{L}[r].\mathsf{sh}'_{\mathsf{tb},i} := -\mathsf{L}[r].\mathsf{sh}_{\mathsf{tb},i}$. Send $\mathsf{L}[r].\mathsf{SH}'_{\mathsf{tb},i}$, $\mathsf{L}[r].\mathsf{SH}_{\mathsf{cr},i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{\mathsf{cr},r,0,i}$. Repeat for $\mathsf{L}[r].\mathsf{SH}'_{\mathsf{tb},i}$, $\mathsf{L}[r].\mathsf{SM}_{\mathsf{cr},i}$. Add $\{\mathsf{SH}_{\mathsf{cr},r,0,i}, \mathsf{SM}_{\mathsf{cr},r,0,i}\}_{i\in[3]}$ to LMAC.
3. For every $r \in [|\mathsf{L}|]$, send $\mathsf{SH}_{\mathsf{cr},r,0,i}$, $\mathsf{SM}_{\mathsf{cr},r,0,i}$, dpcap to $\mathcal{F}_{\mathsf{CE}}$, receive $\mathsf{SH}_{b,\mathsf{cr},r,i} \in \mathbb{Z}_q$, $\mathsf{SM}_{b,\mathsf{cr},r,i} \in \mathbb{Z}_q$.
4. For every $r \in [|\mathsf{L}|]$, set $\mathsf{sh}'_{\mathsf{cr},r,0,1} := \mathsf{dpcap} - \mathsf{sh}_{\mathsf{cr},r,0,1}$ and $\forall i \in \{2,3\}$, $\mathsf{sh}'_{\mathsf{cr},r,0,i} := -\mathsf{sh}_{\mathsf{cr},r,0,i}$. Send $\mathsf{SH}'_{\mathsf{cr},r,0,i}$, $\mathsf{SH}_{b,\mathsf{cr},r,i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{x,\mathsf{cr},r,i}$. Repeat for $\mathsf{SH}'_{\mathsf{cr},r,0,i}$, $\mathsf{SM}_{b,\mathsf{cr},r,i}$. Add $\{\mathsf{SH}_{x,\mathsf{cr},r,i}, \mathsf{SM}_{x,\mathsf{cr},r,i}\}_{i\in[3]}$ to LMAC.
5. For every $r \in [|\mathsf{L}|]$, set $\mathsf{SH}_{\mathsf{cr},r,0,i} := \mathsf{dpcap} - \mathsf{SH}_{x,\mathsf{cr},r,i}$, $\mathsf{SM}_{\mathsf{cr},r,0,i} := \mathsf{dpcap} \cdot \mathsf{SKM} - \mathsf{SM}_{x,\mathsf{cr},r,i}$.
6. Invoke Compute Currently Used Budget sub-protocol from Figure 30.
7. For every $r \in [|\mathsf{L}|]$, send $\mathsf{SH}_{c,r,i}$, $\mathsf{SM}_{c,r,i}$, dpcap to $\mathcal{F}_{\mathsf{CE}}$, receive $\mathsf{SH}_{b,r,i} \in \mathbb{Z}_q$, $\mathsf{SM}_{b,r,i} \in \mathbb{Z}_q$.
8. Invoke Overflow Protection sub-protocol from Figure 31.
9. For every $r \in [|\mathsf{L}|]$, set $\mathsf{sh}'_{c,r+1,1} := \mathsf{sh}_{c,r+1,1} - \mathsf{dpcap}$ and $\forall i \in \{2,3\}$, $\mathsf{sh}'_{c,r+1,i} := \mathsf{sh}_{c,r+1,i}$. Send $\mathsf{SH}'_{c,r+1,i}$, $\mathsf{SH}_{b,r+1,i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{e,r,i}$. Repeat for $\mathsf{SH}'_{c,r+1,i}$, $\mathsf{SM}_{b,r+1,i}$. Add $\{\mathsf{SH}_{e,r,i}, \mathsf{SM}_{e,r,i}\}_{i\in[3]}$ to LMAC.
10. For every $r \in [|\mathsf{L}|]$, set $\mathsf{sh}'_{e,r,1} := \mathsf{sh}_{e,r,1} + \mathsf{dpcap}$ and $\forall i \in \{2,3\}$, $\mathsf{sh}'_{e,r,i} := \mathsf{sh}_{e,r,i}$. Send $\mathsf{SH}'_{e,r,i}$, $\mathsf{L}[r+1].\mathsf{SH}_{\mathsf{hb},i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{d,r,i}$. Repeat for $\mathsf{SH}'_{e,r,i}$, $\mathsf{L}[r+1].\mathsf{SM}_{\mathsf{hb},i}$. Add $\{\mathsf{SH}_{d,r,i}, \mathsf{SM}_{d,r,i}\}_{i\in[3]}$ to LMAC.
11. For every $r \in [|\mathsf{L}|]$, set $\mathsf{sh}'_{d,r,1} := \mathsf{dpcap} - \mathsf{sh}_{d,r,1}$ and $\forall i \in \{2,3\}$, $\mathsf{sh}'_{d,r,i} := -\mathsf{sh}_{d,r,i}$. Set $\mathsf{SH}_{f,r,i} := \mathsf{SH}_{\mathsf{cr},r,0,i} - \mathsf{SH}'_{d,r,i}$.
12. For every $r \in [|\mathsf{L}|]$, send $\mathsf{SH}_{b,r,i}$, $\mathsf{SH}_{f,r,i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{a,r,i}$. Repeat for $\mathsf{SM}_{b,r,i}$, $\mathsf{SH}_{f,r,i}$. Add $\{\mathsf{SH}_{a,r,i}, \mathsf{SM}_{a,r,i}\}_{i\in[3]}$ to LMAC.
13. For every $r \in [|\mathsf{L}|]$, set $\mathsf{L}[r].\mathsf{SH}_{\mathsf{cr},i} := \mathsf{SH}_{a,r,i} + \mathsf{SH}'_{d,r,i}$, $\mathsf{L}[r].\mathsf{SM}_{\mathsf{cr},i} := \mathsf{SM}_{a,r,i} + \mathsf{SM}'_{d,r,i}$.

Figure 29: The protocol for the DP Capping in the $\mathcal{F}_{\mathsf{mult}}$, $\mathcal{F}_{\mathsf{HB}}$, $\mathcal{F}_{\mathsf{SHC}}$, $\mathcal{F}_{\mathsf{CE}}$ hybrid model.

Compute Currently Used Budget, Sub-Protocol for $\mathcal{F}_{\mathsf{CAP}}$:

1. For every $r \in [|\mathsf{L}|]$, initialize $\mathsf{sh}_{\mathsf{sb},r,0,1} := 1 \in \mathbb{Z}_q$, $\mathsf{sh}_{\mathsf{sb},r,0,2} := 0 \in \mathbb{Z}_q$, $\mathsf{sh}_{\mathsf{sb},r,0,3} := 0 \in \mathbb{Z}_q$.
2. For $(t := 0,\ t < \log(|\mathsf{L}|) + 1,\ t := t + 1)$ compute:
   - For $(r := 1,\ r \leq |\mathsf{L}|,\ r := r + 1)$ compute:
     - If $r + 2^t \geq |\mathsf{L}|$
       - Set $\mathsf{SH}_{\mathsf{cr},r,t+1,i} := \mathsf{SH}_{\mathsf{cr},r,t,i}$, $\mathsf{SM}_{\mathsf{cr},r,t+1,i} := \mathsf{SM}_{\mathsf{cr},r,t,i}$.
       - Set $\mathsf{SH}_{\mathsf{sb},r,t+1,i} := 0$, $\mathsf{SM}_{\mathsf{sb},r,t+1,i} := 0$
     - else
       - Send $\mathsf{SH}_{\mathsf{sb},r,t,i}$, $\mathsf{L}[r + 2^t].\mathsf{SH}_{\mathsf{hb},i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{y,r,t,i}$. Repeat for $\mathsf{SH}_{\mathsf{sb},r,t,i}$, $\mathsf{L}[r + 2^t].\mathsf{SM}_{\mathsf{hb},i}$. Add $\{\mathsf{SH}_{y,r,t,i}, \mathsf{SM}_{y,r,t,i}\}_{i\in[3]}$ to LMAC.
       - Send $\mathsf{SH}_{y,r,t,i}$, $\mathsf{SH}_{\mathsf{cr},r+2^t,t,i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{z,r,t,i}$. Repeat for $\mathsf{SM}_{y,r,t,i}$, $\mathsf{SH}_{\mathsf{cr},r+2^t,t,i}$. Add $\{\mathsf{SH}_{z,r,t,i}, \mathsf{SM}_{z,r,t,i}\}_{i\in[3]}$ to LMAC.
       - Set $\mathsf{SH}_{\mathsf{cr},r,t+1,i} := \mathsf{SH}_{\mathsf{cr},r,t,i} + \mathsf{SH}_{z,r,t,i}$, $\mathsf{SM}_{\mathsf{cr},r,t+1,i} := \mathsf{SM}_{\mathsf{cr},r,t,i} + \mathsf{SM}_{z,r,t,i}$.
       - Send $\mathsf{SH}_{y,r,t,i}$, $\mathsf{SH}_{\mathsf{sb},r+2^t,t,i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{\mathsf{sb},r,t+1,i}$. Repeat for $\mathsf{SM}_{y,r,t,i}$, $\mathsf{SH}_{\mathsf{sb},r+2^t,t,i}$. Add $\{\mathsf{SH}_{\mathsf{sb},r,t+1,i}, \mathsf{SM}_{\mathsf{sb},r,t+1,i}\}_{i\in[3]}$ to LMAC.
3. For every $r \in [|\mathsf{L}|]$, set $\mathsf{SH}_{c,r,i} := \mathsf{SH}_{\mathsf{cr},r,\lceil\log(|\mathsf{L}|)\rceil,i}$.

Figure 30: The sub-protocol Compute Currently Used Budget for the DP Capping in the $\mathcal{F}_{\mathsf{mult}}$ hybrid model.

Overflow Protection, Sub-Protocol for $\mathcal{F}_{\mathsf{CAP}}$:

1. For every $r \in [|\mathsf{L}|]$, initialize $\mathsf{sh}_{\mathsf{sb},r,0,1} := 1 \in \mathbb{Z}_q$, $\mathsf{sh}_{\mathsf{sb},r,0,2} := 0 \in \mathbb{Z}_q$, $\mathsf{sh}_{\mathsf{sb},r,0,3} := 0 \in \mathbb{Z}_q$.
2. For every $r \in [|\mathsf{L}|]$, initialize $\mathsf{SH}_{b,r,0,i} := \mathsf{SH}_{b,r,i}$, $\mathsf{SM}_{b,r,0,i} := \mathsf{SM}_{b,r,i}$.
3. For $(t := 0,\ t < \log(|\mathsf{L}|) + 1,\ t := t + 1)$ compute:
   - For $(r := 1,\ r \leq |\mathsf{L}|,\ r := r + 1)$ compute:
     - If $r + 2^t \geq |\mathsf{L}|$
       - Set $\mathsf{SH}_{b,r,t+1,i} := \mathsf{SH}_{b,r,t,i}$, $\mathsf{SM}_{b,r,t+1,i} := \mathsf{SM}_{b,r,t,i}$.
       - Set $\mathsf{SH}_{\mathsf{sb},r,t+1,i} := 0$, $\mathsf{SM}_{\mathsf{sb},r,t+1,i} := 0$
     - else
       - Send $\mathsf{SH}_{\mathsf{sb},r,t,i}$, $\mathsf{L}[r + 2^t].\mathsf{SH}_{\mathsf{hb},i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{y,r,t,i}$. Repeat for $\mathsf{SH}_{\mathsf{sb},r,t,i}$, $\mathsf{L}[r + 2^t].\mathsf{SM}_{\mathsf{hb},i}$. Add $\{\mathsf{SH}_{y,r,t,i}, \mathsf{SM}_{y,r,t,i}\}_{i\in[3]}$ to LMAC.
       - Send $\mathsf{SH}_{y,r,t,i}$, $\mathsf{SH}_{b,r,t,i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{x,r,t,i}$. Repeat for $\mathsf{SM}_{y,r,t,i}$, $\mathsf{SH}_{b,r,t,i}$. Add $\{\mathsf{SH}_{x,r,t,i}, \mathsf{SM}_{x,r,t,i}\}_{i\in[3]}$ to LMAC.
       - Set $\mathsf{sh}'_{b,r+2^t,t,1} := \mathsf{sh}_{b,r+2^t,t,1} - 1$. $\forall i \in \{2,3\}$, set $\mathsf{sh}'_{b,r+2^t,t,i} := \mathsf{sh}_{b,r+2^t,t,i}$. Send $\mathsf{SH}_{x,r,t,i}$, $\mathsf{SH}'_{b,r+2^t,t,i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{z,r,t,i}$. Repeat for $\mathsf{SM}_{x,r,t,i}$, $\mathsf{SH}'_{b,r,t+2^t,i}$. Add $\{\mathsf{SH}_{z,r,t,i}, \mathsf{SM}_{z,r,t,i}\}_{i\in[3]}$ to LMAC.
       - Set $\mathsf{SH}_{b,r,t+1,i} := \mathsf{SH}_{b,r,t,i} + \mathsf{SH}_{z,r,t,i}$, $\mathsf{SM}_{b,r,t+1,i} := \mathsf{SM}_{b,r,t,i} + \mathsf{SM}_{z,r,t,i}$.
       - Send $\mathsf{SH}_{y,r,t,i}$, $\mathsf{SH}_{\mathsf{sb},r+2^t,t,i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{\mathsf{sb},r,t+1,i}$. Repeat for $\mathsf{SM}_{y,r,t,i}$, $\mathsf{SH}_{\mathsf{sb},r+2^t,t,i}$. Add $\{\mathsf{SH}_{\mathsf{sb},r,t+1,i}, \mathsf{SM}_{\mathsf{sb},r,t+1,i}\}_{i\in[3]}$ to LMAC.
4. For every $r \in [|\mathsf{L}|]$, set $\mathsf{SH}_{b,r,i} := \mathsf{SH}_{b,r,\lceil\log(|\mathsf{L}|)\rceil,i}$.

Figure 31: The sub-protocol Overflow Protection for the DP Capping in the $\mathcal{F}_{\mathsf{mult}}$ hybrid model.

If there is still budget, but $\mathsf{L}[r]$'s credit exceeds the budget, $\mathsf{L}[r]'s$ credit is set to the remaining budget.

The protocol in Figure 29 computes the capping as follows. In Step 2, it initializes $\mathsf{cr}_{r,0} := (1 - \mathsf{tb}) \cdot \mathsf{cr}_r$. Therefore, it $\mathsf{cr}_{r,0}$ to zero for all trigger events. In Step 3, 4 and 5, it sets $\mathsf{cr}_{r,0} := \mathsf{dpcap} - (\mathsf{cr}_{r,0} \leq \mathsf{dpcap}?) \cdot (\mathsf{dpcap} - \mathsf{cr}_{r,0})$. This is equivalent of setting $\mathsf{cr}_{r,0} := \mathsf{cr}_{r,0}$, when $\mathsf{cr}_{r,0} \leq \mathsf{dpcap}$ and $\mathsf{cr}_{r,0} = \mathsf{dpcap}$ otherwise. Since in $\mathcal{F}_{\mathsf{CAP}}$, no credit after the capping can exceed dpcap, this does not result in inconsistency with $\mathcal{F}_{\mathsf{CAP}}$. However, since the protocol computes over $\mathbb{Z}_q$, and since $2 \cdot \mathsf{dpcap} < q$, it ensures that no sum of two credits will exceed q, i.e. for any $r, r' \in \mathsf{L}$ with $\mathsf{cr}_{r,0} + \mathsf{cr}_{r',0} \geq \mathsf{dpcap}$, $(\mathsf{cr}_{r,0} + \mathsf{cr}_{r',0}) \bmod q \geq \mathsf{dpcap}$ holds as well.

The sub protocol Compute Currently Used Budget keeps track of the currently spend budget. L is sorted and therefore events with the same match key are next to each other. Further, we have computed the helper bits in Step 1 to separate events with different match key. It is therefore sufficient to compute a prefix sum that is reset at each helper bit to keep track of the match key specific budget. For now we assume that Compute Currently Used Budget computes the correct prefix sum over the event credits. The variables storing the prefix sum is c_r. If the sub protocol is correct, $c_r = \sum_{r' \in \{r,\ldots,|\mathsf{L}|\}} (\mathsf{L}[r].\mathsf{mk} = \mathsf{L}[r'].\mathsf{mk}) \cdot \mathsf{cr}_{r',0}$ holds. We will show the correctness of the sub protocol later.

Step 7 computes b_r which is 0 when $c_r > \mathsf{dpcap}$ and 1 otherwise. Step 8 invokes the sub-protocol Overflow Protection. Assuming that it is correct (we show correctness later), for all $r, r' \in [|\mathsf{L}]$ with $r' < r$, $\mathsf{L}[r].\mathsf{mk} = \mathsf{L}[r'].\mathsf{mk}$, it holds that if $b_r = 0$ then $b_{r'} = 0$. This is not necessarily true before Step 8 since the prefix sum is computed over Z_q and could wrap around such that it decreases. However, due to Step 3, 4 and 5, as well as $2\mathsf{dpcap} < q$, it is ensured that if the sum exceeds dpcap the first time over the integers, for that event, let it be event r, $b_r = 0$ holds.

Step 9 computes $e_r := b_{r+1} \cdot (c_{r+1} - \mathsf{dpcap})$. Step 10 computes $d_r := (e_r + \mathsf{dpcap}) \cdot \mathsf{L}[r+1].\mathsf{hb}$. Step 11 computes $f_r := \mathsf{cr}_{r,0} - (\mathsf{dpcap} - d_r)$. Step 12 computes $a_r := b_r \cdot f_r$ and Step 13 computes $\mathsf{L}[r].\mathsf{cr} := a_r + (\mathsf{dpcap} - d_r)$. We can summarize these steps by the following computation:

$$\mathsf{L}[r].\mathsf{cr} = a_r + (\mathsf{dpcap} - d_r) = b_r \cdot (\mathsf{cr}_{r,0} - (\mathsf{dpcap} - d_r)) + (\mathsf{dpcap} - d_r)$$

where $d_r = (b_{r+1} \cdot (c_{r+1} - \mathsf{dpcap}) + \mathsf{dpcap}) \cdot \mathsf{L}[r+1].\mathsf{hb}$. If event $r+1$ has a different match key, $\mathsf{L}[r+1] = 0$ and therefore $d_r = 0$. In this case, $\mathsf{L}[r].\mathsf{cr} = b_r \cdot (\mathsf{cr}_{r,0} - \mathsf{dpcap}) + \mathsf{dpcap}$ which is $\mathsf{L}[r].\mathsf{cr} = \mathsf{dpcap}$ if $\mathsf{cr}_{r,0} > \mathsf{dpcap}$ and $\mathsf{L}[r].\mathsf{cr} = \mathsf{cr}_{r,0}c$ otherwise. This is consistent with $\mathcal{F}_{\mathsf{CAP}}$. Let us consider now that $\mathsf{L}[r+1].\mathsf{hb} = 1$, which means that the previous event has the same match key. In this case, $d_r = (b_{r+1} \cdot (c_{r+1} - \mathsf{dpcap}) + \mathsf{dpcap})$. We distinguish now the following cases

$b_{r+1} = 0$: When $b_{r+1} = 0$, it means that the previous event exceeded dpcap already. In this case $d_r = \mathsf{dpcap}$ and therefore $\mathsf{L}[r].\mathsf{cr} = b_r \cdot \mathsf{cr}_{r,0}$. Due to the overflow protection, $b_{r+1} = 0$ implies that $b_r = 0$. Hence $\mathsf{L}[r] = 0$ which is consistent with the specification of $\mathcal{F}_{\mathsf{CAP}}$.

$b_{r+1} = 1$**:** When $b_{r+1} = 1$, it means that the previous event has not exceeded dpcap and therefore there is some budget left. In this case, $d_r = c_{r+1}$ and thus, $\mathsf{L}[r].\mathsf{cr} = b_r \cdot (\mathsf{cr}_{r,0} - (\mathsf{dpcap} - c_{r+1})) + (\mathsf{dpcap} - c_{r+1})$. Now, if $b_r = 0$, i.e. event r exceeds the budget, $\mathsf{L}[r] = \mathsf{dpcap} - c_{r+1}$ which is the remaining budget after event $r+1$ reduces the budget for its credit. In the other case, i.e. $b_r = 1$, there is still enough budget. $\mathsf{L}[r].\mathsf{cr} = (\mathsf{cr}_{r,0} - (\mathsf{dpcap} - c_{r+1})) + (\mathsf{dpcap} - c_{r+1}) = \mathsf{cr}_{r,0}$ and therefore the final credit for event r is identical to the credit before the capping (unless it has been previously capped to dpcap during Step 3,4 and 5).

As we have shown, in all cases the protocol of Figure 29 is consistent with $\mathcal{F}_{\mathsf{CAP}}$. It remains to show that the sub-protocols are correct.

We start with Compute Currently Used Budget. This protocol is very similar to the procotol for $\mathcal{F}_{\mathsf{LTA}}$ and therefore its correctness proof is very similar, too. Therefore, we keep the proof brief. We need to show that for all $r \in [|\mathsf{L}|]$, c_r resembles the correct prefix sum, i.e $c_r = \sum_{r' \in \{r,\dots,|\mathsf{L}|\}} (\mathsf{L}[r].\mathsf{mk} = \mathsf{L}[r'].\mathsf{mk}) \cdot \mathsf{cr}_{r',0}$. Since L is sorted and the helper bits are set correctly, it is sufficient to show that $c_r = \sum_{r' \in \{r,\dots,r'\}} \mathsf{cr}_{r',0}$, were event r' is the closest event with $r' > r$ and $\mathsf{L}[r'+1].\mathsf{hb} = 0$.

According to the protocol, $c_r := \mathsf{cr}_{r,\lceil \log(|\mathsf{L}|) \rceil}$, where $\mathsf{cr}_{r,t+1} := \mathsf{cr}_{r,t} + y_{r,t} \cdot \mathsf{cr}_{r+2^t,t}$. We can expand this to $\mathsf{cr}_{r,t} = \mathsf{cr}_{r,0} + \sum_{s=0}^{t-1} y_{r,s} \cdot \mathsf{cr}_{r-2^s,s}$.

We use a complete induction to show that we obtain the correct sum. For $t = 0$ and all $r \in [|\mathsf{L}|]$, $\mathsf{cr}_{r,0}$ resembles the correct sum via definition, i.e. it is just the sum of the events credit itself. Now let for all $t < T$ and all r, $\mathsf{cr}_{r,t}$ be the sum of previous events until there is different match key, i.e. $\mathsf{hb} = 0$. We need to show that this also holds for $\mathsf{cr}_{r,T}$. By assumption, all the $\mathsf{cr}_{r+2^t,t}$ components of the sum have the right distribution. By the definition of $y_{r,t}$, i.e. $y_{r,t} := \mathsf{sb}_{r,t} \cdot \mathsf{L}[r+2^t].\mathsf{hb}$ holds. Therefore, $y_{r,t} = 0$, whenever event $\mathsf{L}[r+2^t]$ has a different match key, i.e. $\mathsf{L}[r+2^t].\mathsf{hb} = 0$.

There are two things that we need to show. First, once a sum is zeroed out, i.e. $y_{r,t} = 0$, all following sums are also zeroed out, i.e. $y_{r,t+s} = 0$ for all $s > 0$. This is easy to show since it is sufficient to show that $y_{r,t} = 0$ implies that $y_{r,t+1} = 0$. Further, since $\mathsf{sb}_{r,t+1}$ is a factor of $y_{r,t+1}$, it is sufficient for us to show that $y_{r,t} = 0$ implies that $\mathsf{sb}_{r,t+1} = 0$. By definition, $\mathsf{sb}_{r,t+1} := y_{r,t} \cdot \mathsf{sb}_{r+2^t,t}$ and therefore the first statement follows.

Second, it could be that there is an event r' that has a different match key, but doesn't have a power of two distance to $\mathsf{L}[r]$. In that case the credits $\mathsf{cr}_{r+2^t,t}$ for $t \in \{0,\dots,T\}$ are still correct per induction assumption. Let $\mathsf{cr}_{r+2^t,t}$ be the sum that includes $\mathsf{L}[r'].\mathsf{cr}$. We need to ensure that all following summands, i.e. $\mathsf{cr}_{r+2^{t'},t'}$ for $t' > t$, are zeroed out, i.e. $y_{r,t'} = 0$. By the first statement, it is sufficient to show this for $t' = t+1$. We can leverage, that some y terms involved in the computation of $\mathsf{cr}_{r+2^t,t}$ are zero to argue that $y_{r,t+1} = 0$. Again, by the first statement, $y_{r'',t} = 0$ for "lower" order term (meaning a term computed during a comparably early iteration t) $y_{r'',t}$ implies that $y_{r'',t+s} = 0$ for any "higher" order term (computed during a comparably late iteration $t+s$) $y_{r'',t+s}$ with $s > 0$. Therefore, there is at least one "highest" order term involved in the

computation of $\mathsf{cr}_{r+2^t,t}$ that is zero. By definition,

$$\mathsf{cr}_{r+2^t,t} := \mathsf{cr}_{r+2^t,0} + \sum_{i=0}^{t-1} y_{r+2^t,i} \cdot \mathsf{cr}_{r+2^t+2^i,i}.$$

By expanding the sum for the "highest" order terms (e.g. $i = t-1$ and then $i = t-2, \dots$) we can see that these "highest" order terms are

$$y_{r+2^t,t-1}; y_{r+2^t+2^{t-1},t-2}; \dots; y_{r+\sum_{i=0}^{s} 2^{t-i},t-1-s}; \dots.$$

We need to show that if any of these "highest" terms is zero, then $y_{r,t+1} = 0$. Or more formal, for any $s \geq 0$, $y_{r+\sum_{i=0}^{s} 2^{t-i},t-1-s} = 0$ implies $y_{r,t+1} = 0$. By definition, $y_{r,t+1} := \mathsf{sb}_{r,t+1} \cdot \mathsf{L}[r+2^{t+1}].\mathsf{tb} \cdot \mathsf{L}[r+2^{t+1}].\mathsf{hb}$. We need to show that $\mathsf{sb}_{r,t+1} = 0$. Again, by definition $\mathsf{sb}_{r,t+1} := y_{r,t} \cdot \mathsf{sb}_{r+2^t,t}$. We can expand $\mathsf{sb}_{r,t+1}$ as follows

$$\mathsf{sb}_{r,t+1} = y_{r,t} \cdot y_{r+2^t,t-1} \cdot \mathsf{sb}_{r+2^t+2^{t-1},t-1} = y_{r,t} \cdot \prod_{s=0} y_{r+\sum_{i=0}^{s} 2^{t-i},t-1-s}.$$

Since any "highest" order term $y_{r+\sum_{i=0}^{s} 2^{t-i},t-1-s}$ is a factor of $\mathsf{sb}_{r,t+1}$, for any $s \geq 0$, $y_{r+\sum_{i=0}^{s} 2^{t-i},t-1-s} = 0$ implies $\mathsf{sb}_{r,t+1} = 0$ and therefore $y_{r,t+1} = 0$. This concludes our complete induction argument and shows that the sub-protocol is correct.

We now show the correctness of the second sub-protocol, the Overflow Protection. Again, this protocol has a very similar structure. For correctness, we need to show that once $b_r = 0$, $b_{r'} = 0$ for all r' with $r' < r$ and $\mathsf{L}[r].\mathsf{mk} = \mathsf{L}[r'].\mathsf{mk}$. Again, because L is sorted and the helper bits are set correctly, we only need to show that $b_r = 0$ implies $b_{r'} = 0$ for $r' \in \{r, \dots, r''\}$, where r'' is the smallest $r'' > r$ with $\mathsf{L}[r''+1].\mathsf{hb} = 0$.

$y_{r,t}$ and $\mathsf{sb}_{r,t}$ are defined as before. Therefore, we can rely on the previous analysis to obtain that if $y_{r,t} = 0$, then $y_{r,t+s} = 0$ for all $s > 0$. Further, if for any $r' \in \{r, \dots, r + \sum_{i=0}^{t-1} 2^{t-i}\}$, $\mathsf{L}[r'].\mathsf{hb} = 0$, then for any s with $r + 2^s \geq r'$ ($s < \log(r - r')$), $y_{r,s} = 0$. Let r' be the closest event to r with $\mathsf{L}[r'].\mathsf{hb} = 0$. Since for all $r'' \in \{r, \dots, r' - 1\}$, $\mathsf{sb}_{r'',0} = 1$ as well as $\mathsf{L}[r''].\mathsf{hb} = 1$, it follows that for any s with $r + 2^s < r'$, $y_{r,s} = 1$ by the definition of $y_{r,s} := \mathsf{sb}_{r,s} \cdot \mathsf{L}[r+2^s]$ and $\mathsf{sb}_{r,s} := y_{r,s-1} \cdot \mathsf{sb}_{r+2^{t-1},t-1}$.

By definition, $b_r := b_{r,\lceil \log(|\mathsf{L}|) \rceil}$. Further, $b_{r,t+1} := b_{r,t} + (b_{r+2^t,t} - 1) \cdot y_{r,t} \cdot b_{r,t}$.

We now expand $b_{r,t}$ to (assuming $r' > r+3$ and $s' = \lceil \log(r'-r) \rceil - 1$)

$$\begin{aligned}
b_{r,t} &= b_{r,0} + \sum_{s=0}^{s'} (b_{r+2^s,s} - 1) \cdot b_{r,s} \\
&= b_{r,0} + (b_{r+1,0} - 1) b_{r,0} + (b_{r+2,1} - 1) b_{r,1} + \sum_{s=2}^{s'} (b_{r+2^s,s} - 1) \cdot b_{r,s} \\
&= b_{r,0} b_{r+1,0} + (b_{r+2,0} + (b_{r+3,0} - 1) \cdot b_{r+2,0} - 1)(b_{r,0} + (b_{r+1,0} - 1) b_{r,0}) \\
&+ \sum_{s=2}^{s'} (b_{r+2^s,s} - 1) \cdot b_{r,s} \\
&= b_{r,0} b_{r+1,0} b_{r+2,0} b_{r+3,0} + \sum_{s=2}^{s'} (b_{r+2^s,s} - 1) \cdot b_{r,s} \\
&= \prod_{s=0}^{r'-r-1} b_{r+s,0}.
\end{aligned}$$

Therefore, if any event $r'' \in \{r, \ldots, r'-1\}$ has spend the dpcap budget, i.e. $b_{r'',0} = 0$, then $b_{r,\lceil \log(|\mathsf{L}|) \rceil} = 0$ as well, which is what we wanted to show in order to show the correctness of sub-protocol Overflow Protection.

Since both sub-protocols are correct, we have shown the lemma. □

Aggregation There are different approaches to secure aggregation including sorting by the breakdown key and computing a prefix sum with stop bits similar to the last touch attribution. That approach would work well for a large amount of different breakdown categories. We use a more straightforward approach by just using multiplications and comparisons. This works sufficiently well for small amount of breakdown categories. We present the protocol in Figure 32.

Protocol realizing $\mathcal{F}_{\mathsf{AGR}}$ for $\mathsf{BK} = \mathsf{D}^{\mathsf{bk}}_{\mathsf{sh}} \subset \{0,1\}^*$, $\mathsf{cr} \in \mathsf{D}^{\mathsf{cr}}_{\mathsf{sh}} := \mathbb{Z}_p$:

1. For each $r \in [|\mathsf{L}_i|]$ and $\mathsf{bk} \in \mathsf{BK}$, compute $\mathsf{SH}_{b,r,\mathsf{bk},i} := \mathsf{OR}(\neg(\mathsf{bk} \oplus (\mathsf{L}[r]).\mathsf{SH}_{\mathsf{bk},i}))$.
2. For each $r \in [|\mathsf{L}_i|]$ and $\mathsf{bk} \in \mathsf{BK}$, invoke $\mathcal{F}_{\mathsf{SHC}}$ on $\mathsf{SH}_{b,r,\mathsf{bk},i} \in \{0,1\}$ and define the output as $\mathsf{SH}_{b,r,\mathsf{bk},i} \in \mathbb{Z}_q$.
3. For each $r \in [|\mathsf{L}_i|]$ and $\mathsf{bk} \in \mathsf{BK}$, Send $\mathsf{SH}_{b,r,\mathsf{bk},i}$, $\mathsf{L}[r].\mathsf{SH}_{\mathsf{cr},i}$ to $\mathcal{F}_{\mathsf{mult}}$, receive $\mathsf{SH}_{x,\mathsf{bk},r,i}$. Repeat for $\mathsf{SH}_{b,r,\mathsf{bk},i}$, $\mathsf{L}[r].\mathsf{SM}_{\mathsf{cr},i}$. Add $\{\mathsf{SH}_{x,\mathsf{bk},r,i}, \mathsf{SM}_{x,\mathsf{bk},r,i}\}_{i\in[3]}$ to LMAC.
4. Perform an aggregation on L: For each $\mathsf{bk} \in \mathsf{BK}$, compute $\mathsf{SH}_{\mathsf{AG},\mathsf{bk},i} := \sum_{r\in[|\mathsf{L}|]} \mathsf{SH}_{x,\mathsf{bk},r,i}$.

Figure 32: The protocol conditionally realizing functionality $\mathcal{F}_{\mathsf{AGR}}$.

Lemma 8. *The protocol in Figure 32 conditionally realizes $\mathcal{F}_{\mathsf{AGR}}$ in the $\mathcal{F}_{\mathsf{mult}}$, $\mathcal{F}_{\mathsf{SHC}}$ hybrid model.*

Proof. Similar to the previous proofs, the second requirement from Definition 10 trivially holds, because the protocol only outputs secret shared values to the corrupted party which are independent from any actual sensitive value. For the rest of the proof, we focus on the case that LMAC verifies. In that case, we can invoke Lemma 3 to prove that $d = 0$ whenever $\mathcal{F}_{\mathsf{mult}}$ is queried.

We construct the simulator Sim as follows. It receives L_j from $\mathcal{F}_{\mathsf{AGR}}$. Sim follows the protocol for any local computation of any shares for $j \in \mathcal{C}$ and whenever a hybrid functionality is invoked, it uses L_j to send A the correct shares and receives the results of the hybrid functionality from A in forms of shares for $j \in \mathcal{C}$. It uses these shares to update L_j and any other stored secret shares according to protocol. After all computation is done, Sim submits the shares $\mathsf{SH}_{\mathsf{AG},\mathsf{bk},i}$ for all $\mathsf{bk} \in \mathsf{BK}$ to $\mathcal{F}_{\mathsf{AGR}}$. Sim outputs the output of A.

As before, since Sim follows the protocol description and only forwards messages between A and the hybrid functionalities, the view of A when interacting with Sim is indistinguishable from its view when interacting with the honest parties during the real protocol. It remains to show that the output of the honest parties during the real world protocol execution is indistinguishable from the output of $\mathcal{F}_{\mathsf{CAP}}$ interacting with Sim. In the following, we argue that the underlying values that are secret shared have the correct distribution. It follows that then the secret shares also have the correct distribution, since the secret shares of the honest parties are determined by the shares of $j \in \mathcal{C}$ and the actual values.

The correctness straightforwardly follows. During the first step, the helper parties compute the equality between bk and the secret shared bk' of each event and $\mathsf{bk} \in \mathsf{BK}$. This bit is then converted to $\mathbb{Z}_q$ via $\mathcal{F}_{\mathsf{SHC}}$. $x_{\mathsf{bk},r} = \mathsf{L}[r].\mathsf{cr}$ if $\mathsf{bk} = \mathsf{L}[r].\mathsf{bk}$ and $x_{\mathsf{bk},r} = 0$ otherwise. Since the protocol sums up all $x_{\mathsf{bk},r}$ to $\mathsf{AG}_{\mathsf{bk}}$, $\mathsf{AG}_{\mathsf{bk}}$ is consistent with $\mathcal{F}_{\mathsf{AGR}}$ □

10 Performance

We are in the process of implementing our protocol in Rust [6]. Here are some current performance numbers for the query stage showing the network usage. For the MPC we use in the 3-party honest majority setting. In general, in this setting of MPC, network is known to dominate the compute in the overall cost of running the protocol. We expect similar results for IPA.

In table 1 and figure 1 we show the total amount of network used by all the helper parties for different query sizes when running the protocol. For all the runs below we use 16 breakdowns and 40 bit matchkeys and run with malicious security. We have measured for up to 100k records. Scaling is a bit worse than linear as certain stages of the protocol like attribution scale as $O(n \log n)$ in the number of records. Extrapolating to 1M records puts the network around 18GB which at \$0.08 per GB cost \$1.44 (throughout we mean 1 GB = 1024^3 bytes). We also show the breakdown for how much network each stage of the query takes in table 1 and figure 34.

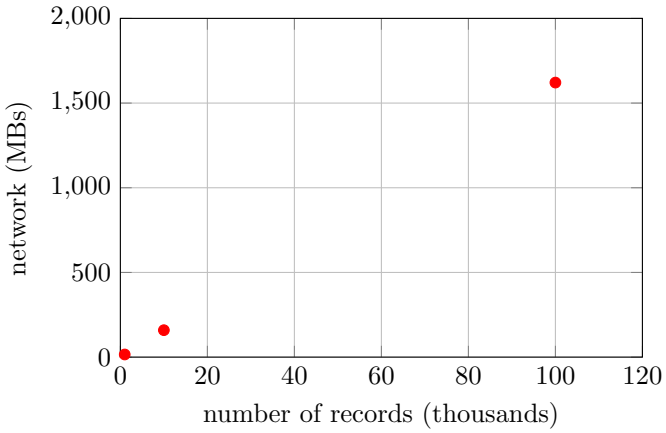

Figure 33: Network used for different size queries

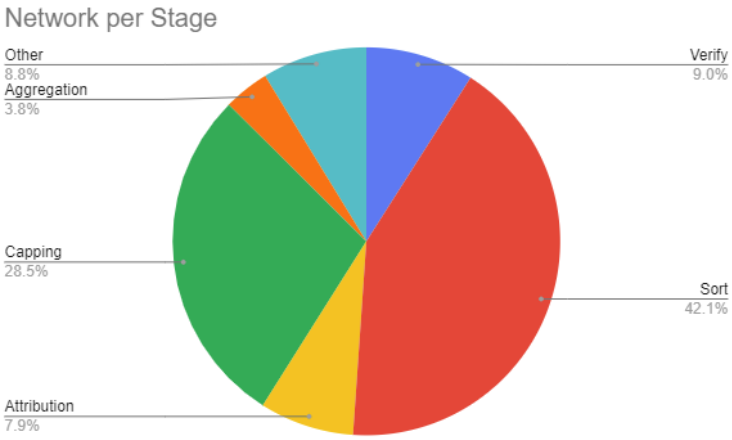

Figure 34: Network percentages per stage

Number of records	Network (MBs)
1000	15.6
10,000	159.1
100,000	1621

Table 1: Network used for different size queries

Stage	Network (MBs)	Percentage
Verify	146	9%
Sort	682	42.1%
Attribution	128	7.9%
Capping	461	28.5%
Aggregation	62	3.8%
Other	141	8.8%

Table 2: Network for different stages of the protocol when run with 100k records.

References

[1] Certificate authority browser forum. https://cabforum.org/. 5

[2] Private Computation Framework 2.0. https://github.com/facebookresearch/Private-ID, 2020. 2

[3] Attribution reporting api. https://github.com/WICG/attribution-reporting-api/blob/main/README.md, 2021. 3

[4] Introducing private click measurement, pcm. https://webkit.org/blog/11529/introducing-private-click-measurement-pcm/, 2021. 3

[5] Privacy-enhancing technologies and building for the future. https://www.facebook.com/business/news/building-for-the-future, 2021. 2

[6] Ipa open source code. https://github.com/private-attribution/ipa, 2022. 52

[7] Our progress on developing and incorporating privacy-enhancing technologies. https://www.facebook.com/business/news/our-progress-on-developing-and-incorporating-privacy-enhancing-technologies, 2022. 2

[8] Private advertising technology community group. https://www.w3.org/community/patcg, 2022. 1, 2

[9] Private Computation Framework 2.0. https://github.com/facebookresearch/fbpcf, 2022. 2

[10] Private measurement important dimensions for attribution. https://github.com/patcg/docs-and-reports/tree/main/design-dimensions, 2022. 3

[11] Google Apple. Exposure notification privacy-preserving analytics (enpa). https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf, 2021. 2

[12] Gilad Asharov, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Ariel Nof, Benny Pinkas, Katsumi Takahashi, and Junichi Tomida. Efficient secure three-party sorting with applications to data analysis and heavy hitters. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 125–138. ACM Press, November 2022. 9, 20

[13] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. *2021 IEEE Symposium on Security and Privacy (SP)*, pages 762–776, 2020. 2

[14] Prasad Buddhavarapu, Benjamin M Case, Logan Gore, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Min Xue. Multi-key private matching for compute. *Cryptology ePrint Archive*, 2021. 2

[15] Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin. Private matching for compute. *Cryptology ePrint Archive*, 2020. 2

[16] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 3–22. Springer, Heidelberg, August 2015. 11

[17] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 34–64. Springer, Heidelberg, August 2018. 6, 32, 33

[18] Aloni Cohen. Attacks on deidentification's defenses. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1469–1486, Boston, MA, August 2022. USENIX Association. 7

[19] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *Symposium on Networked Systems Design and Implementation*, 2017. 2

[20] Hannah Davis, Christopher Patton, Mike Rosulek, and Phillipp Schoppmann. Verifiable distributed aggregation functions. *Cryptology ePrint Archive*, 2023. 2

[21] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 370–389, 2020. 2

[22] Ryo Kikuchi, Nuttapong Attrapadung, Koki Hamada, Dai Ikarashi, Ai Ishida, Takahiro Matsuda, Yusuke Sakai, and Jacob C. N. Schuldt. Field extension in secret-shared form and its applications to efficient secure computation. Cryptology ePrint Archive, Paper 2019/386, 2019. https://eprint.iacr.org/2019/386. 6

[23] Mahnush Movahedi, Benjamin M Case, Andrew Knox, James Honaker, Li Li, Yiming Paul Li, Sanjay Saravanan, Shubho Sengupta, and Erik Taubeneck. Privacy-preserving randomized controlled trials: A protocol for industry scale deployment. *arXiv preprint arXiv:2101.04766*, 2021. 2

[24] Mozilla. Origin telemetry. https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/, 2022. 2

[25] Kartik Nayak, Xiao Shaun Wang, Stratis Ioannidis, Udi Weinsberg, Nina Taft, and Elaine Shi. GraphSC: Parallel secure computation made easy. In *2015 IEEE Symposium on Security and Privacy*, pages 377–394. IEEE Computer Society Press, May 2015. 10, 40

[26] Sikha Pentyala, Davis Railsback, Ricardo Maia, Rafael Dowsley, David Melanson, Anderson Nascimento, and Martine De Cock. Training differentially private models with secure multiparty computation. Cryptology ePrint Archive, Report 2022/146, 2022. https://eprint.iacr.org/2022/146. 22

[27] Joseph J. Pfeiffer, Denis Xavier Charles, Davis Gilton, Young Hun Jung, Mehul Parsana, and Erik Anderson. Masked lark: Masked learning, aggregation and reporting workflow. *ArXiv*, abs/2110.14794, 2021. 2