Masked LARK

Masked Learning, Aggregation & Reporting worKflow

Proposal for Decentralized Aggregation for Conversion Reporting & Modeling

Denis Charles, Joel Pfeiffer, Mehul Parsana, Erik Anderson - Microsoft

Goals of the talk

- Introduce technical aspects of [Masked LARK]
    - Aim of the proposal.
    - Algorithmic & Technical details of Masked LARK.
    - How it differs from existing proposals.
    - Known limitations.
- We will not cover API/Workflow details here.
    - ToDo in a future session.

Background - 3rd party cookies deprecation

Removal of 3rd party cookies impacts the following:

- Fraud detection
- Targeting
- Conversion reporting and modeling

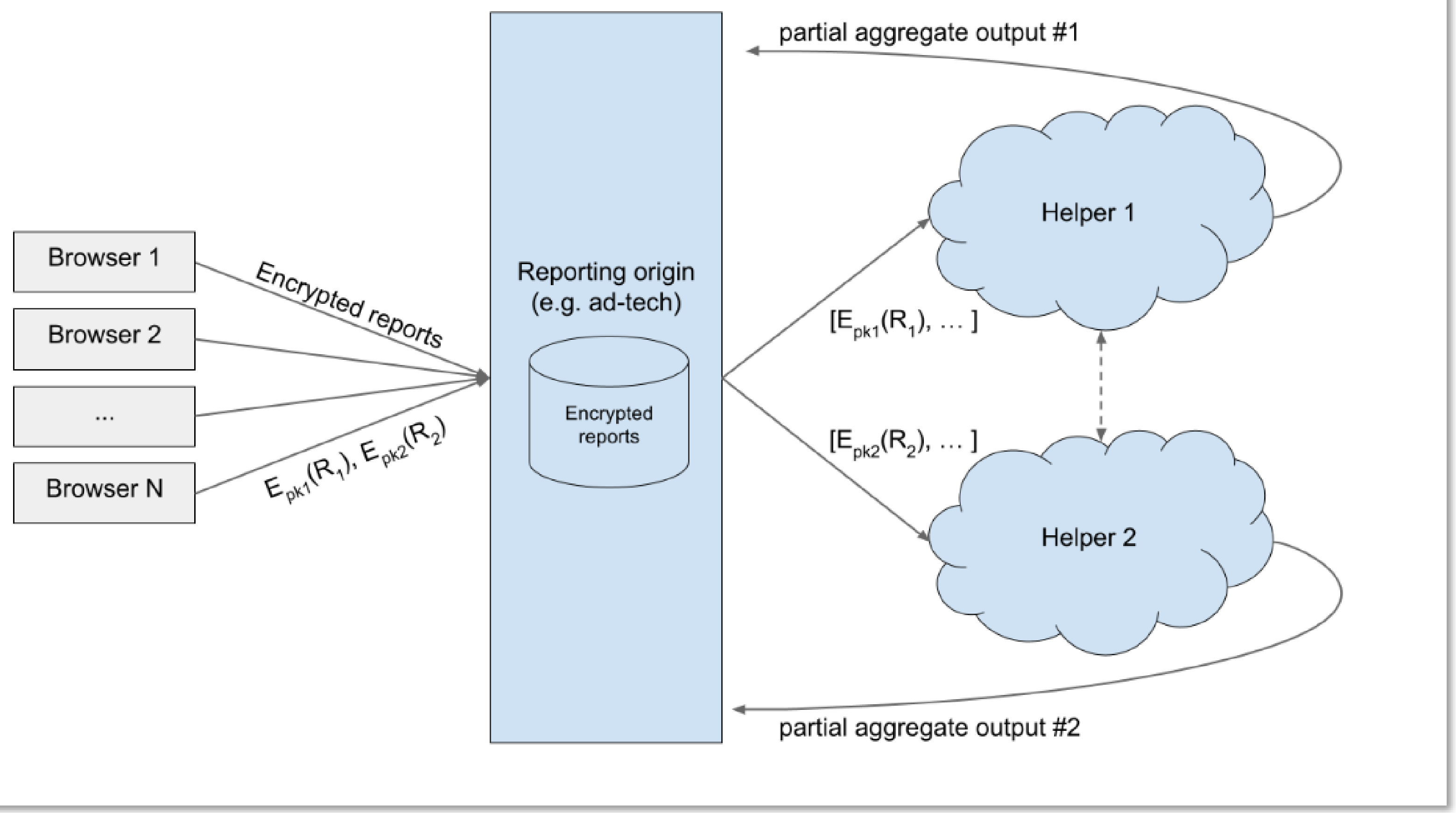
Some Proposals:

- Fraud – Trust tokens
- For targeting
  - FLoC
  - Turtledove
  - Parakeet
- Conversion reporting – Multi-Browser Aggregation Service

Google Proposal for Conversion reporting

Idea:

- Implement trusted mediator abstraction with multiple semi-trusted helpers.
- Use secure Multi-party computation and differential privacy to aggregate data.
- In particular, no single helper has data about final aggregates.



Google Proposal for Conversion reporting

Pros:

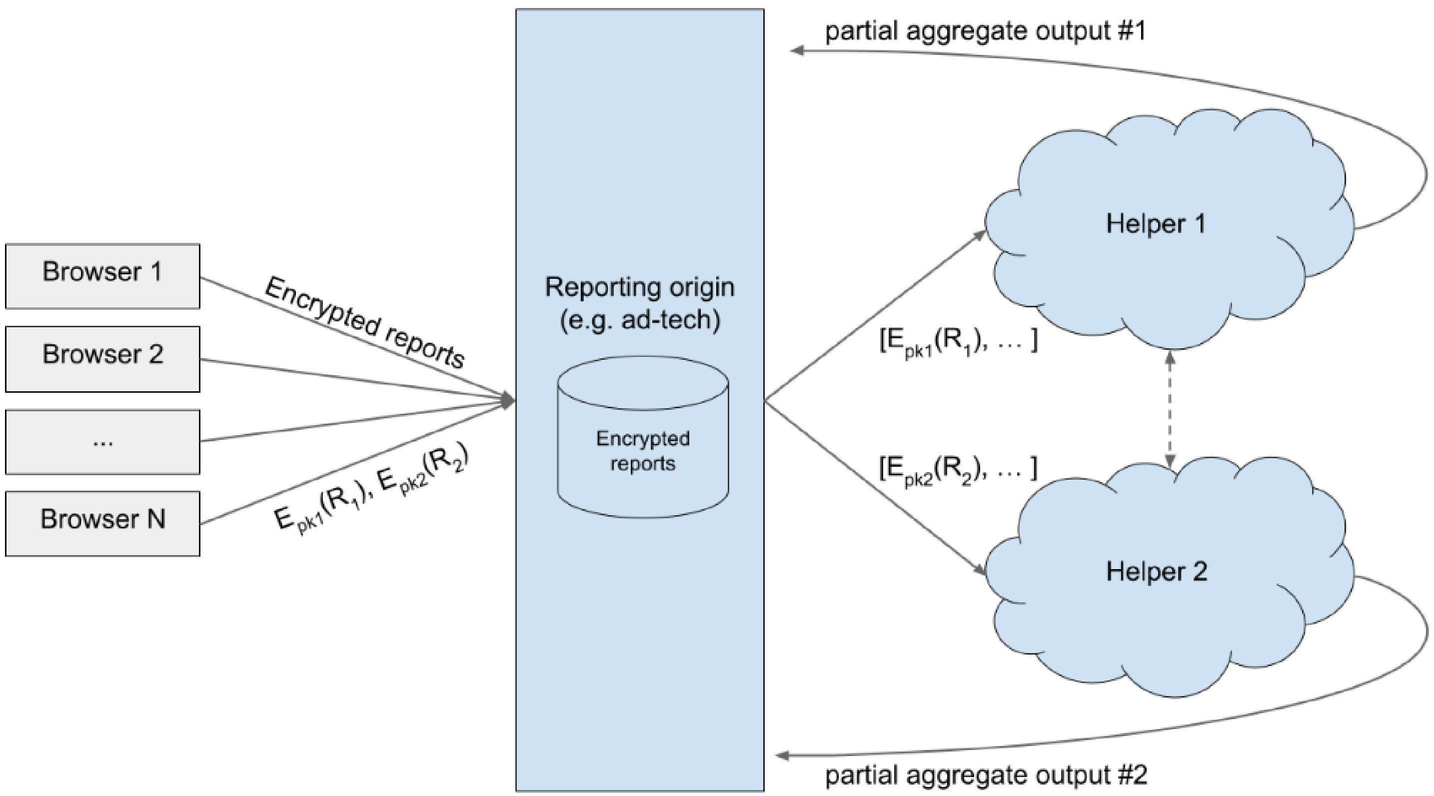
- Segregated helpers implementing aggregation more palatable than single trusted mediator.
- MPC implementation is easier with always available trusted parties.

Limitations:

- Handles aggregation for reporting needs but does not* address modeling.

Our Goal:

- Build on top of proposal to address modeling needs.
- Focus on simple ideas for privacy (minimal crypto – only secret sharing, use only finite rings).



* Can handle with FL & secure aggregation (has disadvantages, discuss later).

General View: Differentially private Map-Reduce

- One can view the aggregation service as implementing a differentially private Map-Reduce framework with semi-trusted helpers.
- Browsers apply a secure “Map” operation.
  - They take local user activity and create keys and data for aggregation.
- Helpers apply a differentially private “Reduce” operation.
  - Any secure MPC function that is differentially private can be a reduce operation. Some examples are sum, approximate rank in sorted list etc.
  - Clarifying requirements for helpers allow us to onboard more helpers.
- Ad Network only sees output of the full process.
- Honesty of helpers can be tested by independent parties (including the ad network).
- Formalization of framework allows users to reason about the system.

Requirements for Helpers

Requirements: Very similar to secure aggregation (reduce operation) with additional differential privacy requirements.

- In some time period T
- Each user (browser) has $\boldsymbol{v_i} \in \mathbb{Z}^d \subseteq \mathbb{R}^d$, this can be client level statistics on user activity
- Helpers compute and output $\boldsymbol{s} \approx \sum_i \boldsymbol{v_i} \equiv \boldsymbol{w}$ with the following properties:
  - **[Noisy aggregation.]** $\boldsymbol{s_i} \in_R \boldsymbol{D}(\boldsymbol{\mu_i} = \boldsymbol{w_i}, \boldsymbol{\sigma_i} = \boldsymbol{\epsilon \mu_i})$**.**
  - **[Sparse reporting constraint/k-Anonymity.]** Another useful property is that if there are $< k$ clients reporting with $\boldsymbol{v}_{i\ell} \neq 0$, then $\boldsymbol{s}_\ell = \emptyset$.
    - In other words, if less that k clients share an aggregation key that key is garbage collected.
- As part of the protocol, AdNetwork publicly discloses ϵ and k.

Notes:

- One difference from Google proposal is there is no explicit hiding of the aggregation keys.
- Technical point, σ_i need to be specified per independent observation $\widehat{\sigma}_i$. Helpers will sample from $D(0, \sigma_i = \widehat{\sigma}_i\sqrt{k})$.

Example: Counting conversions

- Suppose we wish to compute *$Query\_X\_Advertiser$* level #Clicks & #Conversions, with sparse reporting constraint of $\bar{N}$ users.
- Assume we have 2 helpers.
- In Time window T, browser i creates key $k = Enc_{AdNet}(q, a)$ and sets value $\tilde{\boldsymbol{v}} = \langle c_i, conv_i \rangle$.
    - If we have two helpers and suppose we work with ring $\mathbb{Z}/m\mathbb{Z}$, browser generates random values $\langle r_1, r_2 \rangle$ and creates $\boldsymbol{v}_{i,1} = \langle r_1 + c_i, r_2 + conv_i \rangle$ mod m and $\boldsymbol{v}_{i,2} = \langle -r_1, -r_2 \rangle$ mod m.
    - Browser encrypts $v_{i,1}$ and $v_{i,2}$ using public key of the helpers.
- Aggregation service computes $\langle \sum clicks_i , \sum conv_i \rangle + \boldsymbol{\epsilon}_{noise} \approx \sum_{i,j} v_{i,j}$.
- If number of *distinct users* reporting information for key, k $= (q, a)$, is $\leq N$ we get #clicks = ∅ and #conv = ∅.
    - **Note:** Since each helper sees data from every user reporting a key. They can individually apply the sparse reporting constraint.
- This encourages aggregations to not be granular. Perhaps use query categories for aggregation or require larger time window.

Model training

- For differentiable models M:
  - For Model training set $\boldsymbol{v_i} = \boldsymbol{\partial}\ \boldsymbol{loss}(\boldsymbol{f_i}, \boldsymbol{label_i}, \boldsymbol{M})$, $\boldsymbol{f_i}$ = local feature vector.
  - If we use models with count features, we can update aggregates by the protocol directly.
- **Potential approaches:**
  - Since $\boldsymbol{v_i}$ in this case depends on actual label and model, it needs to be computed on user's device. This implies a federated learning setup is required.
    - May be expensive to do on user's device as number of models can be large.
  - If user trusts helper with label, gradient computation can be done at helper.
    - Trust in single helper is too strong of an assumption.
  - Another alternative is for user to send true label to aggregate with probability p and random label with probability $1 - p$.
    - Not great from privacy standpoint.

Our solution - Masked aggregation

Outline of solution in simplified setting.

For binary labels (like conversion models), we can do the following.

- Browser sends both $\langle f_i, label = 0\rangle$ and $\langle f_i, label = 1\rangle$ to each helper.
  - **Note:** f_i is known to AdNetwork, so AdNetwork can provide feature vector to helper. This saves communication cost on user side.
- Helpers compute two gradient vectors $\boldsymbol{g_{i,0}} = \boldsymbol{\partial\, loss}(\boldsymbol{f_i}, \boldsymbol{0}, \boldsymbol{M})$and $\boldsymbol{g_{i,1}} = \boldsymbol{\partial\, loss}(\boldsymbol{f_i}, \boldsymbol{1}, \boldsymbol{M})$, where we encode these gradients in $(\mathbb{Z}/m\mathbb{Z})^d$.
- Separately, browser generates $\langle \alpha_{i,0}, \alpha_{i,1}\rangle$ and $\langle \beta_{i,0}, \beta_{i,1}\rangle$.
  - If 0 is the true label, $\alpha_{i,0} + \alpha_{i,1} = 1$ and 0 otherwise.
  - Similarly, if 1 is the true label $\beta_{i,0} + \beta_{i,1} = 1$ and 0 otherwise.
  - Here $\alpha_{i,k} = r_i$ where r_i is random in $\mathbb{Z}/m\mathbb{Z}$, and we set $\alpha_{i,0} + \alpha_{i,1} = \mathbf{1}[label = 0]$ (mod m) as above. Similarly, we set $\beta_{i,k}$.
- Browser sends $\alpha_{i,k}$ and $\beta_{i,k}$ to Helper k.
- Finally, helpers compute gradient $\mathbf{G} = \sum_{i,k} \alpha_{i,k} g_{i,0} + \beta_{i,k} g_{i,1}$.
  - For each i, we have by construction that $\sum_k \alpha_{i,k} g_{i,0} + \beta_{i,k} g_{i,1} \equiv True\ gradient$.
- $\boldsymbol{G}$ is revealed to AdNetwork after the differential privacy constraints are applied. This is used by AdNetwork to update model.

Idea generalizes to non-binary case by introducing “real” gradients and “fake” gradients for aggregation.

Also generalizes to “real” and “fake” features / models, which can protect sensitive ad network data.

Masked aggregation – Abstract setting

Lemma: There is a simple secret sharing protocol to compute $\langle v, w\rangle$ for $v, w \in V$, an R-module (equipped with a bilinear form $\langle \cdot,\cdot \rangle$), with three parties.

Proof: Suppose we have three parties H_1, H_2 and C, and we assume that w belongs to C, and H_1, H_2 both have v.

Now to compute $\langle v, w\rangle$, C does a secret sharing protocol with H_i and shares α_i to H_1 and β_i to H_2 with $\alpha_i \in_R$ R: $\alpha_i + \beta_i = w_i$.

The H_i can now compute $\sum \alpha_i v_i$, $\sum \beta_i v_i$ and the sum is $\langle v, w\rangle$ by bilinearity of the inner product.■

Corollary: There is a simple secret sharing protocol to compute $\sum_{i \in S} v_i$ where S is secret to one party and v_i are shared with two other parties.

Proof: Apply lemma with $w = \chi(S)$. ■

Masked LARK - Notes

Known issues and potential solutions:

- Differentially private map-reduce:
  - Aggregation keys are not hidden but are opaque (encrypted). Helpers can learn if a key was involved in aggregation.
    - Users can insert fake self-cancelling records for aggregation.
  - k-anonymity can be attacked by ad network via ballot stuffing.
    - Ballot stuffing needs ad network to know/guess rare keys.
    - DP on top of aggregation gives another layer of protection.
- Model training:
  - Feature vector is not hidden to helper.
    - Work only with dense feature vectors.
    - Users can perturb features or ad network can help by another layer of splitting of feature vectors (communication cost between helpers to sync after first layer update).
  - Label space can leak information in non-binary case.
    - Quantized-label space with randomized rounding.
  - Pollution attacks from users/browsers.
    - Solutions need more expensive crypto.
  - Timing attacks:
    - If users send model updates when activity occurs, this can leak information.
    - Model training need not be synchronous with actual conversion/non-conversion events. Hence real/fake gradients can always be reported at set times for each reporting client.
  - Solution limited to models that are continuous functions trained using SGD.
    - More general models can be trained with general secure MPC. Comes at high cost, several orders of magnitude greater than masked aggregation.