VDAF Analysis

Christopher Patton

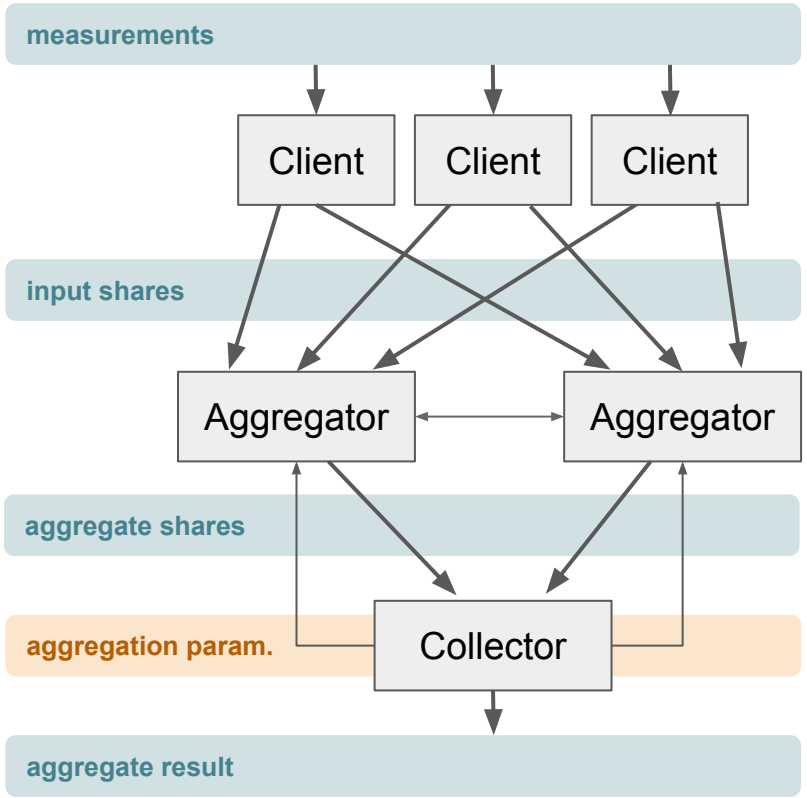
IETF 116 – UFMRG

Background

- PPM ("Privacy Preserving Measurement") WG will standardize methods of securely aggregating privacy-sensitive measurements generated client-side:
  - Probable exposure to illness (COVID-19 exposure notifications et al.)
  - Performance or threat-detection telemetry (operating system, browser, app, …)
  - Ad conversion (how many sales are attributable to a given ad campaign)
- draft-irtf-cfrg-vdaf: VDAFs ("Verifiable Distributed Aggregation Functions") are a class of lightweight **multi-party computation (MPC)** protocols that help address these use cases.
- draft-ietf-ppm-dap: DAP ("Distributed Aggregation Protocol") specifies execution of VDAFs over HTTP.

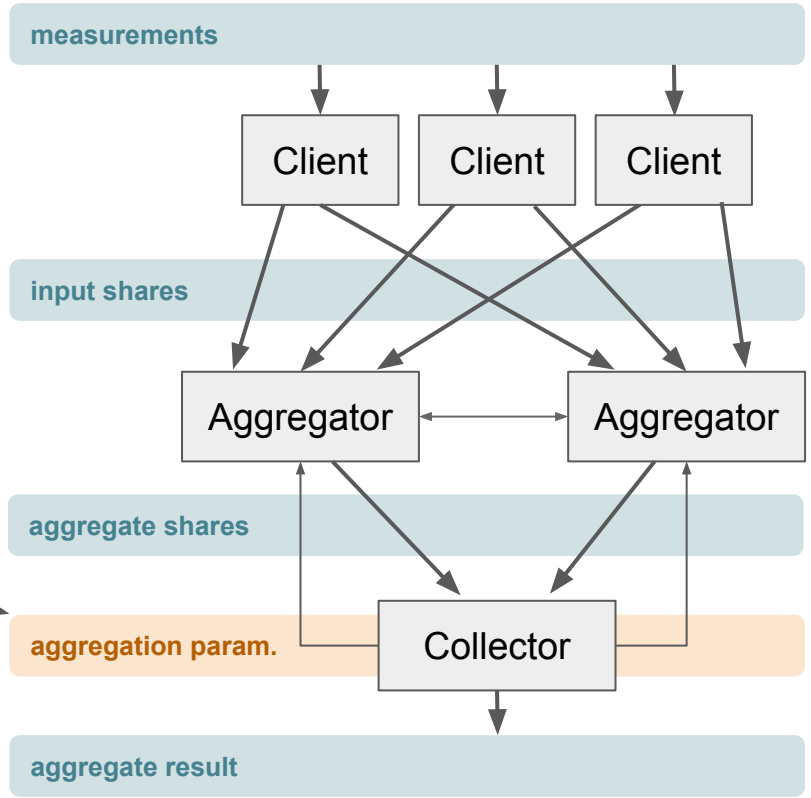
VDAF overview

- VDAF execution is distributed amongst a (large) set of **Clients**, a (small) set of **Aggregators** and a **Collector:**
  - Each client shards its **measurement** into **input shares**
  - Aggregators interact in order to prepare each set of input shares for aggregation
    - Refine them into an aggregatable form
    - Verify the refined shares correspond to a well-formed value
  - Each aggregator locally aggregates its refined shares into an **aggregate share**
  - Collector unshards the aggregate shares into the **aggregate result**.



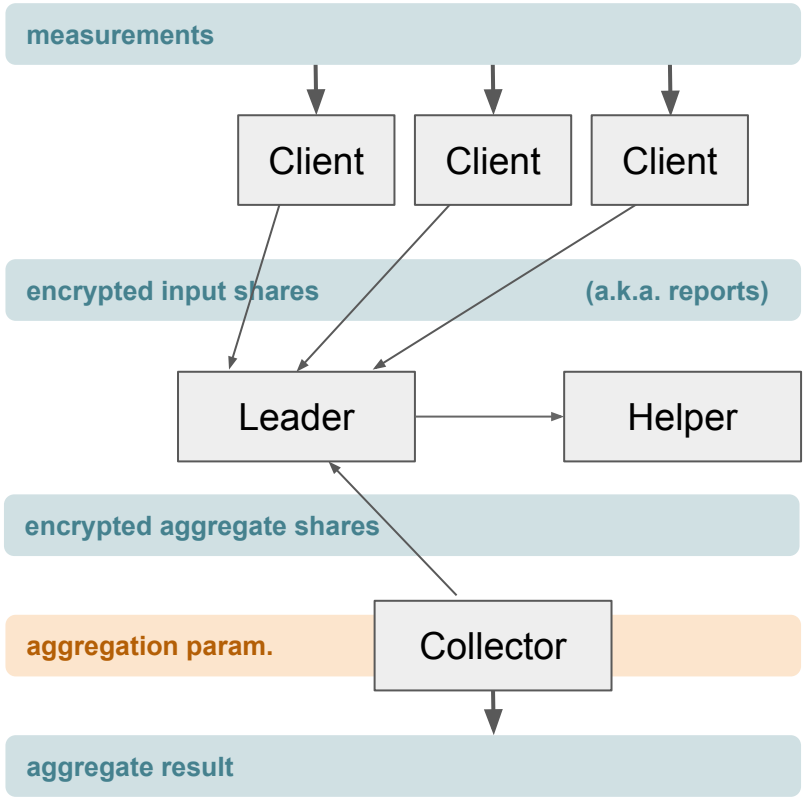
VDAF overview

- Notable features of this architecture
  - Bulk of the computation (sharding and preparation) is fully parallelizable across all measurements being aggregated
  - **Aggregation functions** $F(m_1, \ldots, m_N)$ computable by VDAFs can be decomposed into $f(g(m_1), \ldots, g(m_N))$, where f is linear and g is determined by the **aggregation parameter**
    - Not all of MPC, but a useful subset



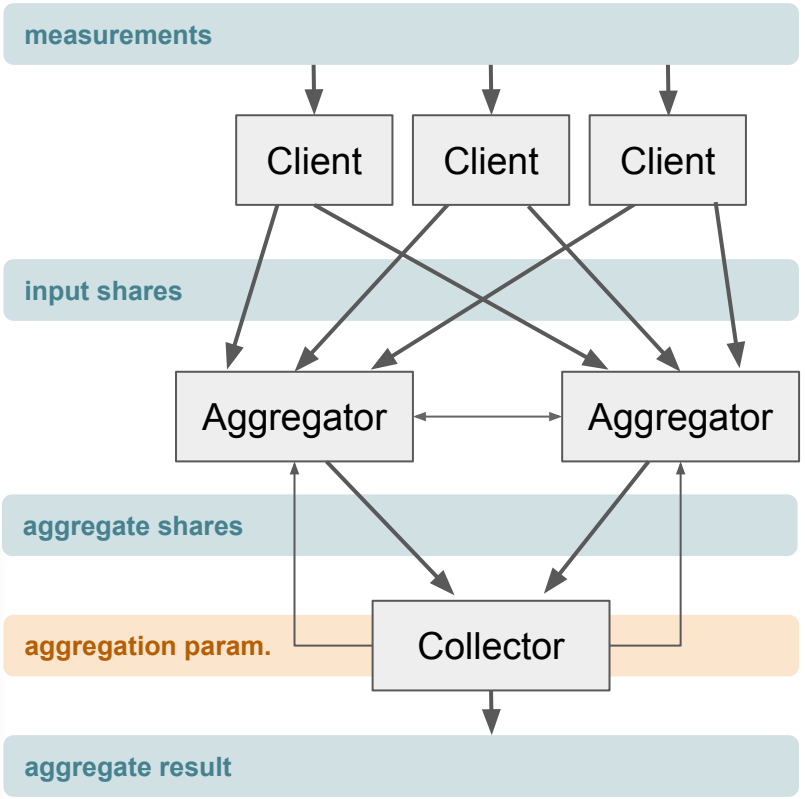
DAP overview

- VDAF execution coordinated by a designated Aggregator (the **Leader**)
  - Upload: Client push encrypted input shares (a.k.a. **reports**) to the Leader
  - Aggregate: Leader works with **Helper** to prepare and aggregate the reports
  - Collect: The Collector pulls encrypted aggregate shares from the Leader
- HTTP used for transport
- HPKE (RFC 9180) for share encryption
- "Business logic": picking the VDAF, partitioning reports into batches, additional privacy parameters, etc.



VDAF security considerations

- (Informal) security goals: **privacy** of measurements and **robustness** of the computation
  - Different trust models for privacy (one Aggregator is honest) and robustness (all Aggregators are honest)



9. Security Considerations

VDAFs have two essential security goals:

1. Privacy: An attacker that controls the network, the Collector, and a subset of Clients and Aggregators learns nothing about the measurements of honest Clients beyond what it can deduce from the aggregate result.

2. Robustness: An attacker that controls the network and a subset of Clients cannot cause the Collector to compute anything other than the aggregate of the measurements of honest Clients.

VDAF security analysis of [DPRS23]

- *Starting* goal: Formalize security considerations of draft-irtfg-cfrg-vdaf-03 (current draft at time of writing) and prove that the candidates (Prio3 and Poplar1) meet these goals
  - Model needs to fully account for how DAP uses VDAFs (issue #161):
    - Aggregators share a secret key ("VDAF verification key") for verifying each report.
    - DAP does not specify how the key is distributed. For maximum flexibility, it should be safe for a (malicious) Aggregator to pick the key unilaterally and distribute it to the Aggregators out-of-band.
- Our results:
  - Security proofs for a tweaked version of Prio3
  - Security proofs for a round-reduced variant of Poplar1 (called Doplar)

[DPRS23] H. Davis, C. Patton, M. Rosulek, P. Schoppmann. "Verifiable Distributed Aggregation Functions." In submission. ia.cr/2023/130

Security model of [DPRS23]: Robustness

$\underline{\text{Game } \mathsf{Exp}^{\text{robust}}_{\Pi}(A)}$:

$sk \leftarrow\!\!\$\ \mathcal{SK}$; $w \leftarrow$ `false`; $A^{\underline{\mathsf{Prep}}}()$; ret w

$\underline{\mathsf{Prep}}(n \in \mathcal{N}, \vec{x} \in \mathcal{X}^s, msg_{\text{Init}} \in \mathcal{M}, st_{\text{Init}} \in \mathcal{Q}_{\text{Init}})$:

if not $\Pi.\mathsf{validSt}(\text{Used}[n], st_{\text{Init}})$: ret $\perp$
$\text{Used}[n] \leftarrow \text{Used}[n] \,\|\, (st_{\text{Init}},)$
$\text{Msg}[0,1] \leftarrow msg_{\text{Init}}$
$y \leftarrow \Pi.\mathsf{refineFromShares}(st_{\text{Init}}, msg_{\text{Init}}, \vec{x})$
for $\hat{j} \in [s]$: $\text{St}[\hat{j}] \leftarrow st_{\text{Init}}$
for $\hat{\ell} \in [r+1]$:
  for $\hat{j} \in [s]$:
    $(sts, out) \leftarrow \Pi.\mathsf{Prep}(\hat{j}, sk, \text{St}[\hat{j}]$
                $n, \text{Msg}[\hat{\ell}\text{-}1, \cdot], \vec{x}[\hat{j}])$
    if $sts =$ `running`:
      $(\text{St}[\hat{j}], msg) \leftarrow out$
      $\text{Msg}[\hat{\ell}, \hat{j}] \leftarrow msg$
    else if $sts =$ `finished`:
      $y_{\hat{j}} \leftarrow out$; $\tilde{w} \leftarrow [y \notin \mathcal{V}_{st_{\text{Init}}}]$
    else if $sts =$ `failed`: pass
if not $\tilde{w}$:
  $\tilde{w} \leftarrow [y \neq \Pi.\mathsf{Unshard}(1, (\Pi.\mathsf{Agg}(y_{\hat{j}}))_{\hat{j} \in s}]$
$w \leftarrow w \vee \tilde{w}$; ret (w, Msg)

- Defined concretely in terms of a game played by a set of corrupt Clients interacting with a set of honest Aggregators.
- Adversary wins if:
  - (1) any Aggregator accepts an invalid report
  - (2) all Aggregators accept, but compute shares of the wrong value

[DPRS23] H. Davis, C. Patton, M. Rosulek, P. Schoppmann. "Verifiable Distributed Aggregation Functions." In submission. ia.cr/2023/130

Security model of [DPRS23]: Privacy

Game $\mathsf{Exp}^{\mathrm{priv}}_{\Pi,t}(A)$:

$(st_A, \mathcal{V}, (sk_{\hat{j}})_{\hat{j}\in\mathcal{V}}) \leftarrow\$ A()$
if $|\mathcal{V}| + t \neq s$ return $\bot$
$b \leftarrow\$ \{0,1\}$
$b' \leftarrow\$ A^{\underline{\mathsf{Shard}},\underline{\mathsf{Setup}},\underline{\mathsf{Prep}},\underline{\mathsf{Agg}}}(st_A)$
ret $b = b'$

$\underline{\mathsf{Shard}}(\hat{k} \in \mathbb{N}, m_0, m_1 \in \mathcal{I})$:

if $\mathrm{Used}[\hat{k}] \neq \bot$: ret $\bot$
$n \leftarrow\$ \mathcal{N}$
$(\mathrm{Pub}[\hat{k}], \mathrm{In}[\hat{k}, \cdot]) \leftarrow\$ \Pi.\mathsf{Shard}(m_b, n)$
$\mathrm{Used}[\hat{k}] \leftarrow (n, m_0, m_1)$
ret $(n, \mathrm{Pub}[\hat{k}], (\mathrm{In}[\hat{k}, \hat{j}])_{\hat{j}\in\mathcal{T}})$

$\underline{\mathsf{Setup}}(\hat{i} \in \mathbb{N}, \hat{j} \in \mathcal{V}, st_{\mathrm{Init}} \in \mathcal{Q}_{\mathrm{Init}})$:

if $\mathrm{Status}[\hat{i}, \hat{j}] \neq \bot$
  or not $\Pi.\mathsf{validSt}(\mathrm{Setup}[\cdot, \hat{j}], st_{\mathrm{Init}})$:
  ret $\bot$
$\mathrm{Setup}[\hat{i}, \hat{j}] \leftarrow st_{\mathrm{Init}}$
$\mathrm{Status}[\hat{i}, \hat{j}] \leftarrow \texttt{running}$

$\underline{\mathsf{Prep}}(\hat{i} \in \mathbb{N}, \hat{j} \in \mathcal{V}, \hat{k} \in \mathbb{N}, \vec{msg} \in \mathcal{M}^*)$:

if $\mathrm{Status}[\hat{i}, \hat{j}] \neq \texttt{running}$ or $\mathrm{In}[\hat{k}, \hat{j}] = \bot$: ret $\bot$
if $\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] = \bot$:
  $\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] \leftarrow \mathrm{Setup}[\hat{i}, \hat{j}]$; $\vec{msg} \leftarrow (\mathrm{Pub}[\hat{k}],)$
$(n, m_0, m_1) \leftarrow \mathrm{Used}[\hat{k}]$
$(sts, out) \leftarrow$
  $\Pi.\mathsf{Prep}(\hat{j}, sk_{\hat{j}}, \mathrm{St}[\hat{i}, \hat{j}, \hat{k}], n, \vec{msg}, \mathrm{In}[\hat{k}, \hat{j}])$
if $sts = \texttt{running}$:
  $(st, msg) \leftarrow out$; $\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] \leftarrow st$
else if $sts = \texttt{finished}$:
  $\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] \leftarrow \bot$; $\mathrm{Out}[\hat{i}, \hat{j}, \hat{k}] \leftarrow out$
  $\mathrm{Batch}_0[\hat{i}, \hat{j}, \hat{k}] \leftarrow m_0$; $\mathrm{Batch}_1[\hat{i}, \hat{j}, \hat{k}] \leftarrow m_1$
else if $sts = \texttt{failed}$: $\mathrm{St}[\hat{i}, \hat{j}, \hat{k}] \leftarrow \bot$
ret (sts, msg)

$\underline{\mathsf{Agg}}(\hat{i} \in \mathbb{N}, \hat{j} \in \mathcal{V})$:

if $\mathrm{Status}[\hat{i}, \hat{j}] \neq \texttt{running}$: ret $\bot$
$(st_1, \ldots, st_s) \leftarrow \mathrm{Setup}[\hat{i}, \cdot]$
if $F(st_{\hat{j}}, \mathrm{Batch}_0[\hat{i}, \hat{j}, \cdot]) \neq F(st_{\hat{j}}, \mathrm{Batch}_1[\hat{i}, \hat{j}, \cdot])$
  and $(\forall j, j' \in \mathcal{V})\ st_j = st_{j'} \wedge sk_j = sk_{j'}$:
  ret $\bot$
$\mathrm{Status}[\hat{i}, \hat{j}] \leftarrow \texttt{finished}$
ret $\Pi.\mathsf{Agg}(\mathrm{Out}[\hat{i}, \hat{j}, \cdot])$

- Defined concretely in terms of a game played by a corrupt Collector and subset of Aggregators
- Adversary mounts **chosen-batch attack:**
  - Via the Shard oracle, it chooses a pair of measurements for each honest Client
  - Game processes one of these, depending on the outcome of a coin flip
  - Adversary wins if it correctly guesses the coin flip

[DPRS23] H. Davis, C. Patton, M. Rosulek, P. Schoppmann. "Verifiable Distributed Aggregation Functions." In submission. ia.cr/2023/130

Impact of this work

- Revisions to Prio3
- Revised security considerations for DAP/VDAF documents (summarized in this thread on the CFRG mailing list) based on analysis of Prio3 and Doplar (Poplar1 variant)
    - VDAF verification key: Aggregators MUST commit to the key prior to reports being generated.
    - Nonce (accompanying each report): Clients MUST choose this at random.
    - Aggregation parameter: VDAFs MAY specify constraints on sequences of aggregation parameters used for the same report
- (Not a new observation, but bears repeating) Papers regularly leave gaps that need to be filled by corresponding specs
    - In our case: Instantiate "ideal coin-flipping functionality" presumed in prior work
    - Question for RG: **Can anyone think of examples?**

[DPRS23] H. Davis, C. Patton, M. Rosulek, P. Schoppmann. "Verifiable Distributed Aggregation Functions." In submission. ia.cr/2023/130

Aside: why games?

- Simulation paradigm is more conventional for MPC.
  - Universal Composability (UC): Define the ideal functionality for computing an aggregation function such that any VDAF that UC-realizes that functionality implies privacy and robustness.
- Consideration:
  - To guide parameter selection, it is helpful to obtain concrete security bounds for privacy and robustness *separately*.
    - Loose robustness bound may be tolerable in some applications, but a loose privacy bound is never tolerable.
  - Ideal functionality is complicated by the fact that privacy and robustness have different corruption models.
  - (Personal view of speaker): Protocols designed in the UC paradigm push details to spec authors that are not always fully addressed.

Future work

- Machine-checked proofs for VDAF candidates
  - Proof for Prio3 is pen-and-paper
  - No proof for Poplar1
- Symbolic or computational analysis of DAP. Two ways to frame this:
  - Prove that DAP preserves the security of the VDAF being executed (i.e., its usage in DAP does not result in an attack vector not accounted for in the existing security model)
  - Prove that DAP meets its own security goals, which are likely to go beyond privacy and robustness (i.e., differential privacy)

[DPRS23] H. Davis, C. Patton, M. Rosulek, P. Schoppmann. "Verifiable Distributed Aggregation Functions." In submission. ia.cr/2023/130