Verifiable Distributed Aggregation Functions

draft-patton-cfrg-vdaf-00

Presented at IETF 112 (CFRG)
Authors: Christopher Patton (speaker), Richard Barnes, Phillipp Schoppmann
Motivation

● PRIV BoF (Wednesday): Privacy preserving aggregation of user measurements
  ○ Privacy made possible by distributing the computation across multiple servers
  ○ Coordination required to ensure correctness of the computation

● Lots of recent work in the literature, but...
  ○ There is no "one-size-fits-all" solution
    ■ Each protocol is tailored to a particular (class of) aggregation functions
  ○ Protocols vary in their security and operational considerations
    ■ Lack consistent abstraction boundary for PRIV (and other standardization efforts) to build upon
Objective of this draft

● Provide an abstraction boundary (VDAF) that:
  ○ addresses the security/operational considerations of real-world deployments (ENPA, Origin Telemetry)
  ○ provides design criteria for cryptographers to build new and improved schemes

● Standardize a few VDAFs from the literature
  ○ in particular, those discussed in PRIV so far
Verifiable Distributed Aggregation Function (VDAF)

- Want to compute $A := F(p, m[1], ..., m[n])$
  - $m[1], ..., m[n]$ are the client measurements
  - $p$ is the aggregation parameter
  - $A$ is the aggregate result

- Examples:
  - $A$ is arithmetic mean
  - $A$ is a histogram estimating the distribution
  - $A$ counts how many times $p$ occurs in $m[1], ..., m[n]$
Verifiable Distributed Aggregation Function (VDAF)

- **Privacy via secret sharing**
  - **Shard**($m$)$\rightarrow$(x, x): Client shards its measurement into input shares and distributes them among the Aggregators.
  - **Prepare**($p$, x)$\rightarrow$y: Aggregator maps the aggregation parameter and its input share to its output share (e.g., DPFs).
  - **Aggregate**($y[1]$, ..., $y[n]$)$\rightarrow$Y: Aggregator combines output shares to get its aggregate share.
  - **Unshard**(Y, Y)$\rightarrow$A: Collector combines aggregate shares to get aggregate.

- **Correctness**
Verifiable Distributed Aggregation Function (VDAF)

- What about malicious (or merely misconfigured) clients?
Verifiable Distributed Aggregation Function (VDAF)

- Robustness via multi-party computation
  - Shard\((m)\rightarrow(x, x)\): Client shards its measurement into input shares and distributes them among the Aggregators.
  - Prepare: Aggregators engage in a secure MPC of \((y, y) := \text{Dist-Prepare}(p, x, x)\).
  - Aggregate\((y[1], ..., y[n])\rightarrow Y\): Aggregator combines output shares to get its aggregate share.
  - Unshard\((Y, Y)\rightarrow A\): Collector combines aggregate shares to get aggregate.

- Not general-purpose MPC!
Constructions of VDAFs

● **prio3** [CBG17, BBCG+19]
  ○ Encode each measurement $m$ as vector $x$ of elements of a finite field
  ○ Aggregation parameter: number of measurements $n$
  ○ Any aggregation function of the form $f(n, x[1] + \ldots + x[n])$
  ○ Any number of aggregators
  ○ **Dist-Prepare**: $C(x) = 0$ for arithmetic circuit $C$ that defines validity

● **hits** [BBCG+21]
  ○ Measurement: $N$-bit string (encoded as IDPF shares)
  ○ Aggregation parameter: sequence of $P$-bit strings (the "candidate prefixes") where $P \leq N$
  ○ Aggregation function: how many inputs are prefixed by each candidate
  ○ Two aggregators
  ○ **Dist-Prepare**: input is prefixed by at most one candidate

● ... and many more!
Implementations (so far)

- **Rust** [github.com/abetterinternet/libprio-rs](https://github.com/abetterinternet/libprio-rs)
  - prio3
  - hits (proof-of-concept only, missing efficient IDPF)
  - "Prio v2" (used in ENPA)
- **C++** [github.com/google/distributed_point_functions](https://github.com/google/distributed_point_functions)
  - IPDF
- **C++** [github.com/google/libprio-cc](https://github.com/google/libprio-cc)
  - "Prio v2" (used in ENPA)
- **C** [github.com/mozilla/libprio](https://github.com/mozilla/libprio)
  - "Prio v1" (used in Origin Telemetry)

<table>
<thead>
<tr>
<th>aggregation function</th>
<th>shard time</th>
<th>communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>8 μs</td>
<td>208 bytes</td>
</tr>
<tr>
<td>histogram (10 buckets)</td>
<td>15 μs</td>
<td>432 bytes</td>
</tr>
<tr>
<td>sum (32 bit integers)</td>
<td>35 μs</td>
<td>960 bytes</td>
</tr>
</tbody>
</table>
References