Secure Partitioning Protocols

Phillipp Schoppmann – IETF 116
Background: Aggregate Statistics Measurements

MPC Cluster

Clients
Background: Aggregate Statistics Measurements
Background: Aggregate Statistics Measurements
Background: Aggregate Statistics Measurements

MPC Cluster

Encrypted Reports

Clients

...
Sharding MPC Clusters

Challenge: How to partition reports across shards, s.t. all reports of the same client end up in the same shard?
Goals

- Low overhead: Blow up communication per client by a small factor
- Low round complexity
- Partitioning must not affect correctness / utility of downstream computation

Assumptions

- Bound M on the number of contributions per client
- Lots of clients (billions), few shards (thousands)
Threat Model

- Two (or more) non-colluding servers
- All parties are assumed to misbehave (as long as one server remains honest)
- Output of partitioning protocol must be *differentially private*
Threat Model
Threat Model
Threat Model
Differentially Private Views

\[ \Pr(\text{View}) \leq \exp(\epsilon) \cdot \Pr(\text{View}') + \delta \]

Client i’s data changed
Blueprint: Partitioning from Distributed OPRFs

i: index / client identifier
v: value / payload
Blueprint: Partitioning from Distributed OPRFs

i: index / client identifier
v: value / payload
Enc: Encryption scheme that allows homomorphic evaluation of PRF, e.g. ElGamal or Dodis-Yampolski
Blueprint: Partitioning from Distributed OPRFs

Client

(i, v) → Enc_2(i), Enc_{1,2}(v)

Server 1

K → Enc_2(F_K(i)), Enc_{1,2}(v)

Server 2

Map F_K(i) to partition for Enc_{1,2}(v)

Add dummies

i: index / client identifier
v: value / payload
Enc: Encryption scheme that allows homomorphic evaluation of PRF, e.g. ElGamal or Dodis-Yampolski
Dense Partitioning: OPRF Output = Shard ID

Assume there are exactly $S$ shards, and let $[S]$ be the range of $F_K$.

$F_K: \text{ID} \rightarrow [S]$

$i$: index / client identifier
$v$: value / payload

(i, v)
Dense Partitioning: OPRF Output = Shard ID

Assume there are exactly $S$ shards, and let $[S]$ be the range of $F_{k}$. Add dummies to every possible $F_{k}(i)$.

$i$: index / client identifier
$v$: value / payload
$F_{K}$: ID $\rightarrow$ $[S]$
Dense Partitioning: OPRF Output = Shard ID

Assume there are exactly \( S \) shards, and let \([S]\) be the range of \( F_K\).

\[
\begin{align*}
\text{Client} & : (i, v) \\
\text{Server 1} & : \text{Enc}_2(i), \text{Enc}_{1,2}(v) \rightarrow K \\
\text{Server 2} & : \text{Enc}_2(F_K(i)), \text{Enc}_{1,2}(v)
\end{align*}
\]

- \( i \): index / client identifier
- \( v \): value / payload
- \( F_K \): ID \( \rightarrow [S] \)

Add dummies to every possible \( F_K(i) \)

Use \( F_K(i) \) as shard ID
Dense Partitioning: Adding Dummies

M: Upper bound on the number of ciphertexts with the same index / from the same client
S: Number of shards
TSDLap(\(\lambda\), t): Truncated, shifted, discrete Laplace distribution with mean t and scale \(\lambda\)

Expected #dummies per bucket for \(\varepsilon = 0.5\) and \(\delta = 10^{-11}\): 49M per server
Sparse Partitioning: OPRF Output = Random Client ID

- If the OPRF codomain is large enough to make collisions unlikely, we can use the OPRF outputs as a pseudorandom client identifier.
- Allows *local* per-client aggregation (e.g., using Homomorphic Encryption)

\[(i, v) \quad K\]

\[\text{i: index / client identifier}\]
\[\text{v: value / payload}\]
\[\text{F_K: ID -> } \{0,1\}^\sigma\]
Sparse Partitioning: OPRF Output = Random Client ID

- If the OPRF codomain is large enough to make collisions unlikely, we can use the OPRF outputs as a pseudorandom client identifier.
- Allows *local* per-client aggregation (e.g., using Homomorphic Encryption)

\[
\text{Enc}_2(i), \text{Enc}_{1,2}(v) \quad \text{K}
\]

- **i**: index / client identifier
- **v**: value / payload
- **F_K**: ID -> \{0,1\}^\sigma
Sparse Partitioning: OPRF Output = Random Client ID

- If the OPRF codomain is large enough to make collisions unlikely, we can use the OPRF outputs as a pseudorandom client identifier.
- Allows *local* per-client aggregation (e.g., using Homomorphic Encryption)

\[
\text{Enc}_2(i), \quad \text{Enc}_{1,2}(v) \quad (i, v) \quad \text{Enc}_2(F_K(i)), \quad \text{Enc}_{1,2}(v) \quad K
\]

- Map \(F_K(i)\) to partition for \(\text{Enc}_{1,2}(v)\);
- Perform local aggregation

\(i\): index / client identifier
\(v\): value / payload
\(F_K: \text{ID} \to \{0,1\}^\sigma\)
Assigning Ciphertexts to Shards

$N'$: Number of ciphertexts after adding dummies
$M'$: Upper bound on the number of ciphertexts with the same index
$S$: Number of shards

Observation: As long as $M' \ll \lceil N'/S \rceil$, the overhead will be small in practice.
But: $N'$ might still be significantly larger than $N$. For $\varepsilon = 0.5$ and $\delta = 10^{-11}$, $N'/N = 1.1$
Assigning Ciphertexts to Shards

N’: Number of ciphertexts after adding dummies
M’: Upper bound on the number of ciphertexts with the same index
S: Number of shards

Observation: As long as $M' \ll \lceil N'/S \rceil$, the overhead will be small in practice. But: N’ might still be significantly larger than N. For $\varepsilon = 0.5$ and $\delta = 10^{-11}$, $N'/N = 1.1$
Assigning Ciphertexts to Shards

\( N' \): Number of ciphertexts after adding dummies
\( M' \): Upper bound on the number of ciphertexts with the same index
\( S \): Number of shards

Observation: As long as \( M' \ll \lceil \frac{N'}{S} \rceil \), the overhead will be small in practice.
But: \( N' \) might still be significantly larger than \( N \). For \( \epsilon = 0.5 \) and \( \delta = 10^{-11} \), \( N'/N = 1.1 \)
Assigning Ciphertexts to Shards

$N'$: Number of ciphertexts after adding dummies
$M'$: Upper bound on the number of ciphertexts with the same index
$S$: Number of shards

Observation: As long as $M' \ll \lceil N'/S \rceil$, the overhead will be small in practice.
But: $N'$ might still be significantly larger than $N$. For $\varepsilon = 0.5$ and $\delta = 10^{-11}$, $N'/N = 1.1$
Assigning Ciphertexts to Shards

N': Number of ciphertexts after adding dummies
M': Upper bound on the number of ciphertexts with the same index
S: Number of shards

Observation: As long as \( M' \ll \lceil N'/S \rceil \), the overhead will be small in practice.
But: \( N' \) might still be significantly larger than \( N \). For \( \varepsilon = 0.5 \) and \( \delta = 10^{-11} \), \( N'/N = 1.1 \)
Conclusion

- Distributed OPRFs allow for efficient sharding protocols.
- When the number of shards is much smaller than the number of clients, the overhead is negligible.
- For a slightly larger (10%) overhead, we can enable local aggregation at one of the servers. Example application: Sparse histogram computation [1].

Next Steps

- General interest from the working group in secure partitioning?
- Other protocols or settings where this might be useful?
- Do we need additional properties (e.g., keep the order of inputs)?