Secure Partitioning Protocols

Phillipp Schoppmann – IETF 116
Background: Aggregate Statistics Measurements

MPC Cluster

Clients
Background: Aggregate Statistics Measurements

MPC Cluster

Encrypted Reports

Clients
Background: Aggregate Statistics Measurements

MPC Cluster

Encrypted Reports

Clients
Background: Aggregate Statistics Measurements

MPC Cluster

Encrypted Reports

Clients

Aggregate Result
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

- Inputs from the same client end up in the same shard
- Low communication overhead and 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)
Blueprint: Partitioning from Distributed OPRFs

Client

(i, v)

Server 1

K

Server 2

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

Client
(i, v)

Server 1

Server 2

Enc_{2}(i),
Enc_{1,2}(v)

K

Add dummies

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

(i, v)

Client

\[ Enc_2(i), Enc_{1,2}(v) \]

Server 1

K

Add dummies

\[ Enc_2(F_K(i)), Enc_{1,2}(v) \]

Server 2

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

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 $F_K$.

Client: ($i, v$)

Server 1: $K$

Server 2: 

$i$: 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 $F_K$.

- $i$: index / client identifier
- $v$: value / payload
- $K$: ID -> $[S]$
Dense Partitioning: OPRF Output = Shard ID

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

- $Enc_2(i)$
- $Enc_1,2(v)$
- $F_K(i)$
- $Enc_2(F_K(i))$
- $Enc_1,2(v)$

Client

Server 1

Server 2

(i, v)

K

Add dummies to every possible $F_K(i)$

Use $F_K(i)$ as shard ID

i: index / client identifier
v: value / payload
$F_K$: ID -> $[S]$
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)  
K

i: index / client identifier  
v: value / payload  
F_K: ID -> \{0,1\}^g
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)

![Diagram showing mobile device sending Enc2(i), Enc1,2(v) to server]

i: index / client identifier
v: value / payload

F_K: ID -> {0,1}^g

Obliviously add dummies to histogram
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)

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

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

\( i \): index / client identifier
\( v \): value / payload
\( F_K : \text{ID} \to \{0,1\}^q \)
Sparse Partitioning Protocol

\[ \text{val}_i = [\text{val}_i^1] + [\text{val}_i^2] + [\text{val}_i^3] \]

\[ \text{Enc}_{K2}(H(\text{ind}_i)^{K1'}) \]
\[ \text{Enc}_{K2} (\text{Enc}_{K1}([\text{val}_i^1]), \text{Enc}_{K2}([\text{val}_i^2]), \text{Enc}_{K3}([\text{val}_i^3])) \]

\[ \text{ind}'_i = H(\text{ind}_i)^{K1'} \]

Add dummies and rerandomize

Assign ciphertexts to shards s.t. all distinct ind' end up in the same shard
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 $\varepsilon = 0.5$ 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 \( \epsilon = 0.5 \) 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 $M’ \ll \lceil \frac{N’}{S} \rceil$, 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 $\delta = 10^{-11}$, $N'/N = 1.1$
How to make sparse histogram private without seeing it?

Server 1 can add dummy contributions in two ways:

- **duplicate existing clients’ indices, replacing values with Enc(0)**
  - Useful to hide the exact count of indices that are **common**
- **add new fake indices with value 0**
  - Useful to hide the exact count of indices that are **rare**

Our approach:

1. Choose Threshold $T$
2. For each multiplicity $i < T$, add $i$ fake indices $\sim$Laplace times
3. Duplicate each ciphertext $\sim$NegativeBinomial times
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)?