Lightweight Techniques for Private Heavy Hitters

Dan Boneh  
*Stanford*

Elette Boyle  
*IDC Herzliya*

Henry Corrigan-Gibbs  
*MIT CSAIL*

Niv Gilboa  
*Ben-Gurion University*

Yuval Ishai  
*Technion*

IEEE Security & Privacy 2021
Mozilla wants to know…
which URLs most often crash the browser?

- stanford.edu/images/logo.png
- nytimes.com/index.html
- google.com/search?q=prostate+cancer
- nytimes.com/index.html
Today: Non-private data collection

- stanford.edu/images/logo.png
- nytimes.com/index.html
- google.com/search?q=prostate+cancer
- nytimes.com/index.html
We show: Mozilla can learn the most-often reported URLs, without learning which client reported which URL.

- stanford.edu/images/logo.png
- nytimes.com/index.html
- google.com/search?q=prostate+cancer
- ...
- nytimes.com/index.html
We show: Mozilla can learn the most-often reported URLs, without learning which client reported which URL.

The URLs that caused $\geq 1000$ crashes are:
- nytimes.com/2020/03/27/dogs.html
- about.twitter.com/logo.png
- ...

The URLs include:
- stanford.edu/images/logo.png
- nytimes.com/index.html
- google.com/search?q=prostate+cancer
Private heavy-hitters problem
In setting of local differential privacy: [Bassily, Smith 2015] [Qin, Yan, Yu, Khalil, Xiao, Ren 2016] [Bassily, Nissim, Stemmer, Guha 2017] [Bun, Nelson, Stemmer 2019] ...

Millions of clients
Each client holds an $n$-bit string (e.g., $n \approx 256$)

Two data-collection servers
Should learn the set of strings that $\geq t$ clients hold
Private heavy-hitters problem

In setting of local differential privacy: [Bassily, Smith 2015] [Qin, Yan, Yu, Khalil, Xiao, Ren 2016] [Bassily, Nissim, Stemmer, Guha 2017] [Bun, Nelson, Stemmer 2019] ...

Millions of clients
Each client holds an $n$-bit string
(e.g., $n \approx 256$)

Privacy* against one malicious server, colluding with malicious clients

Two data-collection servers
Should learn the set of strings that $\geq t$ clients hold
Private heavy-hitters problem

In setting of local differential privacy: [Bassily, Smith 2015] [Qin, Yan, Yu, Khalil, Xiao, Ren 2016] [Bassily, Nissim, Stemmer, Guha 2017] [Bun, Nelson, Stemmer 2019] …

Millions of clients
Each client holds an $n$-bit string (e.g., $n \approx 256$)

Two data-collection servers
Should learn the set of strings that $\geq t$ clients hold

Correctness against malicious clients.
Private heavy-hitters problem

In setting of local differential privacy: [Bassily, Smith 2015] [Qin, Yan, Yu, Khalil, Xiao, Ren 2016] [Bassily, Nissim, Stemmer, Guha 2017] [Bun, Nelson, Stemmer 2019] ...

Each client holds an $n$-bit string (e.g., $n \approx 256$)

Minimal communication and computation costs

Support 100s of submissions per second (Using 100-1000x less bandwidth than general-purpose MPC)
Applications

• Which URLs crash Firefox most often?
• Which phone apps consume the most battery life?
• Which passwords are most popular?
• Which programs consume the most CPU?
• Where do users of my app spent their time?
• …
This talk

• The private heavy-hitters problem
• New tools
  – Incremental distributed point functions
  – Malicious-secure sketching
• Evaluation
This talk

• The private heavy-hitters problem
• New tools
  – Incremental distributed point functions
  – Malicious-secure sketching
• Evaluation
Private heavy hitters: A warm-up scheme

1. Client $i$ with string $s_i$ prepares a binary tree, with 1s on the path to the $s_i$-th leaf of the tree.

Client $i$

String $s_i = 011$
Private heavy hitters: A warm-up scheme

2. Each client secret-shares the labels on the tree’s nodes and sends one share to each of the servers.
Private heavy hitters: A warm-up scheme

2. Each client secret-shares the node labels and sends one share to each of the servers.

→ Single message from client to servers
Private heavy hitters: A warm-up scheme

2. Each client secret-shares the node labels and sends one share to each of the servers.

→ Single message from client to servers
Private heavy hitters: A warm-up scheme

2. Each client secret-shares the node labels and sends one share to each of the servers.

→ Single message from client to servers
Private heavy hitters: A warm-up scheme

2. Each client secret-shares the node labels and sends one share to each of the servers.

→ Single message from client to servers
Private heavy hitters: A warm-up scheme

2. Each client secret-shares the node labels and sends one share to each of the servers.

→ Single message from client to servers
Private heavy hitters: A warm-up scheme

3. Servers sum up shares from each client to get “aggregate” shares.
Private heavy hitters: A warm-up scheme

4. Servers publish shares to perform BFS search for heavy hitters.

Threshold \( t = 3 \)
Private heavy hitters: A warm-up scheme

4. Servers publish shares to perform BFS search for heavy hitters.

Threshold $t = 3$
Private heavy hitters: A warm-up scheme

4. Servers publish shares to perform BFS search for heavy hitters.
Private heavy hitters: A warm-up scheme

4. Servers use BFS with pruning to find all heavy hitters.

Threshold $t = 3$
Private heavy hitters: A warm-up scheme

4. Servers use BFS with pruning to find all heavy hitters.
Private heavy hitters: A warm-up scheme

4. Servers use BFS with pruning to find all heavy hitters.

Threshold $t = 3$

Heavy hitter: 101
(string is path to heavy leaf)
Warm-up scheme: Properties

**Correctness**
Servers find exactly the set of heavy hitters (no error).

**Privacy**
If one server is honest, servers learn only the set of heavy “prefixes”

Are we done?
Technical challenges

1. Each tree is exponentially large ⇒ Client cannot materialize it

   **Idea:** Incremental distributed point functions.
   \[ \rightarrow \text{Succinct secret sharing of a tree with one non-zero path} \]
   \[ \rightarrow \text{Communication } O(\lambda n) \text{ instead of } O(\lambda n^2) \] [with normal DPF]

2. Client can send malformed secret shares ⇒ Data corruption

   **Idea:** Malicious-secure sketching.
   \[ \rightarrow \text{Servers can test whether a secret-shared vector is non-zero in a single coordinate.} \]
   \[ \rightarrow \text{No interaction with client, } O(\lambda) \text{ comm b/w servers.} \]
   + Extractable distributed point functions (see paper)
Technical challenges

1. Each tree is exponentially large ⇒ Client cannot materialize it

   **Idea:** Incremental distributed point functions.
   → Succinct secret sharing of a tree with one non-zero path
   → Communication $O(\lambda n)$ instead of $O(\lambda n^2)$ [with normal DPF]

2. Client can send malformed secret shares ⇒ Data corruption

   **Idea:** Malicious-secure sketching.
   → Servers can test whether a secret-shared vector is non-zero in a single coordinate.
   → No interaction with client, $O(\lambda)$ comm b/w servers.
   + Extractable distributed point functions (see paper)
Technical challenges

1. Each tree is exponentially large ⇒ Client cannot materialize it

   **Idea:** Incremental distributed point functions.
   → Succinct secret sharing of a tree with one non-zero path
   → Communication $O(\lambda n)$ instead of $O(\lambda n^2)$ [with normal DPF]

2. Client can send malformed secret shares ⇒ Data corruption

   **Idea:** Malicious-secure sketching.
   → Servers can test whether a secret-shared vector is non-zero in a single coordinate.
   → No interaction with client, $O(\lambda)$ comm b/w servers.
   + Extractable distributed point functions (see paper)
Contribution 1:
Incremental distributed point functions (DPFs)

Each client secret-shares the labels on a tree with one non-zero path and sends one share to each server. Communication $\approx 2^n$ ☹️
Contribution 1:
Incremental distributed point functions (DPFs)

With incremental DPFs, client only sends each server a short key.

For a tree of depth $n$, and security parameter $\lambda \approx 128$, the keys have size $O(\lambda n)$. 😊

Using standard DPFs would give keys of size $O(\lambda n^2)$. 

Client $i$  

String $i = 011 \in \{0,1\}^n$  

KeyGen
Contribution 1:

Incremental distributed point functions (DPFs)

With incremental DPFs, client only sends each server a short key.

For a tree of depth $n$, and security parameter $\lambda \approx 128$, the keys have size $O(\lambda n)$. 😊

Using standard DPFs would give keys of size $O(\lambda n^2)$.

Client $i$

String $i = 011 \in \{0,1\}^n$
Contribution 1:
Incremental distributed point functions (DPFs)

With incremental DPFs, client only sends each server a short key.

For a tree of depth $n$, and security parameter $\lambda \approx 128$, the keys have size $O(\lambda n)$. 😊

Using standard DPFs would give keys of size $O(\lambda n^2)$.

Client $i$  
String $i = 011 \in \{0,1\}^n$
Contribution 1:
Incremental distributed point functions (DPFs)

The servers expand the key into shares of a tree, one node at a time. Evaluating the keys on a path takes $O(n)$ AES ops. Standard DPFs would require $O(n^2)$ AES ops.
Contribution 1:
Incremental distributed point functions (DPFs)

The servers expand the key into shares of a tree, one node at a time.

Evaluating the keys on a path takes $O(n)$ AES ops.

Standard DPFs would require $O(n^2)$ AES ops.
Contribution 1:
Incremental distributed point functions (DPFs)

The servers expand the key into shares of a tree, one node at a time.

Evaluating the keys on a path takes $O(n)$ AES ops.

Standard DPFs would require $O(n^2)$ AES ops.
Contribution 1:
Incremental distributed point functions (DPFs)

The servers expand the key into shares of a tree, one node at a time.

Evaluating the keys on a path takes $O(n)$ AES ops.

Standard DPFs would require $O(n^2)$ AES ops.
Contribution 1:
Incremental distributed point functions (DPFs)

The servers expand the key into shares of a tree, one node at a time.

Evaluating the keys on a path takes $O(n)$ AES ops.

Standard DPFs would require $O(n^2)$ AES ops.
Contribution 1:
Incremental distributed point functions (DPFs)

The servers expand the key into shares of a tree, one node at a time

Evaluating the keys on a path takes $O(n)$ AES ops.

Standard DPFs would require $O(n^2)$ AES ops.
Contribution 1: Incremental distributed point functions (DPFs)

The servers expand the key into shares of a tree, one node at a time.

Evaluating the keys on a path takes $O(n)$ AES ops.

Standard DPFs would require $O(n^2)$ AES ops.
Contribution 1:
Incremental distributed point functions (DPFs)

The servers expand the key into shares of a tree, one node at a time.

Evaluating the keys on a path takes $O(n)$ AES ops.

Standard DPFs would require $O(n^2)$ AES ops.
Contribution 1: Incremental distributed point functions (DPFs)

The servers expand the key into shares of a tree, one node at a time.

Evaluating the keys on a path takes $O(n)$ AES ops.

Standard DPFs would require $O(n^2)$ AES ops.
Contribution 1:
Incremental distributed point functions (DPFs)

The servers expand the key into shares of a tree, one node at a time.

Evaluating the keys on a path takes $O(n)$ AES ops.

Standard DPFs would require $O(n^2)$ AES ops.
Contribution 1: Incremental distributed point functions (DPFs)

The servers expand the key into shares of a tree, one node at a time.

Evaluating the keys on a path takes $O(n)$ AES ops.

Standard DPFs would require $O(n^2)$ AES ops.
Contribution 1:
Incremental distributed point functions (DPFs)

Construction
• Our incremental DPF builds on the DPF of BGI16
• Just requires symmetric-key operations (PRG/AES)
• The BGI16 DPF already uses a tree structure internally
  – Our construction just exposes this structure explicitly
Technical challenges

1. Each tree is exponentially large ⇒ Client cannot materialize it
   
   Idea: Incremental distributed point functions.
   → Succinct secret sharing of a tree with one non-zero path
   → Communication $O(\lambda n)$ instead of $O(\lambda n^2)$ [with normal DPF]

2. Client can send malformed secret shares ⇒ Data corruption
   
   Idea: Malicious-secure sketching.
   → Servers can test whether a secret-shared vector is non-zero in a single coordinate.
   → No interaction with client, $O(\lambda)$ comm b/w servers.

   + Extractable distributed point functions (see paper)
Contribution 2: 

Malicious-secure sketching

Client can send shares of garbage/random values

Servers cannot detect this!
Contribution 2: Malicious-secure sketching

Client can send shares of garbage/random values

Servers cannot detect this!
Contribution 2:
Malicious-secure sketching

Honest clients send shares of a tree that has a single “1” at each level
(Each client can only vote for one string)
Contribution 2: Malicious-secure sketching

Honest clients send shares of a tree that has a single “1” at each level (Each client can only vote for one string)

Idea: The servers run a protocol on each client’s key at each layer to check that this invariant holds.

→ Protocol checks that servers hold a secret sharing of a vector of Hamming weight one.
Contribution 2:
Malicious-secure sketching

Honest clients send shares of a tree that has a single “1” at each level (Each client can only vote for one string)

Idea: The servers run a protocol on each client’s key at each layer to check that this invariant holds.

→ Protocol checks that servers hold a secret sharing of a vector of Hamming weight one.
Contribution 2:

Malicious-secure sketching

Honest clients send shares of a tree that has a single “1” at each level
(Each client can only vote for one string)

Idea: The servers run a protocol on each client’s key at each layer to check that this invariant holds.

→ Protocol checks that servers hold a secret sharing of a vector of Hamming weight one.
Contribution 2: Malicious-secure sketching

Honest clients send shares of a tree that has a single “1” at each level (Each client can only vote for one string)

Idea: The servers run a protocol on each client’s key at each layer to check that this invariant holds.

→ Protocol checks that servers hold a secret sharing of a vector of Hamming weight one.
Contribution 2:
Malicious-secure sketching

Honest clients send shares of a tree that has a single “1” at each level
(Each client can only vote for one string)

Idea: The servers run a protocol on each client’s key at each layer to check that this invariant holds.

→ Protocol checks that servers hold a secret sharing of a vector of Hamming weight one.
Contribution 2:
Malicious-secure sketching

Honest clients send shares of a tree that has a single “1” at each level (Each client can only vote for one string)

Idea: The servers run a protocol on each client’s key at each layer to check that this invariant holds.

→ Protocol checks that servers hold a secret sharing of a vector of Hamming weight one.
Contribution 2:

Malicious-secure sketching

Honest clients send shares of a tree that has a single “1” at each level
(Each client can only vote for one string)

Idea: The servers run a protocol on each client’s key at each layer to check that this invariant holds.

→ Protocol checks that servers hold a secret sharing of a vector of Hamming weight one.
Contribution 2:
Malicious-secure sketching

Honest clients send shares of a tree that has a single “1” at each level (Each client can only vote for one string)

Idea: The servers run a protocol on each client’s key at each layer to check that this invariant holds.

→ Protocol checks that servers hold a secret sharing of a vector of Hamming weight one.
Contribution 2:
Malicious-secure sketching

Honest clients send shares of a tree that has a single “1” at each level
(Each client can only vote for one string)

Idea: The servers run a protocol on each client’s key at each layer to check that this invariant holds.

→ Protocol checks that servers hold a secret sharing of a vector of Hamming weight one.
Contribution 2:
Malicious-secure sketching

Prior work allows testing whether shared vector has Hamming weight 1
– with security against semi-honest servers [BGI16]
– when servers can interact with the client [BBCGI19,ECZB19]
– with additional non-colluding servers [CBM15,APY20]

Our technique has none of these limitations.

Idea:
• Convert semi-honest-secure scheme [BGI16] into malicious-secure one.
• To do so, we use “algebraic manipulation detection” codes [CDFP08]
Technical challenges

1. Each tree is exponentially large ⇒ Client cannot materialize it
   
   **Idea:** Incremental distributed point functions.
   → Succinct secret sharing of a tree with one non-zero path
   → Communication $O(\lambda n)$ instead of $O(\lambda n^2)$ [with normal DPF]

2. Client can send malformed secret shares ⇒ Data corruption
   
   **Idea:** Malicious-secure sketching.
   → Servers can test whether a secret-shared vector is non-zero in a single coordinate.
   → No interaction with client, $O(\lambda)$ comm b/w servers.
   + Extractable distributed point functions (see paper)
This talk

• The private heavy-hitters problem
• New tools
  – Incremental distributed point functions
  – Malicious-secure sketching
• Evaluation
This talk

• The private heavy-hitters problem

• New tools
  – Incremental distributed point functions
  – Malicious-secure sketching

• Evaluation
Implementation

Roughly 3,500 lines of Rust
- Our open-source implementation:
  github.com/henrycg/heavyhitters
- Google’s C++ implementation of incremental DPF:
  github.com/google/distributed_point_functions

Experimental setup
• Servers on opposite sides of U.S.
  - Amazon EC2 us-east-1 (VA) and us-west-1 (CA)
• Simulated clients in us-east-1
• Each server is one c4.8xlarge (36 vCPU, 60 GiB RAM)
Incremental DPFs save computation

The diagram illustrates the comparison of client time (seconds) against client string length for different DPF types:

- **Succinct sketches** [MDC15]
- **Standard DPF**
- **Incremental DPF** (this work)

As the client string length increases from 128 to 512, the client time for **Succinct sketches** and **Standard DPF** shows a notable increase, whereas the **Incremental DPF** demonstrates a more modest rise or even slight decrease, indicating better performance.

**Key Points**:
- **Innovative Computation Reduction**: Incremental DPFs offer significant computational savings compared to standard DPFs.
- **Efficiency Metric**: Client time is a strong indicator of efficiency, with Incremental DPFs showing superior efficiency across different string lengths.
Incremental DPFs save communication

Communication cost (per client)

- Standard DPF
- Incremental DPF (this work)

Client string length

- 0B
- 2MiB
- 4MiB
- 6MiB
- 8MiB
- 10MiB

- 128
- 256
- 384
- 512
Total cost is manageable for latency-tolerant applications
Searching for top-900 heavy hitters, 256-bit strings
(Strings sampled from Zipf distribution with parameter 1.03 and support 10k. Two c4-8xlarge communicating over WAN.)

<table>
<thead>
<tr>
<th>Clients</th>
<th>Computation</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>100k</td>
<td>13.8 mins</td>
<td>6.5 GB</td>
</tr>
<tr>
<td>200k</td>
<td>27.2 mins</td>
<td>13.1 GB</td>
</tr>
<tr>
<td>400k</td>
<td>53.8 mins</td>
<td>26.2 GB</td>
</tr>
</tbody>
</table>

Completely parallelizable
Lightweight Techniques for Private Heavy Hitters

With 400,000 clients, server-side computation takes less than one hour over WAN.

Privacy against malicious server, correctness against malicious clients → MPC-style privacy guarantee (not local differential privacy)

New techniques
• More powerful distributed point functions: incremental & extractable (see paper)
• Tools for malicious security in systems using secret sharing
• Application to other private data-collection problems (see paper)

Paper: https://eprint.iacr.org/2021/017
Code: https://github.com/henrycg/heavyhitters