# Lightweight Techniques for Private Heavy Hitters

## IEEE Security & Privacy 2021

**COM-506 Student seminar: security protocols and applications**

**12 May 2025**

**Tapdig Maharramli, Raymond Nasr**

https://eprint.iacr.org/2021/017.pdf

# Contents



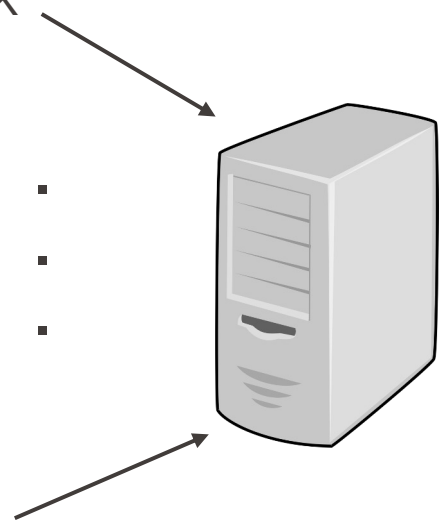
- Problem Statement
- Lightweight Techniques
  - Incremental Distributed Point Functions
  - Malicious-Secure Sketching
  - Extractable Distributed Point Functions
- Poplar System
- Providing Differential Privacy
- Implementation and Evaluation
- Summary

# Browser vendor wants to find out which URLs crash browser most often…

🍪 google.com/search?q=how+to+prove+I+didn't+eat+last+cookie+using+ZK
🔐 lasec.epfl.ch/teaching.php
🐟 nytimes.com/2025/05/05/climate/sustainable-fish-seafood.html

▪
▪
▪

🎓 moodle.epfl.ch/course/view.php?id=13965
📰 www.swissinfo.ch/eng



**Non-private data collection!**

▪

# Browser vendor wants to find out which URLs crash browser most often…

🍪 google.com/search?q=how+to+prove+I+didn't+eat+last+cookie+using+ZK
🔐 lasec.epfl.ch/teaching.php
🐟 nytimes.com/2025/05/05/climate/sustainable-fish-seafood.html

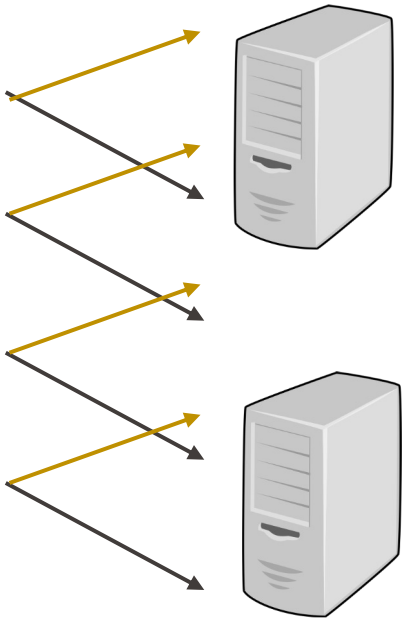
▪

▪

▪

🎓 moodle.epfl.ch/course/view.php?id=13965

www.swissinfo.ch/eng



**This paper: Browser vendor can learn most often reported URLs without learning which client reported which URL.**

# Private Heavy-Hitters Problem

- $C$ clients (mobile devices, browsers, etc.)
- Each client $i, for\ i \in \{1, \ldots, C\}$ holds an n-bit string $\alpha_i \in \{0,1\}^n\ (e.g., n \approx 256)$
- Two non-colluding data-collection servers
- **Goal:** Servers learn set of all strings that $\geq t$ clients hold without revealing individual client data

# Private Heavy-Hitters Problem

- Privacy against one malicious server, colluding with malicious clients
- Correctness against malicious clients
- Minimal communication and computation costs
- Support 100s of submissions per second (using 100-1000x less bandwidth than general-purpose MPC)

# Poplar: Security Properties

- **Completeness**: With honest participants, servers correctly learn the aggregate statistics.
- **Robustness to malicious clients**: Malicious clients cannot bias results beyond choosing their own input arbitrarily.
- **Privacy against a malicious server**: If one server is honest, a malicious server learns nothing about client data beyond the aggregate statistics.

# Other Applications

Any scenario requiring private data collection and analytics

- 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?
- Mobile companies learning popular apps without tracking individuals
- Electric vehicle makers finding roads where batteries run low

# Lightweight Techniques

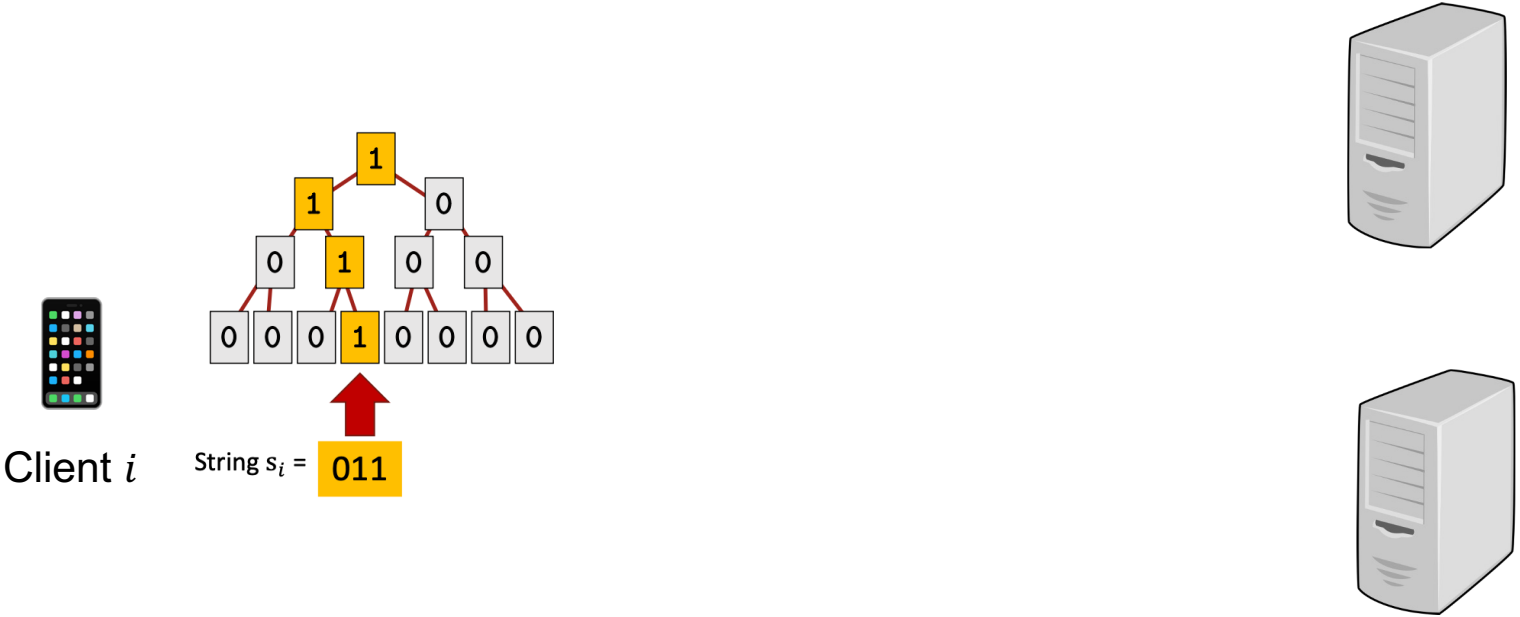
→ Incremental DPFs

→ Malicious-Secure Sketching

→ Extractable DPFs

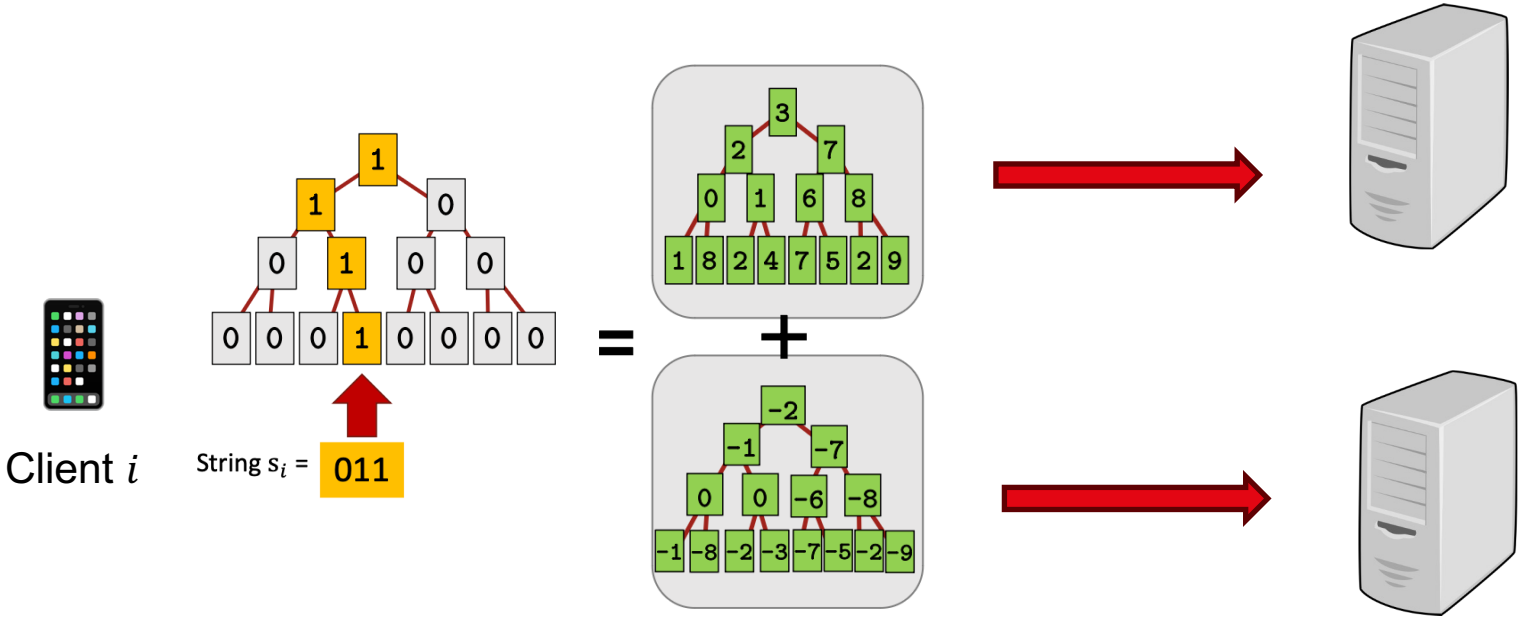
# Warm-up Scheme

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



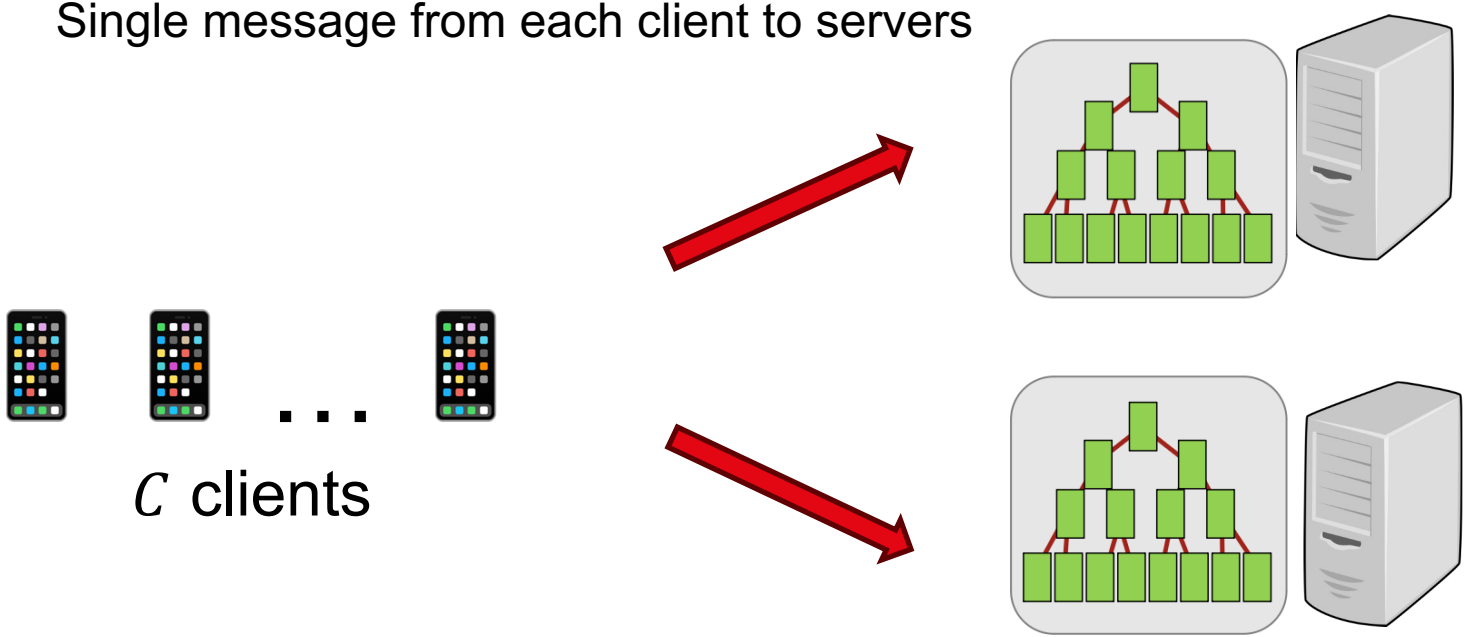
# Warm-up Scheme

- Each client secret-shares the labels on the tree's nodes and sends one share to each of the servers



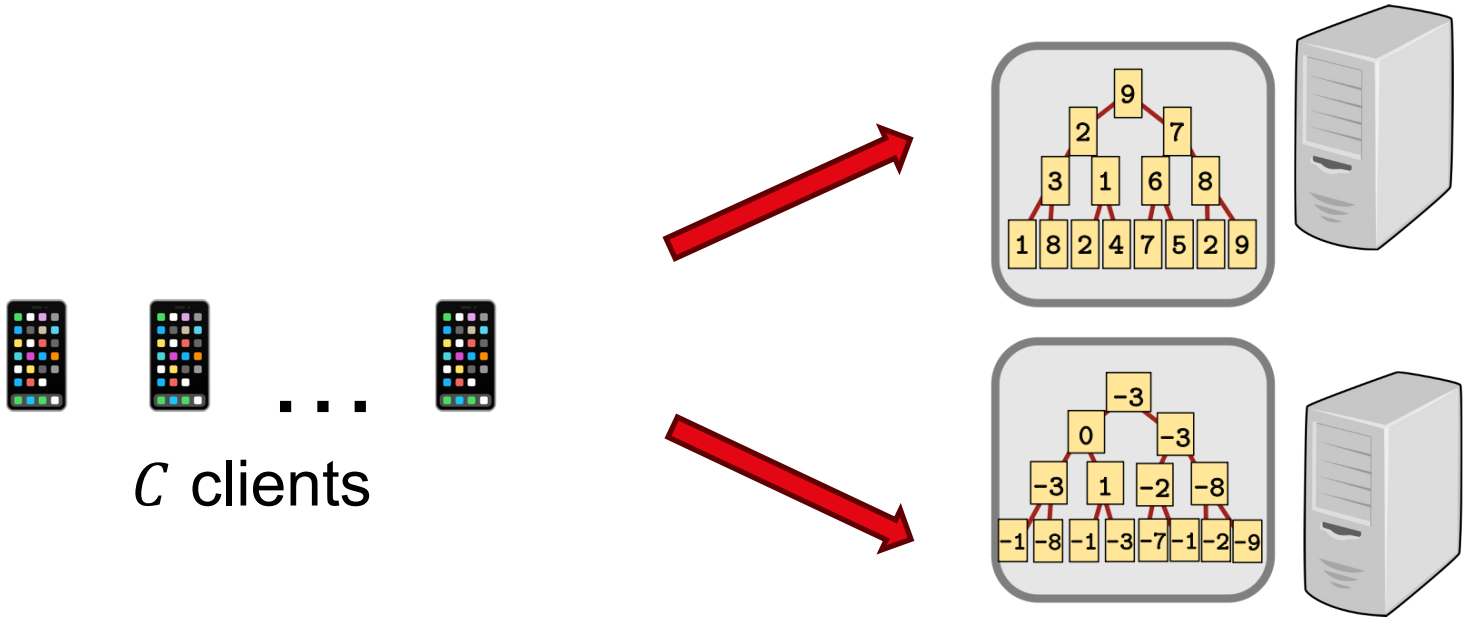
# Warm-up Scheme

- Each client secret-shares the labels on the tree's nodes and sends one share to each of the servers
- Single message from each client to servers



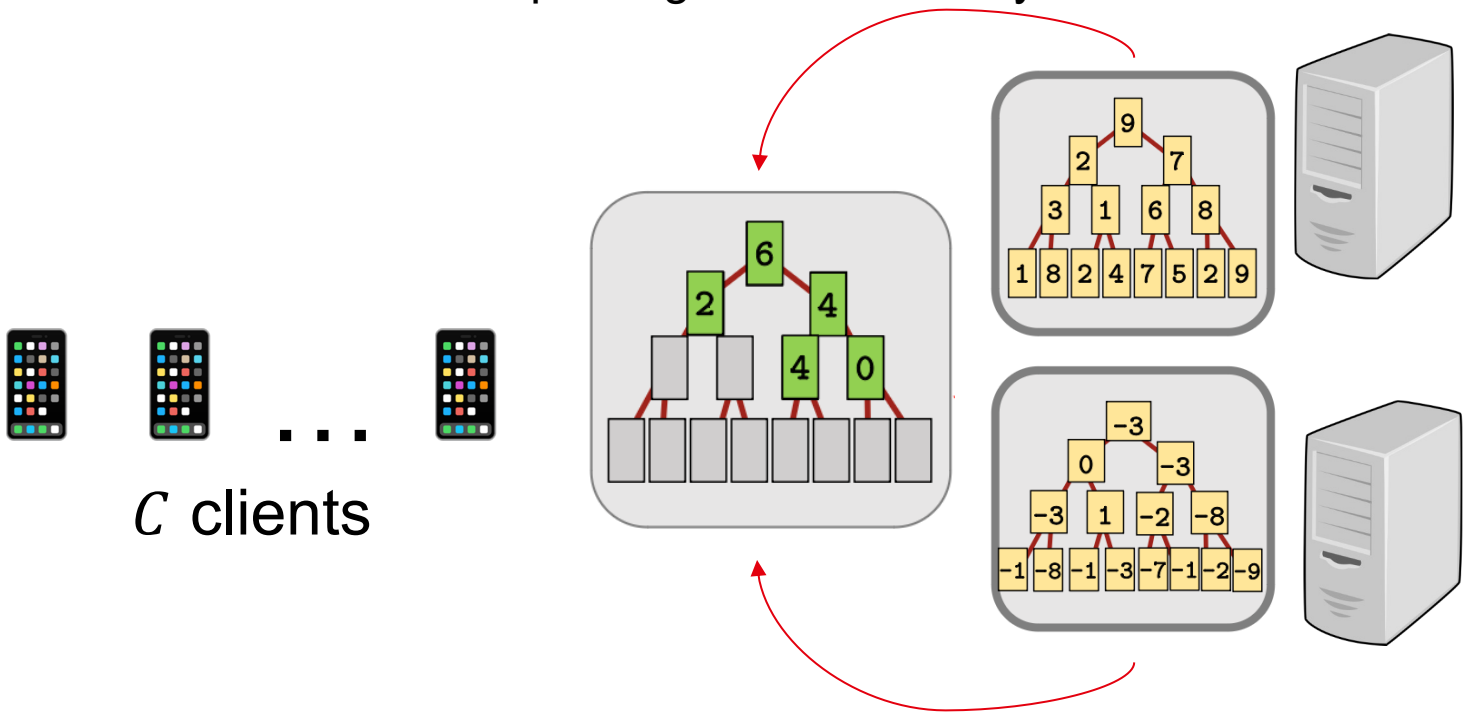
# Warm-up Scheme

- Servers sum up shares from each client to get aggregate shares



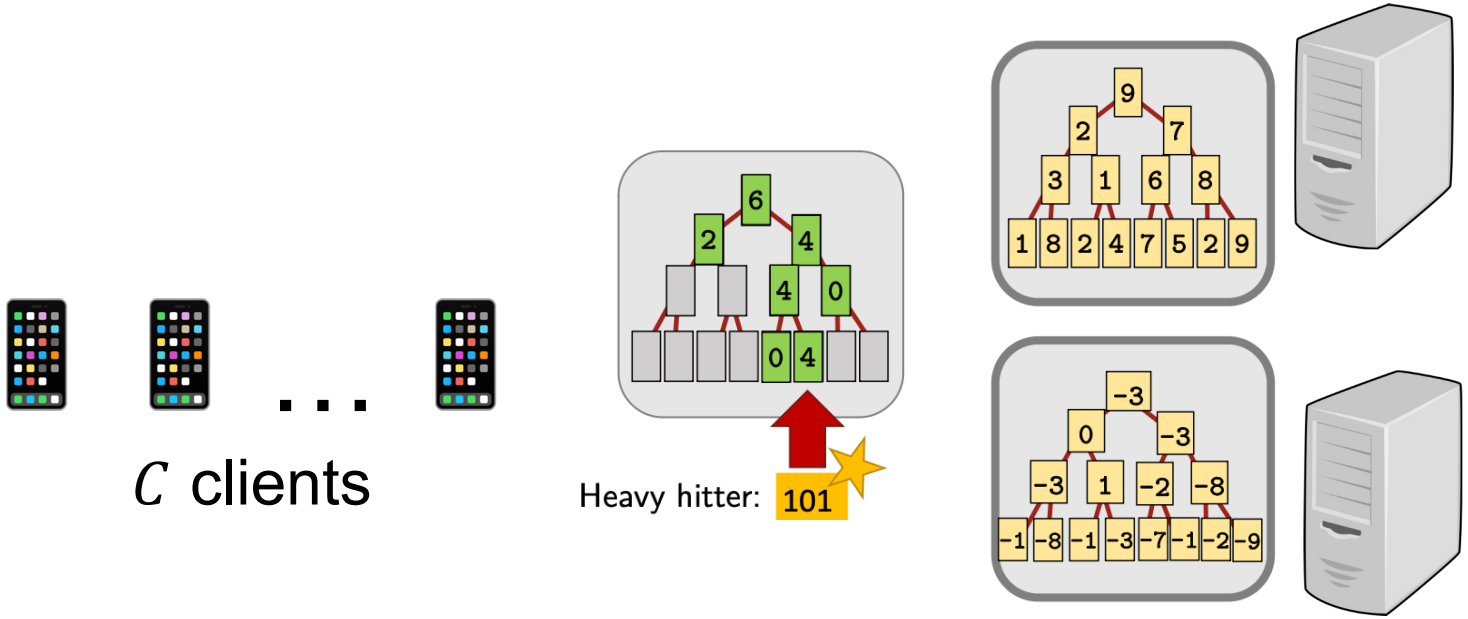
# Warm-up Scheme

- Servers publish shares to perform BFS search for heavy hitters
- Servers use BFS with pruning to find all heavy hitters



# Warm-up Scheme

- Servers use BFS with pruning to find all heavy hitters



# Technical Challenges

- Each tree is exponentially large (e.g., when strings are URLs, locations, passwords) ➔ Client cannot materialize it
  - Use incremental distributed point functions
    - Succint secret-sharing of a tree with one non-zero path
    - Communication $O(\lambda n)$ instead of $O(\lambda n^2)$ with normal DPF
- Client can send malformed secret shares ➔ Data corruption
  - Use malicious-secure sketching
    - Servers can test whether a secret-shared vector is non-zero in a single coordinate
    - Extractable distributed point functions

# Incremental Distributed Point Functions

# Standard DPFs

- A DPF is a cryptographic primitive for secret-sharing a vector that is non-zero at a single point.

$\boldsymbol{DPF\ Scheme} = (Gen, Eval)$ with:

- $Gen(\alpha, \beta) \rightarrow (k_0, k_1)$
  - Given point $\alpha \in \{0,1\}^n$ and value $\beta \in F$, output two DPF keys
- $Eval(k, x) \rightarrow F$
  - Given $DPF\ key\ k$ and $index\ x \in \{0,1\}^n$, output a secret share

**Correctness Property**

For all $\alpha \in \{0,1\}^n, \beta \in F, (k_0, k_1) \leftarrow Gen(\alpha, \beta)$, and $x \in \{0,1\}^n$:

- $Eval(k_0, x) + Eval(k_1, x) = \beta\ if\ x = \alpha$
- $Eval(k_0, x) + Eval(k_1, x) = 0\ otherwise$

- **Efficiency**: Key size is $O(\lambda n)$ bits, not $2^n$ (where $\lambda$ is security parameter)
- **Security**: A server with only one key learns nothing about $\alpha$ or $\beta$

# Incremental DPFs



- Standard DPF secret shares a sparse vector with one non-zero entry.
- Incremental DPF instead secret shares values along a single non-zero path in a binary tree with $2^n$ leaves.

# Incremental DPFs

An incremental DPF scheme, parameterized by finite groups $G_1, \ldots, G_n$, consists of two routines:

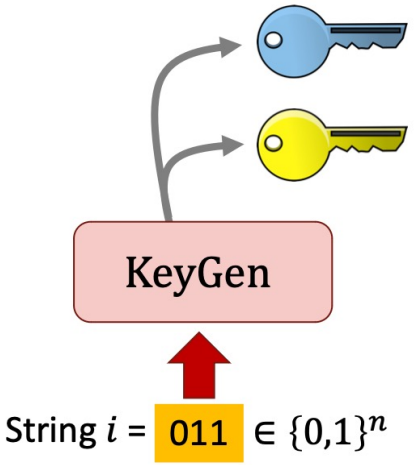
- $\mathrm{Gen}(\alpha, \beta_1, \ldots, \beta_n) \to (k_0, k_1)$. Given $\alpha \in \{0,1\}^n$ and $\beta_1 \in G_1, \ldots, \beta_n \in G_n$, output two keys.
- $\mathrm{Eval}(k, x) \to \bigcup_{\ell=1}^{n} G_\ell$. Given $k$ and $x \in \bigcup_{\ell=1}^{n} \{0,1\}^\ell$, output a secret-shared value.

Incremental DPF correctness property:

$$\mathsf{Eval}(k_0, x) + \mathsf{Eval}(k_1, x) = \begin{cases} \beta_\ell & \text{if } |x| = \ell \text{ and} \\ & x \text{ is a prefix of } \alpha \\ 0 & \text{otherwise} \end{cases}$$

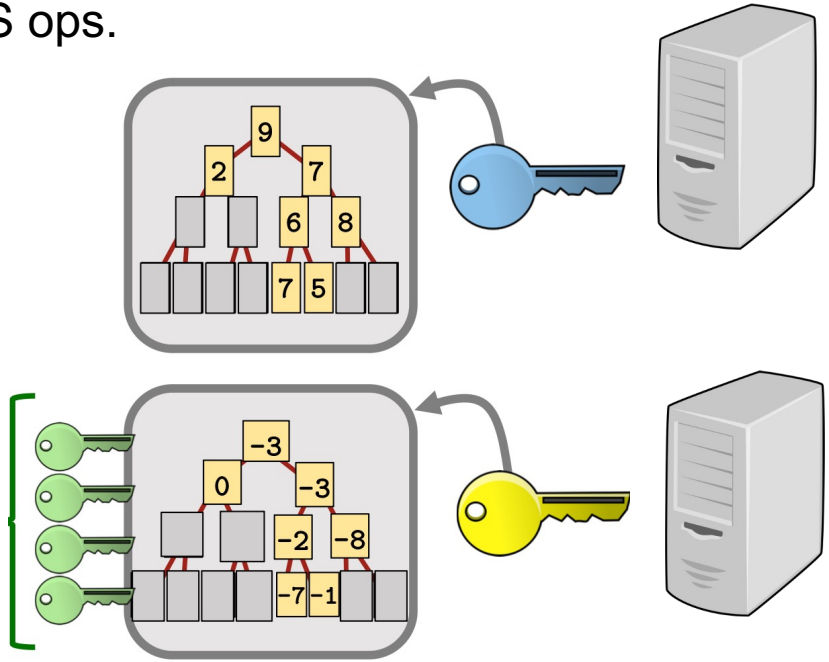
# Incremental DPFs

- Each client secret-shares the labels on a tree with one non-zero path and sends one share to each server. Communication $\approx \mathbf{2}^n$
- 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 $O(\lambda n^2)$.



# Incremental 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.



# Malicious-secure sketching

# Malicious-Secure Sketching

- **Purpose:** Allows servers to verify validity of client input, without leaking client data to a malicious server.
- **Client-Side Encoding**
  - Client wants to share a vector $v$
  - Generates a random secret key $k \in F$
  - Encodes $v$ as a pair $(v, kv)$
  - Sends additive shares:
    - $(v_0, kv_0) \rightarrow Server\ 0$
    - $(v_1, kv_1) \rightarrow Server\ 1$
  - Provides correlated randomness (shares of $a, b, c, A, B$) to enable secure $2PC$

# Malicious-Secure Sketching

- **Server-Side Verification**
  - Servers jointly sample random public sketch vectors $r, r*$
  - Each server $b \in \{0,1\}$ computes local sketch values:

$$z_b = \langle \bar{r}, \bar{v}_b \rangle, \quad z_b^* = \langle \bar{r}^*, \bar{v}_b \rangle, \quad z_b^{**} = \langle \bar{r}, \bar{v}_b^* \rangle$$

  - Servers run a secure 2PC (using client-provided randomness) to check:

$$(z^2 - z^*) + (\kappa \cdot z - z^{**}) = 0$$

  - Where:

$$z = z_0 + z_1, \quad z^* = z_0^* + z_1^*, \quad z^{**} = z_0^{**} + z_1^{**}$$

# Extractable DPFs

- **Problem:** Server-side sketching only validates a client's DPF on the queried input set $S$
- **Solution:** Design DPFs with a public part $pp$ and two private keys $k_0, k_1$
- **Key Property:** It is computationally infeasible for a malicious client to construct keys such that:

$$\mathrm{Eval}(k_0^*, pp^*, x) + \mathrm{Eval}(k_1^*, pp^*, x) = 1$$

- holds for more than one value of $x$ known to the client across the domain.

# Extractable DPFs



- **Extractor:** Efficiently identifies the unique $x$ encoded by the client.
- **Server Check:** Servers verify that both client keys use the same public part $pp$.
- **Defence:** Prevents the client from targeting multiple points - only one valid string is possible.

# Poplar System

# Poplar System

**Protocol 5: Private $t$-Heavy Hitters (Semi-Honest Secure Version)**

There are two servers and $C$ clients. Each client $i \in [C]$ holds a string $\alpha_i \in \{0,1\}^n$. The goal is to identify all $t$-heavy hitters in $(\alpha_1, \ldots, \alpha_C)$. The incremental DPF uses a finite field $\mathbb{F}$ with $|\mathbb{F}| > C$.

1. **Client Key Generation:** Each client $i$ sets $\beta_1 = \cdots = \beta_n = 1 \in \mathbb{F}$ and computes

$$(k_0^{(i)}, k_1^{(i)}) \leftarrow \mathrm{Gen}(\alpha_i, \beta_1, \ldots, \beta_n).$$

   The client sends $k_0^{(i)}$ to Server 0 and $k_1^{(i)}$ to Server 1, then can go offline.

2. **Server Query Computation:** Servers jointly execute Algorithm 3. When a prefix-count oracle query on a prefix string $p \in \{0,1\}^*$ is issued:

$$\mathrm{val}_{p,b} \leftarrow \sum_{i=1}^{C} \mathrm{Eval}(k_b^{(i)}, p), \quad b \in \{0,1\},$$

$$\mathrm{val}_p \leftarrow \mathrm{val}_{p,0} + \mathrm{val}_{p,1} \in \mathbb{F}.$$

3. **Output:** Servers return the result from Algorithm 3.

# Poplar System



**Algorithm 3: $t$-Heavy Hitters from Prefix-Count Queries**

**Input:** Oracle $\mathcal{O}_{\alpha_1,\ldots,\alpha_C}(p)$ returning number of strings with prefix $p$.
**Output:** The set of all $t$-heavy hitters in $(\alpha_1, \ldots, \alpha_C)$.

1. Initialize $H_0 \leftarrow \{\varepsilon\}$, and set $w_\varepsilon \leftarrow C$.
2. For $\ell = 1$ to $n$:
   - Set $H_\ell \leftarrow \emptyset$.
   - For each $p \in H_{\ell-1}$:
     - $w_{p||0} \leftarrow \mathcal{O}(p||0)$
     - $w_{p||1} \leftarrow w_p - w_{p||0}$
     - If $w_{p||0} \geq t$, add $p||0$ to $H_\ell$
     - If $w_{p||1} \geq t$, add $p||1$ to $H_\ell$
3. Return $H_n$.

# Leakage Mitigation with Differential Privacy

# Providing Differential Privacy

The heavy-hitters algorithm reveals:

- The set of heavy strings or prefixes
- For each heavy prefix $p$, the number of strings starting with $p$

This can leak information about individual client inputs.

→ **Goal:** Ensure $\boldsymbol{\varepsilon}$ **− differential privacy** to limit what an adversary can infer from the system's output.

→ Poplar's Mechanism for Differential Privacy:

- → Add noise sampled from a **Laplace distribution** to the prefix-count oracle responses.
- → The added noise can cause false positives or false negatives: balance privacy and correctness.

# Implementation and Evaluation

# Implementation and Evaluation

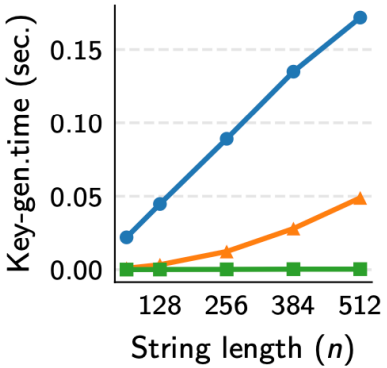
- ≈ 3500 lines of Rust
- Open-source implementation
  - http://github.com/henrycg/heavyhitters
- Google's C++ implementation of incremental DPF
  - https://github.com/google/distributed_point_functions

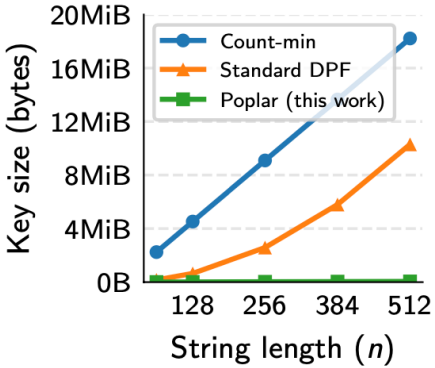
Experimental setup:

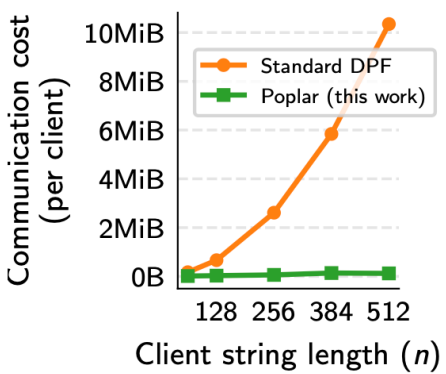
- Servers on opposite sides of US
  - Amazon EC2 us-east-1 (VA) and us-west-1 (CA)
- Simulated clients in us-east-1
- Each server is one c4.8xlarge (36 vCPU, 60 GiB RAM)

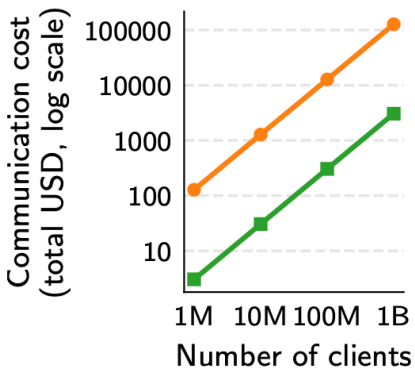
# Implementation and Evaluation

- Using Poplar saves computation and communication



# Implementation and Evaluation

- End-to-end evaluation of Poplar collecting 256-bit strings (long enough to hold a 40-character domain name)
- Client data sampled from Zipf distribution (parameter 1.03, support 10,000)
- Heavy-hitters threshold set to 0.1% of clients

**Completely parallelizable**

| | Running time (sec.) | | | |
|---|---|---|---|---|
| **Clients** | DPF | Sketching | Total | Clients/Sec. |
| 100k | 107.3 | 704.5 | 828.1 | 120.8 |
| 200k | 211.0 | 1,404.1 | 1,633.5 | 122.4 |
| 400k | 433.5 | 2,771.4 | 3,226.0 | 124.0 |

# Summary

- With 400,000 clients, server-side computation takes less than one hour over WAN.
- Privacy against **malicious server**, correctness against **malicious clients**
- New techniques introduced
  - More powerful distributed point functions: incremental & extractable
  - Malicious-secure sketching
- Application to other private data-collection problems
- Future directions:
  - Extend to finding heavy clusters rather than exact matches
  - Multi-server setting with more than 2 servers


# Thanks!