Interoperable Private Attribution (IPA)

August 2022 Update

We have published an updated version of IPA

- You can find the updated proposal in the “patcg-individual-drafts” folder on GitHub (link)
- Our original proposal (more of a loose sketch) was published back in January 2022. You can find it on Google Docs (link)

What's changed, and why?

- **Simplification of Client APIs**: We realized that the client can simply share encrypted matchkeys, rather than individual source and trigger events.
- **Formalization of Helper Party Networks**: This was alluded to in the initial proposal, and now is a formal concept.
- **Differential Privacy**: Our initial proposal stated our goal of using differential privacy, and this update clarifies how we can achieve it.

Most importantly,

- **Achieved our security goals**: Our updated draft provides a new, improved, and far more detailed MPC protocol that achieves our security goals.

Why didn’t it meet our security goals before?

The previous version leaked a small bit of information

- MPC helper nodes would learn the number of events per person

Why is that a problem?

- If any single MPC helper node were malicious, it could collude with the business making the IPA query to use these per-user counts to try to re-identify them.
- For example:
  - The API caller could generate 200 source events for Ben and 1 for everyone else
  - The MPC helper server would later on see that one of the “doubly blinded match keys” was associated with 200 source events.
  - If the MPC helper was colluding with the API caller, it would now know that “doubly blinded match key” referred to Ben

This did not meet our security goals

Our goal is “Malicious Security”

- Our goal is to have no single point of failure
- We want it to be the case, that if a single MPC helper node chooses to act maliciously, user privacy is still preserved.

Things we tried

- We tried adding differential privacy to these counts by generating fake events
- We tried capping the number of events per double–blinded match key and then shuffling before attribution
- We tried giving each helper a random fraction of all the events.

We were not able to solve the problem to our satisfaction

What does our updated proposal do?

We have proposed a zero-leakage approach

- The MPC helper nodes learn nothing
- The entire computation is performed on secret shared numbers

The updated proposal provides “malicious security”

- If any MPC helper node were to act maliciously, including deviating from the protocol and attacking it in arbitrary ways, it would *still* learn nothing.
  - This includes any and all possible collusion between the MPC helper node and the API caller
  - …or collusion between the MPC helper node and the “match key provider”

But malicious security can be expensive...

- Fortunately, there is a solution;
- If you have 3 MPC helpers instead of 2, there exist much more efficient techniques

The updated security model:

- There is a great deal of academic research on highly efficient techniques to perform computations on secret-shared data where there are:
  - 3 Helpers
  - One of them might be malicious
  - But we assume there are at least 2 “honest” helpers
- In this model, if one of the helpers turns “malicious” and attempts to deviate from the protocol:
  - It fails to learn any private information
  - The other two “honest” helpers can detect that something is amiss and abort the computation.

What is “secret sharing”?

One type of “secret sharing” is “additive secret sharing”

```
let secret = 8;

let p = 31;

let share_1 = RandBetween(0, p);

let share_2 = RandBetween(0, p);

let share_3 = Mod(secret - share_1 - share_2, p);
```

All of `share_1`, `share_2` and `share_3` are randomly distributed numbers in the range [0, 31). If you possess just one, or even two of them, you know absolutely nothing about the secret value.

“Replicated Secret Sharing” is very useful

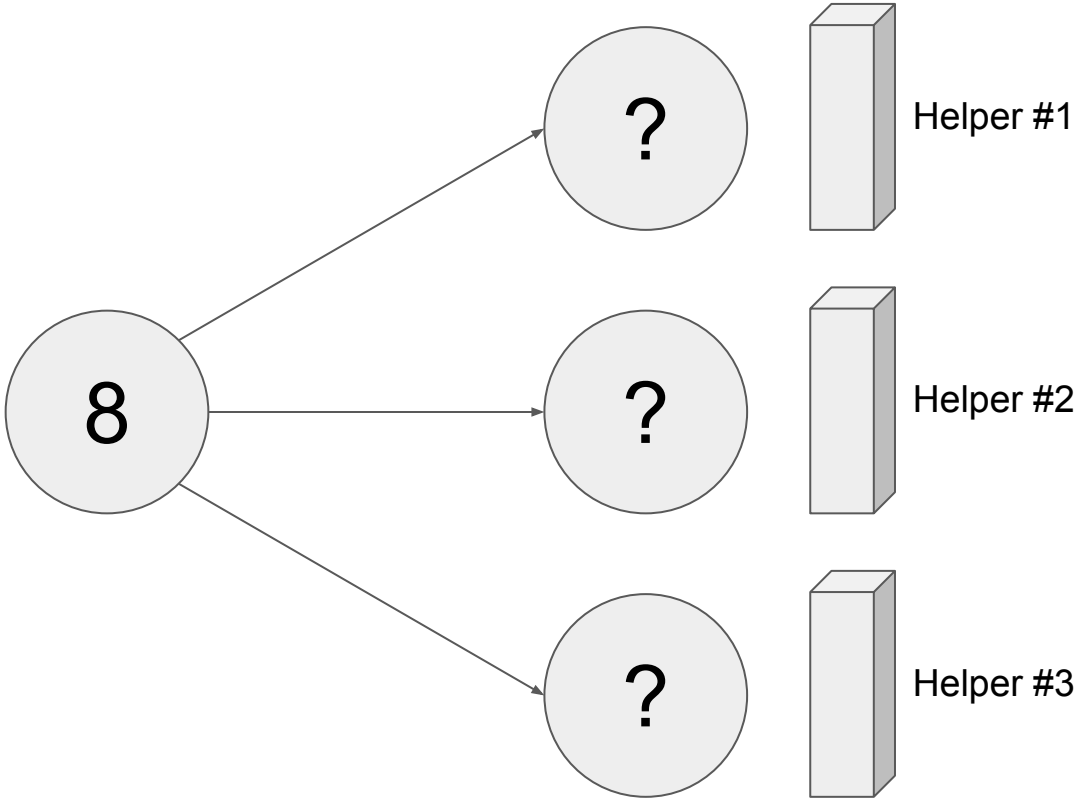
```
Helper 1: (share_1, share_2)

Helper 2: (share_2, share_3)

Helper 3: (share_3, share_1)
```

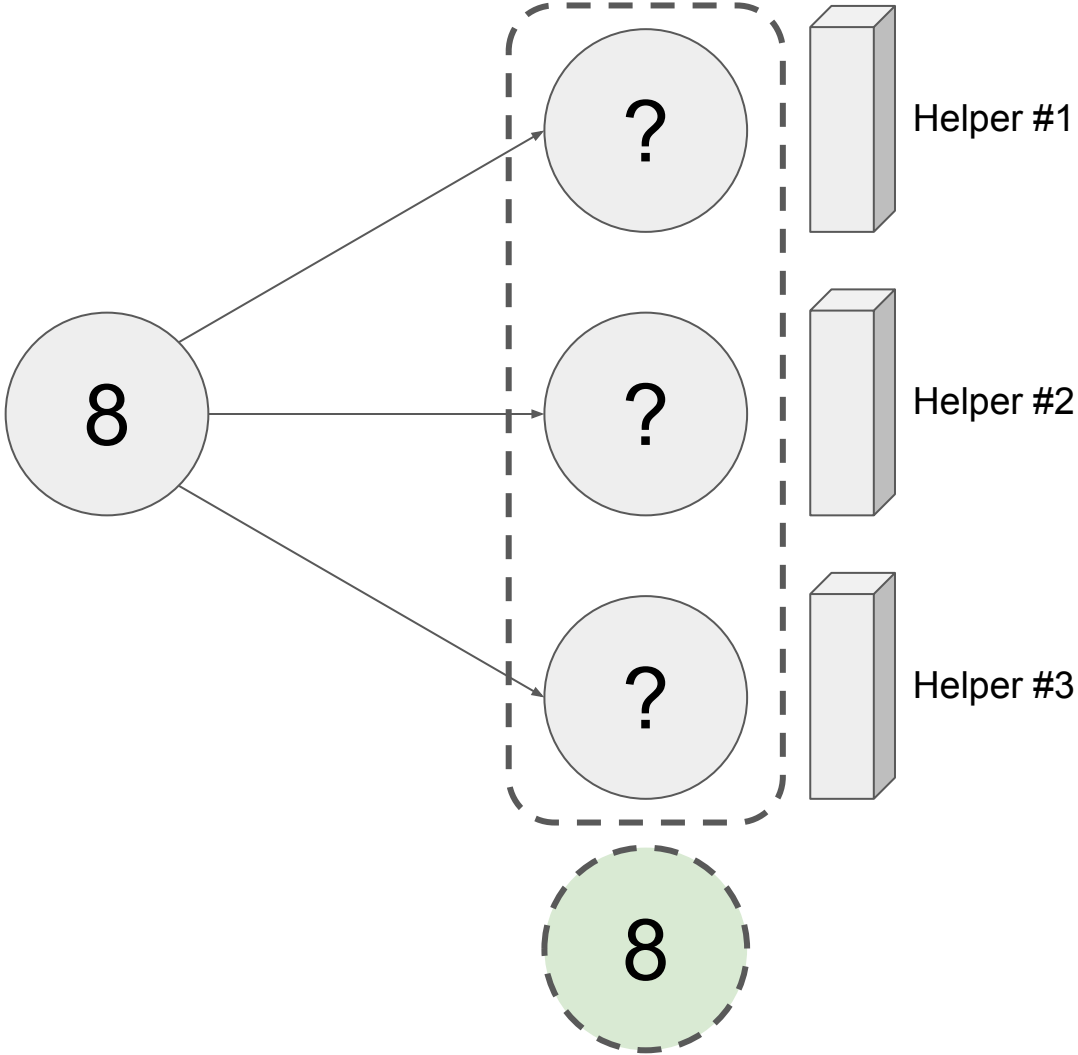
Consider the “additive secret sharing” from the last slide, and give each helper two of the three shares

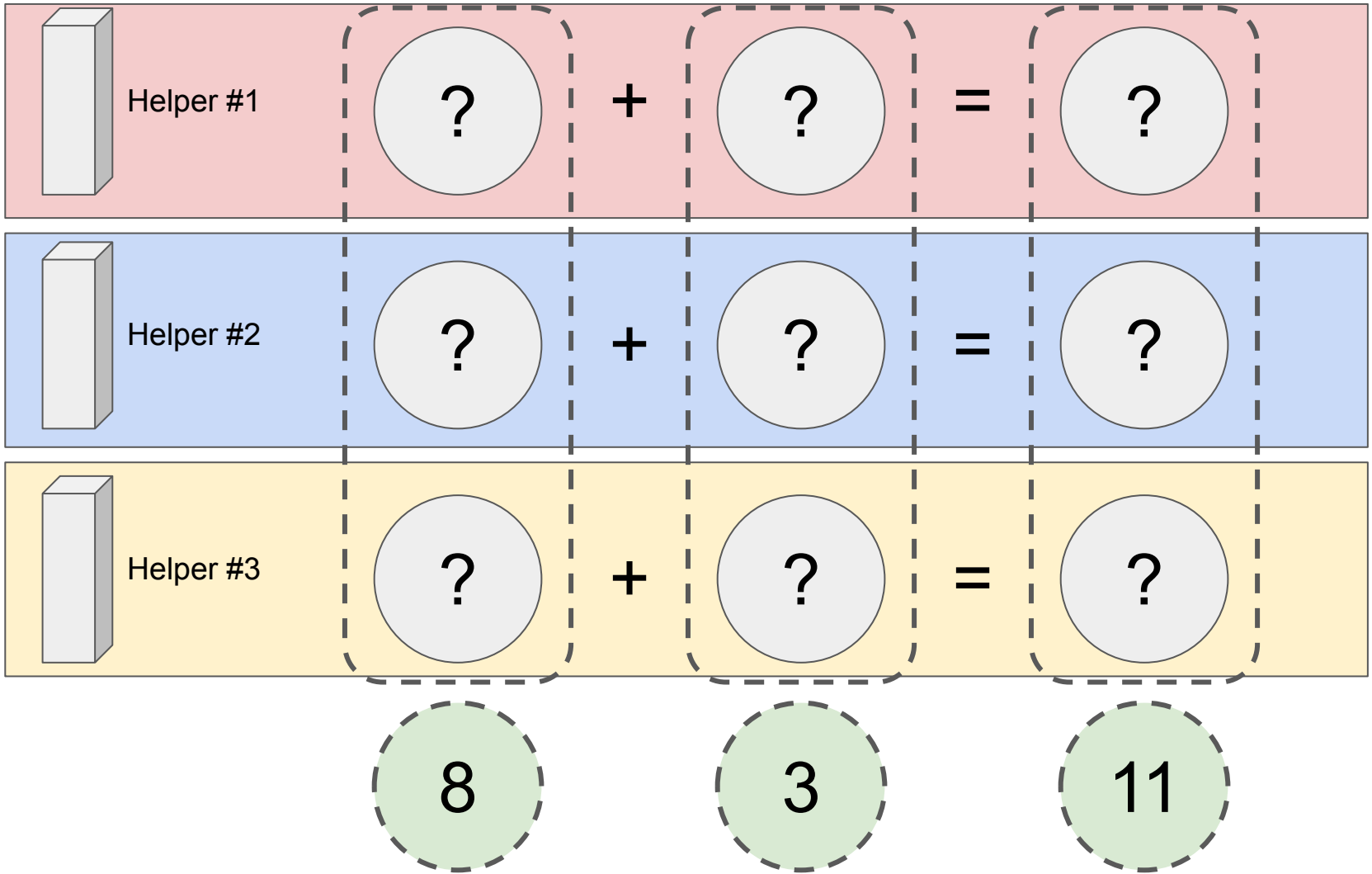
That’s not enough information for any one helper to reconstruct the secret

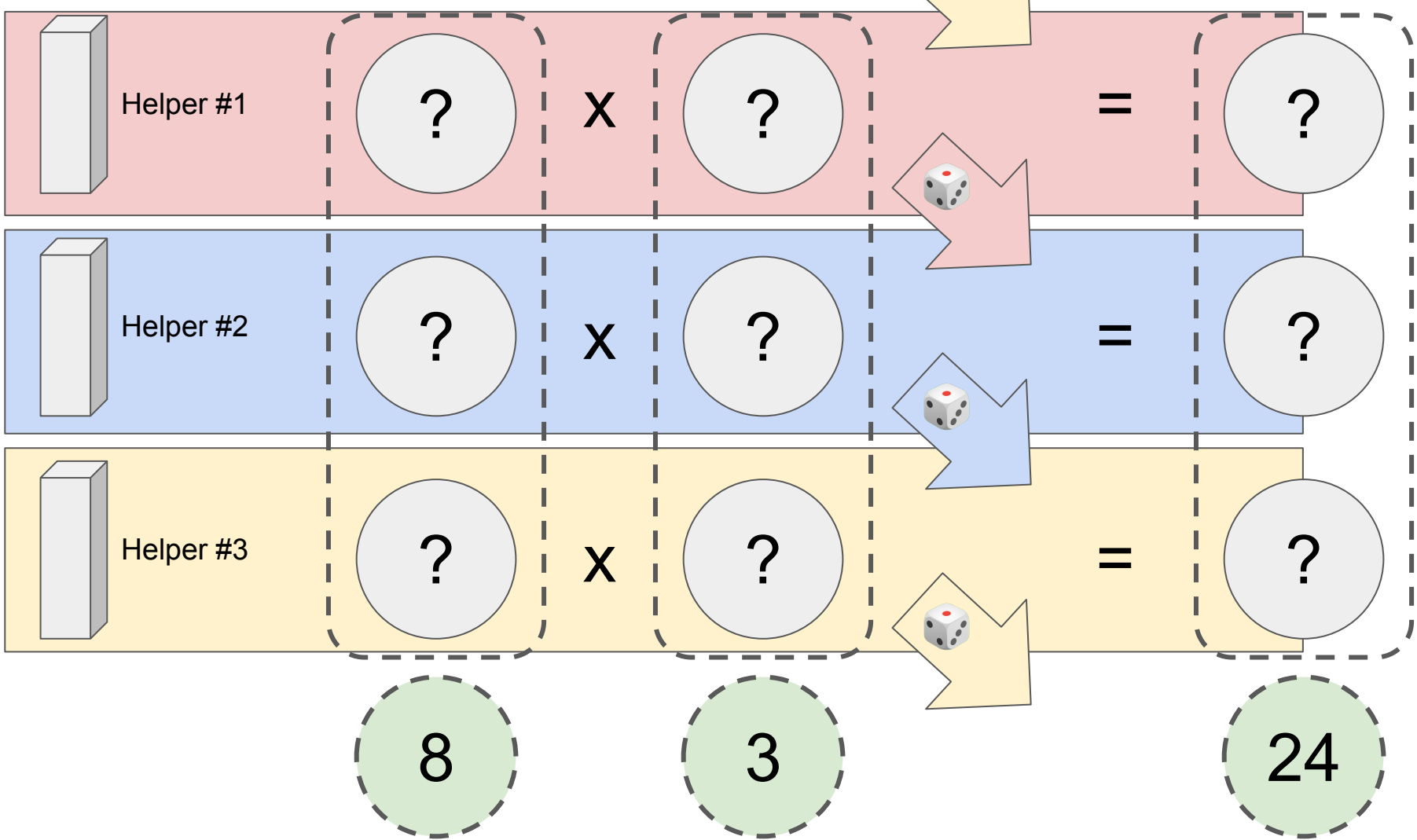
This can make multiplication much simpler

Our current prototypes use this type of secret sharing


8
?
?
?
Helper #1
Helper #2
Helper #3


8
?
?
?
Helper #1
Helper #2
Helper #3
8


Helper #1
?
+
?
=
?
Helper #2
?
+
?
=
?
Helper #3
?
+
?
=
?
8
3
11


Helper #1
?
x
?
=
?
Helper #2
?
x
?
=
?
Helper #3
?
x
?
=
?
8
3
24

Performance

- The bottleneck in such systems tends to be communication between the MPC helper nodes
- CPU tends not to be a bottleneck
- **Addition** of secret shared numbers requires no communication between the nodes
- **Multiplication** of secret shared numbers ***does*** require communication between the nodes
- So it’s important to minimize the number of multiplications you need to do

The cost of a multiplication

In the honest majority with 3 parties setting, the communication cost of performing a multiplication is:

Semi-honest setting: 1 number exchanged per helper

Malicious setting: 2 numbers exchanged per helper

How the updated MPC protocol works

1. First, helpers receive a big list of secret shared events
2. Next, they sort these events by match_key, ~~then timestamp~~
3. Then, they run an “oblivious attribution algorithm” over these events
4. At this point, we need to “cap” the maximum contribution each individual person can contribute to the output
5. Then, they sum up the secret shared “trigger values” (per breakdown key)
6. After this, they generate some random noise and add it to these totals
7. Finally, they “reveal” these totals to the API caller (i.e., give them all 3 shares)

There is a lot of academic research on sorting in MPC

- Specially tailored sorting protocols have been getting faster and faster with better and better security
- This is an active area of research and we expect performance of “sort” to continue to improve
- We have researchers who are actively working several promising optimizations.

Oblivious Attribution Algorithms

- In principle, almost any type of attribution heuristic is possible
  - Only limited by the data that is provided in events
  - Simple multi-touch (e.g. equal credit last 3 touches) is definitely possible
  - Addition is cheap
  - Multiplication is a *little* expensive
  - Division / exponentiation / etc. is going to cost you =)
- We have shared a specific algorithm which performs “last-touch attribution”
  - It only requires addition and multiplication
  - It’s very inexpensive; only requiring a few multiplications per event
- We are happy to share more examples of other “oblivious attribution algorithms” in the future

Capping the per-user contribution

- This does make the result less accurate, but it is necessary to provide differential privacy
- We can, however allow the API caller to decide what this “cap” is
  - High cap
    - Less bias introduced through capping
    - More random noise added to the totals
  - Low cap
    - More bias introduced through capping
    - Less random noise added to the totals
- We would love feedback on how to approach “capping”
  - Stop taking events once the cap is reached?
  - Scale down all events from a user?

Managing a “Privacy Budget”

- We have dramatically simplified the proposal of how to manage a privacy budget
- In the first version of our proposal, we suggested the helpers would manage a user-level privacy budget
  - This was also a potential source of “information leakage”
  - It’s also a lot more complexity for the helpers to deal with
- In this version, we have proposed a much simpler “query-level” privacy budget
  - Each query consumes some amount of “privacy budget” (The API caller decides how much) and when you run out you can’t make any more queries that week
- This is super-simplistic. We know it’s not optimal and this is an area where we would love help and input from this group.

A Research Prototype of IPA

- There is a compiler called MP-SPDZ which can be used to compile code into an MPC program.
- We are using this to iterate upon and test out IPA
- This is NOT production-ready code
- This is just a research prototype
- You can find this research prototype of IPA here: (link)

Research Prototype Benchmarks

We will keep optimizing, but the prototype is likely *already efficient enough* for a great deal of advertisers.

We benchmarked **1 million** source and trigger events. If we assume 1:1 source to trigger event ratio:

- At a CPM of $14.40 it would cost $7,200 to generate 500k impressions
- At a CPC of $0.35 it would cost $175,000 to generate 500k clicks

Given this benchmark, our research prototype:

- **Costs <$50** (at stock AWS egress and compute rates)
  - That's < 0.7% of the campaign cost with $14.40 CPM, and
  - < 0.03% of the campaign cost with $0.35 CPC
- Takes **~100 minutes** to run
- Uses **~390 GB** of communication bandwidth

These are extrapolated from runs on ~500k rows, due to limitations at 2^{19} with MP-SPDZ.
We use m5dn.12xlarge instances that cost $3.20/hr.

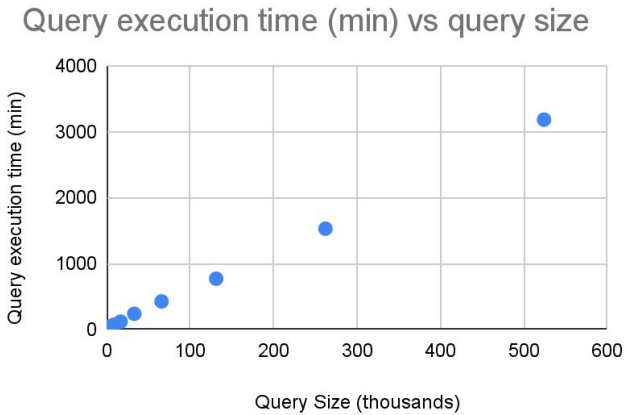
We Can Do Better!

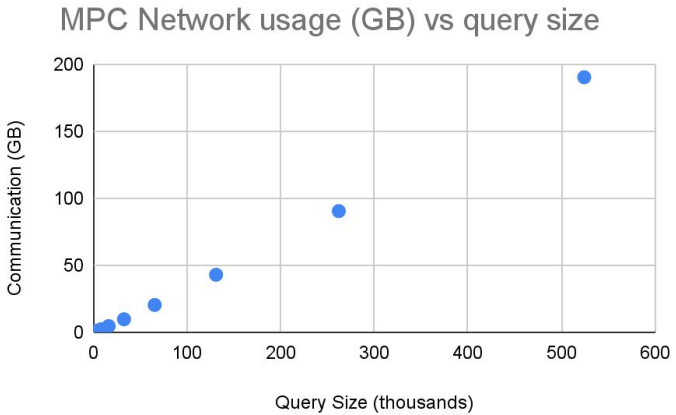
Recent literature suggests that with 2 out of 3 honest majority, malicious security should only be ~2x over semi-honest security. Running in MP-SPDZ with semi-honest, and assuming ~2x cost, we get:

- **Costs <$20** (at stock AWS egress and compute rates)
  - That's < 0.3% of the campaign cost with $14.40 CPM, and
  - < 0.012% of the campaign cost with $0.35 CPC
- Takes **~26 minutes** to run
- Uses **~160 GB** of communication bandwidth

These are extrapolated from runs on ~500k rows, due to limitations at 2^{19} with MP-SPDZ.
We use m5dn.12xlarge instances that cost $3.20/hr.

Research Prototype Benchmarks (Malicious)





Malicious

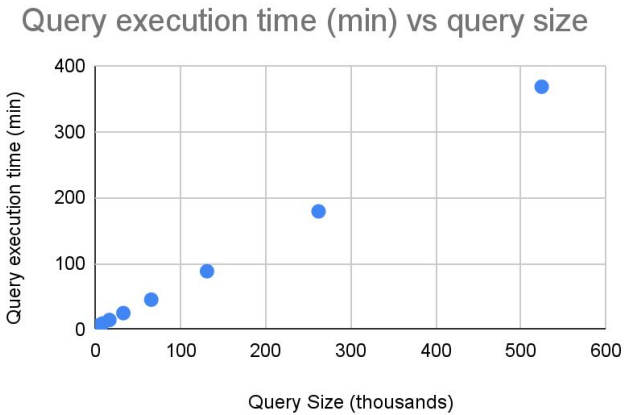
Network usage and time scale as:

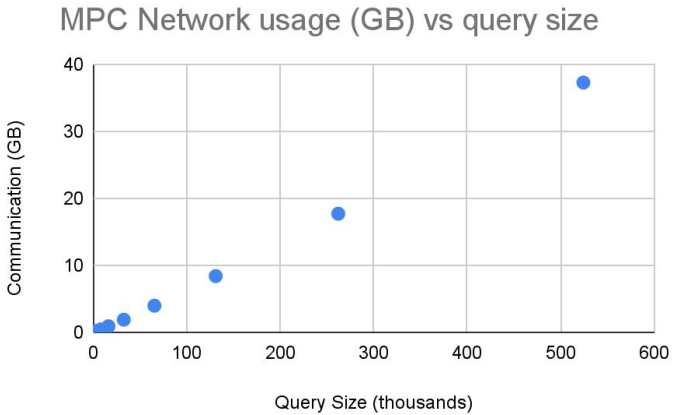
- Network usage (GB) = 0.0196 (query size) * log_2(query size), so a query size 1M extrapolates to 390 GB
- Time (sec) = 0.0003 (query size) * log_2(query size), so a query of size 1M extrapolates to 100 min.

Cost Estimate:

- Network Cost for 1M = ($0.08/GB) *(390 GB) = $31.20
- Compute cost for 1M = (100 min) * ($3.2/hr) * (3 machines) = $16
- Total: $47.20

Research Prototype Benchmarks (Semi-honest)





Semi-honest:
Network usage and time scale as:

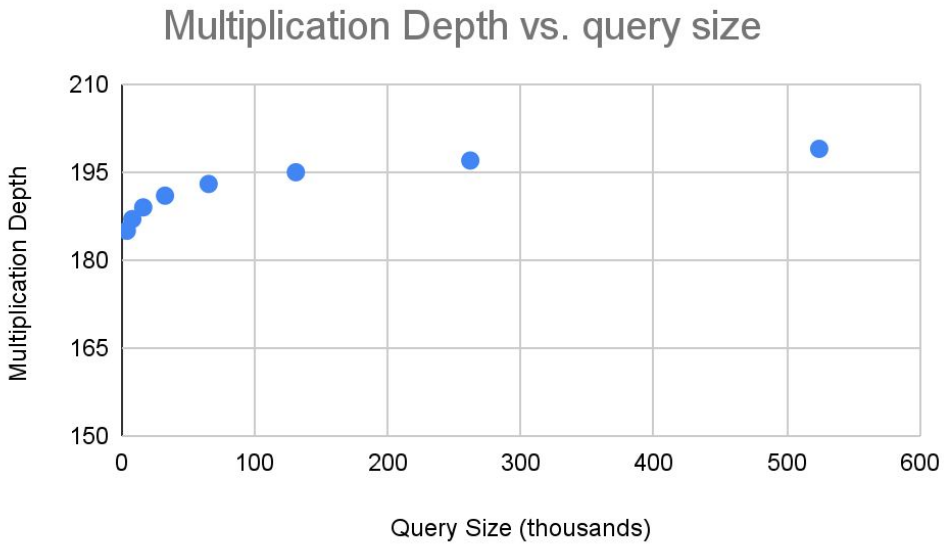
- Network usage (GB) = 0.000004 (query size) * log_2(query size), so a query size 1M extrapolates to 79.7 GB
- Time (sec) = 0.00004 (query size) * log_2(query size), so a query of size 1M extrapolates to 13.3 min.

Cost Estimate:

- Network Cost for 1M = ($0.08/GB) *(79.7 GB) = $6.37
- Compute cost for 1M = (13.3 min) * ($3.2/hr) * (3 machines) = $2.12
- Total: $8.49

Research Prototype Benchmarks

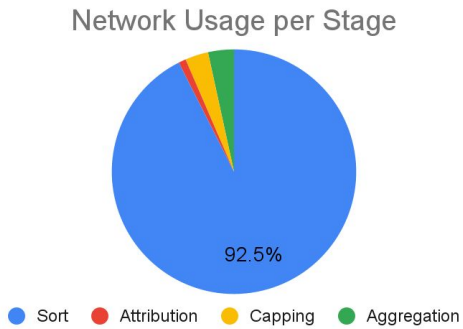
The multiplicative depth of our circuit scale logarithmically in query size

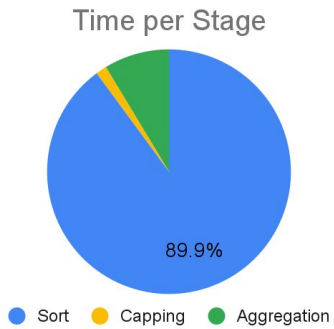


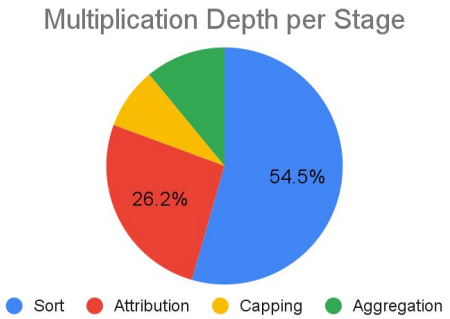
Circuit depth for either malicious or semi-honest

Research Prototype Benchmarks

We measure performance of the different components of IPA.



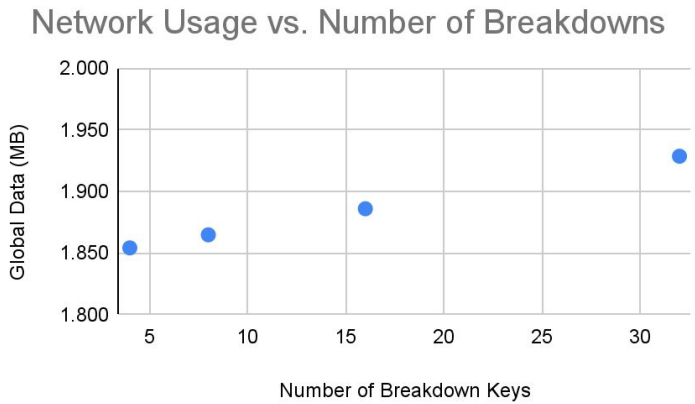


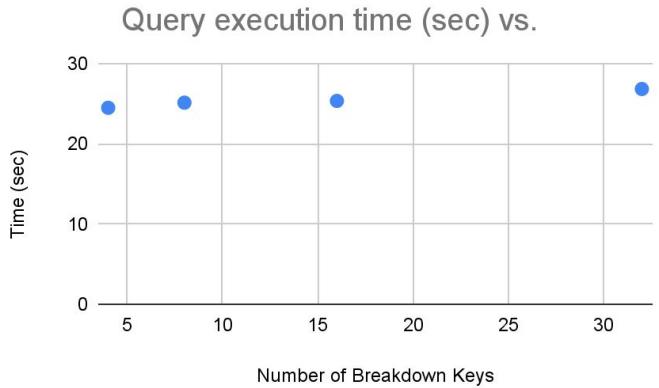


Measured at 2^15 rows, 16 breakdowns, and 32 bit matchkeys with malicious security

Research Prototype Benchmarks

End-to-end cost of IPA for increasing number breakdowns





Measured at 2^15 rows and 32 bit matchkeys with semi-honest security

Q&A