STAR: Distributed Secret Sharing for Threshold Aggregation Reporting

PPM WG, IETF 114

Shivan Kaul Sahib
Alex Davidson
Pete Snyder
Idea: k-anonymity for clients reporting measurements to an untrusted server
Goals

- Cheap: low computational overhead and network usage for clients and servers
- Simple: easy to implement, well-known crypto
- Private: practical privacy guarantees
Central Idea: use Shamir’s Secret Sharing

- Compute **symmetric key** $K$ by hashing measurement $x$: $K = H(x, \text{rand})$
- Client encrypts $x$ using $K$: $M = \text{Encrypt}(K, x)$
- Client generates **secret share** of key: $\text{SecretShare}(K)_i$
- Client sends server: $M, \text{SecretShare}(K)_i$

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
</tr>
</tbody>
</table>

- Server gets: $M, \text{SecretShare}(K)_i$
- After $N$ secret shares, recover $K$: $K = \text{Recover}(\text{SecretShare}(K)_{i..N})$
- Use $K$ to decrypt $M$: $x = \text{Decrypt}(K, M)$
Central Idea

● Use anonymizing proxy
  ○ OHAI

● Use Randomness Server
  ○ Client sends blinded input value to Randomness Server to get salt
  ○ To mitigate server brute-force computing all possible input values
  ○ Use OPRF so Randomness Server does not learn input value
Client

Randomness Phase

request(blinded(x))

response(randomness)

Message Phase

Generate Message

encrypted message

(over an anonymizing proxy)

Key rotation

Aggregation Phase

Aggregation Server

Reveal \( x \) from each message if \( x \) sent by \( \geq k \) clients.
Implementation

- Shipping in Brave browser for **some** telemetry
  - Rust: [https://github.com/brave/sta-rs](https://github.com/brave/sta-rs)
  - WASM bindings: [https://github.com/brave/sta-rs/tree/main/star-wasm](https://github.com/brave/sta-rs/tree/main/star-wasm)

- Using a TCP proxy for now
  - experimenting with OHAI proxy
What’s new in -01

- No longer use puncturable POPRFs.
- Always require use of a Randomness Server.
- Document risk of collusion between various entities in STAR.
- Clarify semantics of auxiliary data.
- Compare security guarantees and significantly expand the Security Considerations section.
- Add details on leakage.
Ready for Adoption?
What’s new in -01

- No longer use puncturable POPRFs.
  - Clients send measurements one epoch after randomness sampling, after Randomness Server has rotated keys
What’s new in -01

- Always require use of a Randomness Server.
  - In some cases, if the input values (the measurements) are high entropy, then you could avoid use of a Randomness Server.
  - Based on feedback, this is tricky to give advice on, so just require non-colluding Randomness Server.