STAR: SECRET SHARING FOR THRESHOLD AGGREGATION REPORTING

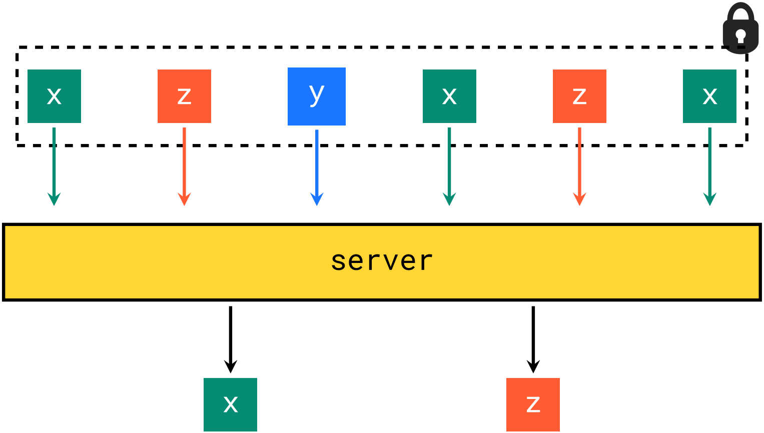
Alex Davidson[1] **Peter Snyder**[1] Joseph Genereux[1]
E. B. Quirk[1] Benjamin Livshits[2] Hamed Haddadi[1,2]

[1]Brave Software

[2]Imperial College London

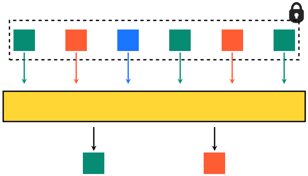
ACM CCS 2022 ::: Los Angeles, USA

$k = 2$

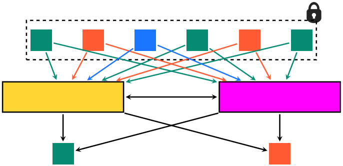


Sometimes known as k-heavy-hitters

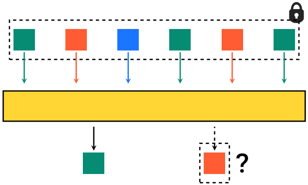
THRESHOLD AGGREGATION



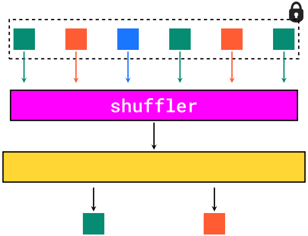
Ideal case: No efficient solutions



N-server aggregation: DPFs, Prio, SMPC

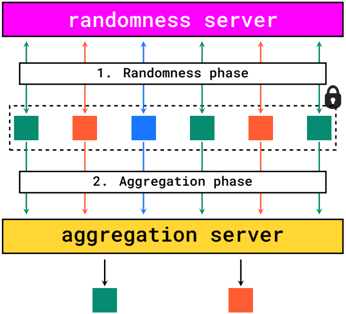


Approximate: DP, randomised resp.



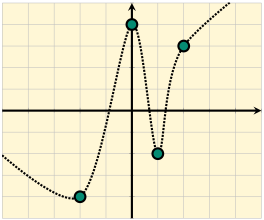
Trusted shuffling: e.g. Prochlo

PREVIOUS WORK



- Emphasis on simplicity and performance
- Well-known cryptography (secret sharing, OPRFs)
- Orders of magnitude cheaper than state-of-the-art
- Malicious security
- Auxiliary data support
- Open-source rust code: github.com/brave/sta-rs

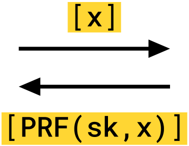
OVERVIEW OF STAR



Shamir secret sharing

Methodology:

- Only use well-understood (secret sharing) or standardized (OPRFs, encryption) primitives
- As efficient as possible
- Existing implementations where possible

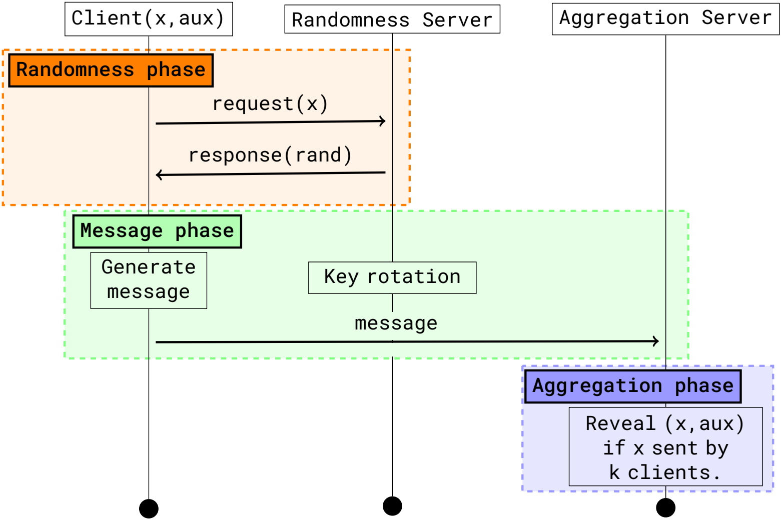


Oblivious PRF

`c = Enc(ek,m)`

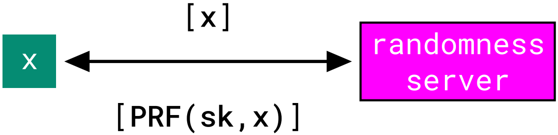
Symmetric encryption

CRYPTOGRAPHIC TOOLS



THE STAR PROTOCOL

Randomness phase



Message phase

- (r_1, r_2, r_3) = H(PRF(sk,x))
- s = Share(secret=r_1;randomness=r_2), t = r_3
- ek = Derive(r_1)
- c = Enc(ek,m=(x,aux))

RANDOMNESS & MESSAGE PHASES

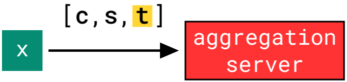
Aggregation phase

Steps

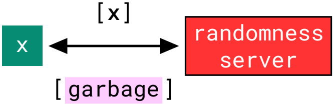
- Group messages based on deterministic tag **t**
- If $\geq$ **k** messages in the group, run share recovery on **s** and retrieve $\mathbf{r_1}$
- Derive **ek** from $\mathbf{r_1}$
- Decrypt each **c** to learn (**x**,aux)

AGGREGATION PHASE

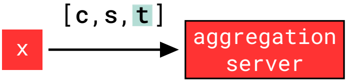
Malicious security in random oracle model



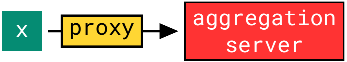
Problem: Deterministic tags
Solution: Randomness server key rotations



Problem: Randomness DoS
Solution: Clients can verify randomness (VOPRF)

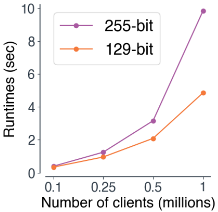


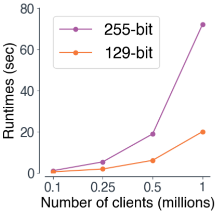
Problem: Sybil attacks
Solution: All threshold aggregation schemes vulnerable

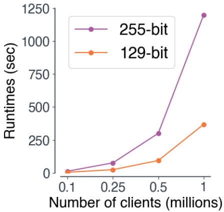


Problem: Client identity
Solution: Proxy messages, e.g. via Tor, or via randomness server using Oblivious HTTP

Aggregation runtimes ($k \in \{0.01\%, 0.1\%, 1\%\}$)







Other costs (per-client)

- **Communication:**
  - Aggregation: 233 bytes (+ auxiliary data)
  - Randomness server: 165 bytes
- **VOPRF:** < 2ms
- **OHTTP:** < 1ms, and approx. 4x communication

PERFORMANCE (256-BIT MEASUREMENTS)

Features

Feature	STAR	Poplar (S&P'21)
Aggregation servers (#)	1	2
Auxiliary data	✓	✗
Leakage	Tag-based	Prefix-based
Identity-hiding	✓ (OHTTP)	✓
Cryptography	Well-known	Distributed point functions

Headlines (including OHTTP)

- Computation: **1773x** faster
- Bandwidth: **62.4x** smaller
- Financial: **24x** cheaper[1]

[1] AWS c4.8xlarge Feb 2022

COMPARISON (STAR & POPLAR)

- Simple, Cheap Privacy-Preserving Threshold Aggregation with **k**-anonymity

- Implementations:
  - github.com/brave/sta-rs (Rust)
  - github.com/chris-wood/star-go (Go)

- IETF standardization: draft-dss-star-02

- Used in Brave for private analytics

CONCLUSIONS & FUTURE WORK