STAR: Secret Sharing for Private Threshold Aggregation Reporting

Alex Davidson
Brave Software

Peter Snyder
Brave Software

E. B. Quirk
Brave Software

Joseph Genereux
Brave Software

Benjamin Livshits
Imperial College London

Hamed Haddadi
Brave Software
Imperial College London

ABSTRACT

Threshold aggregation reporting systems promise a practical, privacy-preserving solution for developers to learn how their applications are used "*in-the-wild*". Unfortunately, proposed systems to date prove impractical for wide scale adoption, suffering from a combination of requiring: *i)* prohibitive trust assumptions; *ii)* high computation costs; or *iii)* massive user bases. As a result, adoption of truly-private approaches has been limited to only a small number of enormous (and enormously costly) projects.

In this work, we improve the state of private data collection by proposing STAR, a highly efficient, easily deployable system for providing cryptographically-enforced κ-anonymity protections on user data collection. The STAR protocol is easy to implement and cheap to run, all while providing privacy properties similar to, or exceeding the current state-of-the-art. Measurements of our open-source implementation of STAR find that STAR is 1773× quicker, requires 62.4× less communication, and is 24× cheaper to run than the existing state-of-the-art.

CCS CONCEPTS

• **Security and privacy** → **Privacy-preserving protocols**.

KEYWORDS

threshold aggregation; private analytics

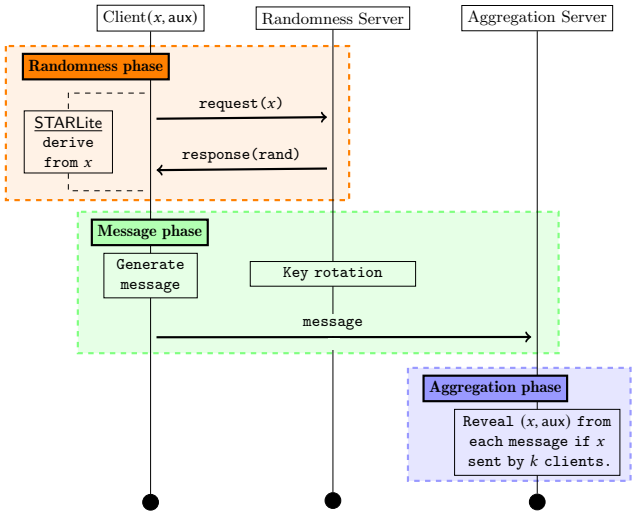


Figure 1: General STAR architecture. In the Randomness phase, clients sample randomness from a dedicated server. In the Message phase, clients generate their messages to send to the aggregation server. The aggregation server learns those measurements sent by κ clients in the Aggregation phase. Client randomness can be sampled locally, if the measurement distribution is sufficiently entropic (STARLite, Section 7.1).

1 INTRODUCTION

Application developers often need to learn how their product is used, and in which environments their applications runs. Such information helps developers debug errors, address security issues, and optimize implementations.

However, collecting such information puts user privacy at risk. Among other concerns, collecting user data, even de-identified data, may allow a data collector to profile a user or link records, revealing increasingly rich information about users over time. Naive data collection can harm user privacy in ways unintended by the developer and/or unexpected by the user.

A common approach for protecting user privacy when collecting client measurement data is to only learn those measurements that are sent by κ clients (κ-heavy-hitters). In these systems, the central server only learns the measurement if there are at least $\kappa - 1$ other clients that provide it as well. This approach prevents the data collector from learning uniquely identifying (or uniquely co-occurring patterns of) values, with the broader goal of preventing the identification of any individuals in aggregate dataset. Such guarantees are strongly related to the privacy notion of κ-anonymity [37]. We refer to systems that can provide such guarantees as **threshold aggregation systems**.

ACM CCS 2022, November 7–11 2022, Los Angeles, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. . . $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Designers of threshold aggregation systems face a challenging dichotomy though: how to allow a server to determine if it has collected κ identical records, without: i) the server first seeing the underlying value; and ii) in a manner that protects the user against a malicious (or generally untrusted) server.

Many systems have been proposed to try and square this circle [4, 5, 8–10, 12–14, 19, 27, 31, 34, 40]. However, all such systems to date have properties that make them impractical for most developers and telemetry systems. More specifically, all systems to date have at least one of the following undesirable properties:

- expensive server-side aggregation [9, 27];
- non-collusion assumptions for servers that communicate with each other [8, 10];
- interactive communication between clients [13, 27, 31];
- trusted third parties or hardware [8];
- difficult to apply for cases where $\kappa > 1$ [13, 31];
- require noise injection, and so require large user bases and/or entail utility loss [4, 5, 12, 19, 34, 40];
- restricted to numeric data types [1, 14];
- unbounded worst case leakage [4, 5, 12, 34, 40].

1.1 The STAR approach

In response to these issues in current threshold aggregation systems, we propose STAR; a practical, private threshold aggregation system that prioritizes *i)* efficiency (so that it can be deployed at extremely low cost), *ii)* limited trust assumptions (so that the trust requirements can be achieved by a wider range projects), and *iii)* simple, well-established cryptography (so that systems can be implemented and audited by a wider range of developers).

Further, STAR provides capabilities existing threshold aggregation systems lack, allowing STAR to solve use cases unaddressed by current state of the art. Specifically, STAR allows developers to attach arbitrary (but still threshold-protected) data to client messages.

Overall idea. Figure 1 presents an overview of the STAR approach. Each client constructs a ciphertext by encrypting their measurement (and any auxiliary data) using an encryption key derived deterministically from i) any randomness present in the client measurement and ii) additional randomness provided by a "randomness server". This randomness server never learns client values or inputs.

The client then sends: i) the ciphertext; ii) a κ-out-of-N secret share of the randomness used to derive the encryption key; and iii) a deterministic tag informing the server which shares to combine. The aggregation server groups reports with the same tag, and recovers the encryption keys from those subsets of size $\geq K$. Thus, the server learns all the measurements that are shared by at least κ clients (along with any auxiliary data).[1]

The aforementioned randomness server runs an oblivious pseudorandom function (OPRF) service that allows clients to receive pseudorandom function evaluations on their measurement and the server OPRF key, without revealing anything about their measurement. The clients use the output as randomness to produce the message that is then sent to the aggregation server. Using this framework allows STAR to provide strong privacy guarantees for clients, even if the measurement space has low entropy at the point when the aggregation takes place. The randomness server must be non-colluding with respect to the aggregation server, though these servers never have to communicate directly.

The full STAR protocol is specified in Section 4. We also describe an alternative form of STAR, "STARLite", that samples randomness only from the measurement itself. This approach is only suitable for sufficiently random data distributions, but removes the need for a distinct randomness server, further simplifying and reducing the costs of private data collection. See Section 7 for more discussion.

Trust assumptions. While STAR protocol is inherently multi-server, we note that the collaboration model is categorically weaker than previous cryptographic approaches such as [9, 10, 14], where multiple servers collaboratively compute the output of the aggregation. In effect, STAR provides the same trust dynamic as submitting plaintext measurements to an untrusted server over an anonymizing proxy (which also provides the randomness server functionality), but with the extra security guarantee that client measurements are hidden until κ-anonymity is provided, and with very little additional performance overhead.

Simple cryptography. STAR uses simple, well-established cryptographic tools, that have been used extensively by non-experts for many years. Previous proposals either use trusted hardware; non-quantifiable noise-based approaches; or novel, complex, and poorly understood cryptographic tools.

Performance. To confirm the practicality of STAR, we present and report on an open-source Rust implementation.[2] For processing server-side aggregation of $1,000,000$ client measurements, STAR requires only 20.01 seconds and a total of 222.21MB of communication, and computation is minimal. Overall, STAR is orders of magnitude cheaper to run than previous systems, see Section 6 for more details.

Standardization. STAR is compatible with the IETF's proposed framework for devising new privacy-preserving measurement systems [35].

1.2 Formal contributions

We make the following contributions:

- The design, systematization, and formalization of the STAR system, and associated privacy goals;
- An open-source Rust implementation of STAR, that is already used in wide-scale software deployments;
- Empirical evaluation of the STAR protocol, that showcases performance and simplicity far superior to previous constructions, while ensuring comparable privacy guarantees;

[1] Note that similar approaches were highlighted previously by Bittau et al. [8], but various complex issues were left as open problems to solve.

[2] https://github.com/brave-experiments/sta-rs

- Specific guidance for navigating trade-offs between additional privacy, and simpler deployment scenarios.

2 OVERVIEW OF DESIGN GOALS

In this section we clarify the problem statement that we are tackling, what constraints we are operating under, and subsequently a set of design goals and non-goals.

2.1 Problem statement

Primary goal. We aim to build a system that allows clients to submit measurements as encoded messages to an untrusted aggregation server. This aggregation server should be able to decode and reveal *only* those measurements that are sent by $\geq \kappa$ clients, where κ is a public parameter chosen by the aggregation server.

Auxiliary data. Clients should be able to send auxiliary data with their measurements, that can differ from client-to-client and is revealed only if the client's measurement satisfies the threshold aggregation policy.

2.2 Motivation and constraints

We aim to enable privacy-preserving threshold aggregation data collection through a protocol that both i) provides strong privacy guarantees, and ii) is practical for implementation and adoption by a wide range of projects and organizations; everything from small hobbyist projects to Web scale software. We particularly aim for a solution for projects that are not well served by existing state of the art (which requires non-trivial budgets, difficult-to-achieve trust assumptions and implementation expertise). To assess suitability, the following points and constraints are crucial to bear in mind.

Client privacy. Any protocol should provide formal guarantees of client privacy in a well-understood and coherent security model, with very limited leakage.

Correctness guarantees. Any solution must provide *correct* aggregation, rather than approximations that rely on receiving very large amounts of client data for providing high utility.

Low financial costs. Small projects usually run servers in standard cloud-based hardware such as Amazon Web Services (AWS), so financial costs can run up quickly. Thus, we can neither tolerate expensive cryptographic computation nor costly bandwidth consumption.

Achievable trust requirements. Data aggregation procedures that rely on multi-round interactions with a non-colluding partner are expensive to set up, run, and maintain.

Avoiding trusted hardware. Running aggregation in trusted hardware platforms, such as secure enclaves (such as Intel SGX) or cloud-based solutions (e.g. Amazon Nitro enclaves[3]), are usually prohibitively expensive and potentially vulnerable to attacks [32]. Overall, requiring trusted hardware significantly increases the complexity of any candidate system.

[3]https://aws.amazon.com/ec2/nitro/

Limiting cryptographic complexity. Avoiding novel cryptographic procedures, that are both expensive to run and require significant cryptographic expertise to implement, allows those with little cryptographic knowledge to implement applications safely and decreases the risk of disastrous privacy vulnerabilities.

2.3 Goals

With the above constraints in mind, we will design a threshold aggregation system with the following characteristics.

Concrete privacy guarantees. We will aim to provide similar privacy guarantees to existing threshold aggregation protocols, both in terms of concretely restricting the capabilities of an adversary to learn measurements that sent by less than the threshold number of clients, and in reducing any leakage that occurs. Overall, we will design a system and formal security model that provides client privacy, up to a concretely specified amount of limited leakage.

Minimal trust assumptions. We will develop a protocol that, at the very least, only involves a single aggregation server. This aggregation server must not require communication with any non-colluding parties, at least during the aggregation process. This categorically eases the overhead of building and maintaining practical deployments, and will significantly reduce bandwidth consumption.

Cheap running costs. Bandwidth usage must be minimal, along with any computation that is required. Ideally, we would like aggregation of 1 million client measurements to incur a cost of less than 1 dollar.

Simple, trusted cryptography. The cryptographic machinery that we use must be simple to understand by non-experts, and provide auditable security guarantees. Thus, we will avoid using any novel cryptographic primitives, that could lead to complicated and potentially vulnerable implementations.

2.4 Non-goals

Furthermore, we make clear that we are not attempting to solve any of the following problems.

Prevention of Sybil attacks. By their very nature, Sybil attacks [18] — where a malicious aggregation server injects clients into the system that send messages to try and reveal data from honest clients — are an unavoidable consequence of building any threshold aggregation system. Therefore, we will not be attempting to provide security for any client measurements that are targeted by such attacks. We will instead provide a security model that restricts the time window in which such attacks can occur (Section 4). Our solution will also be compatible with any typical higher-layer defenses that are typically used (such as identity-based certification [18]).

Leakage-free cryptographic design. All threshold aggregation systems that approach practical performance involve disclosing small amounts of leakage about client measurements that remain hidden. Combined with external public data, this leakage may become more useful in identity-linkage attacks. Rather than preventing leakage entirely, we will instead show

that the STAR approach provides a leakage profile that is comparable with recent work in this area (Section 4.6).

3 PRELIMINARIES

We provide the descriptions of each of the cryptographic primitives that are used for constructing the STAR protocol.

General notation. We use PPT to describe a probabilistic polynomial time algorithm. We use $[n]$ to represent the set $\{1, \ldots, n\}$. We use $x \| y$ to denote the concatenation of two binary strings. We write $\mathcal{X} \stackrel{c}{\approx} \mathcal{Y}$ for (computationally indistinguishable) distributions $\mathcal{X}$ and $\mathcal{Y}$ iff the advantage of distinguishing between $\mathcal{X}$ and $\mathcal{Y}$ for any PPT algorithm is negligible. We write $\mathcal{X} \stackrel{s}{\approx} \mathcal{Y}$ if $\mathcal{X}$ and $\mathcal{Y}$ are (statistically) indistinguishable for any algorithms (even if they run in exponential time).

Symmetric encryption. We will assume a symmetric key encryption scheme, ske, that consists of two algorithms:

- $c \leftarrow \mathsf{Enc}(k, x)$: produces a ciphertext c as the output of encrypting data x with key k;
- $x \leftarrow \mathsf{Dec}(k, c)$: outputs x as the decryption of c under key k.

We separately use derive to denote an algorithm that accepts a seed and a security parameter as input, and returns a randomly sampled encryption key. We will assume that, for randomly sampled keys, ske satisfies IND-CPA security.

Secret-sharing. We assume the usage of a κ-out-of-n threshold secret sharing scheme $\Pi_{\kappa,n}$ with information-theoretic security, operating in a finite field $\mathbb{F}_p$ for some prime $p > 0$. Such a scheme consists of two algorithms:

- $s \leftarrow \mathsf{share}(z; r)$: produces a random *share* $s \in \mathbb{F}_p$ of the value z, with explicitly specified randomness r;
- $(\bar{z}, \perp) \leftarrow \mathsf{recover}(\{s_i\}_{i \in [\ell]})$: outputs $\bar{z}$ when $\ell \geq \kappa$ and each s_i is a valid share of $\bar{z}$, otherwise outputs $\perp$.

For security, we require that any set of shares smaller than κ is indistinguishable from a set of random strings.[4] We call this property *share privacy*, and is achieved for secret sharing approaches based on traditional Shamir secret sharing [6].

Remark 1. *We require that p is large enough that randomly sampling values from $\mathbb{F}_p$ is highly unlikely to lead to collisions. Note that the size of p does not have any bearing on security.*

Oblivious pseudorandom function protocols. We assume the presence of a verifiable oblivious pseudorandom function (VOPRF) protocol denoted by voprf. Oblivious pseudorandom function (OPRF) protocols were first introduced by Freedman et al. [20]. They enable a client to receive PRF evaluations from a server, whilst the client input is kept secret, and nothing is revealed about the server PRF key. Verifiable OPRFs (VOPRFs) such as that of Jarecki et al. [24] provide clients with the ability to verify (in zero-knowledge) that the server has evaluated the PRF properly.

Following the description given by Tyagi et al. [39], we define a VOPRF, voprf, to have the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{voprf.setup}(1^\lambda)$: a server-side algorithm that produces public parameters pp for the VOPRF.
- $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{voprf.keygen}(\mathsf{pp})$: a server-side algorithm that samples a keypair based that is compatible with the public parameters pp.
- $(\mathsf{rq}, \mathsf{st}) \leftarrow \mathsf{voprf.req}(x)$: a client-side algorithm that produces a request rq and some state st, from some initial input $x \in \{0,1\}^*$;
- $\mathsf{rp} \leftarrow \mathsf{voprf.eval}(\mathsf{msk}, \mathsf{rq})$: a server-side algorithm that produces a response rp using a secret key msk, and a client request rq;
- $y \leftarrow \mathsf{voprf.finalize}(\mathsf{mpk}, \mathsf{rp}, \mathsf{st})$: produces the PRF output on msk and the input x encoded in rq, using the server response rp, public key mpk, and client state st.

We assume a VOPRF protocol that follows the standard ideal functionality, as laid out by Albrecht et al. [2]. Such VOPRFs have been shown to exist based on the One-More-Gap-Diffie-Hellman assumption, with security proven in the UC-security model [24].

It should be noted that there are numerous practical use-cases for (V)OPRF protocols and their variations [16, 23, 29, 39], alongside IETF standardization efforts [11, 15].

Min-entropy. For a distribution $\mathcal{D}$ over some input space $\mathcal{X}$, the min-entropy of $\mathcal{D}$ is defined as $\min_{x \in \mathcal{X}}(-\log_2(\Pr[X = x]))$.

3.1 Protocol security model

In Section 4.6, we describe an ideal functionality of the threshold aggregation protocol — including inputs, outputs, and potential leakage — and use it to show that any attack that is possible in the real world protocol is also possible to launch against the ideal world functionality. Intuitively, this proves that the protocol reveals nothing except what is revealed by the function output *plus* a bounded amount of leakage that is output by a specific leakage function.

Protocol security. The ideal functionality is denoted by $\mathcal{F}_\mathcal{P}$ for protocol $\mathcal{P}$. Let $\mathsf{inputs}_\mathcal{H}$ and $\mathsf{inputs}_\mathcal{A}$ denote the set of inputs chosen by both honest parties and the adversary $\mathcal{A}$, respectively. In addition, let $\mathsf{Real}(\mathcal{P}, \mathcal{A}; \mathsf{inputs}_\mathcal{A}, \mathsf{inputs}_\mathcal{H})$ denote the view of the adversary $\mathcal{A}$ in the real protocol, and $\mathsf{Ideal}(\mathcal{F}_\mathcal{P}, \mathcal{S}, \mathcal{A}; \mathsf{inputs}_\mathcal{A}, \mathsf{inputs}_\mathcal{H})$ the view of $\mathcal{A}$ when simulated by a PPT algorithm $\mathcal{S}$ that interacts with $\mathcal{F}_\mathcal{P}$. We say that $\mathcal{P}$ is secure against *malicious adversaries* if, for all choices of inputs, the following equation holds:

$$\mathsf{Real}(\mathcal{P}, \mathcal{A}; \mathsf{inputs}_\mathcal{A}, \mathsf{inputs}_\mathcal{H}) \stackrel{c}{\approx} \mathsf{Ideal}(\mathcal{F}_\mathcal{P}, \mathcal{S}, \mathcal{A}; \mathsf{inputs}_\mathcal{A}, \mathsf{inputs}_\mathcal{H}), \quad (1)$$

This security model is commonly referred to as proving security in the *real/ideal-world* paradigm.

Leakage. The leakage function specifies additional information that the adversary may learn during the protocol that is required for completing the simulation. This information mirrors real leakage that occurs during the protocol execution. We denote by L the leakage function that takes as input a set of inputs inputs (both honest and adversarial), and outputs some leakage $\mathsf{L}(\mathsf{inputs})$.

[4] As is common for secret sharing schemes, shared messages must be sufficiently unpredictable [6].

4 THE STAR PROTOCOL FRAMEWORK

4.1 Notation

We first recall the main participants, parameters, cryptographic tools, and notation that we use when describing the STAR protocol.

Participants and protocol parameters:

- κ is the threshold used for performing aggregation.
- $\mathcal{C}$ is the set of all clients $\{\mathbb{C}_i\}_{i\in[n]}$.
- $\mathbb{S}$ is the aggregation server.
- $\mathbb{O}$ is the randomness server in $\mathcal{P}$.
- We use $\mathcal{P}$ to refer to the STAR protocol, and $\widetilde{\mathcal{P}}$ to refer to STARLite (see Figure 2).

General notation:

- $\mathcal{D}$ be the distribution over universe $\mathcal{U}$, that clients sample their measurements from.
- $(c_i, \mathsf{aux}_i, t_i)$ is the message sent by $\mathbb{C}_i$, as defined in Figure 2.
- $\mathcal{X}$ is the set of all measurements received by $\mathbb{S}$, and let $\mathcal{X}_{\mathcal{H}}$ ($\mathcal{X}_{\mathcal{A}}$) denote the subsets of measurements received from honest (adversarial) clients.
- Let $\mathcal{E}_1 = (x_1, \mathsf{aux}_{x_1}, \kappa_1), \ldots, \mathcal{E}_\ell = (x_\ell, \mathsf{aux}_{x_\ell}, \kappa_\ell)$ correspond to each of the ℓ unique measurements in $\mathcal{E}$, along with the collection of auxiliary data aux_{x_1} sent by the clients that send x_1 number of times they are received by $\mathbb{S}$.
- $\mathcal{Y}$ is the set containing each $\mathcal{E}_\iota$ where $\kappa_\iota \geq \kappa$ that is output to $\mathbb{S}$.

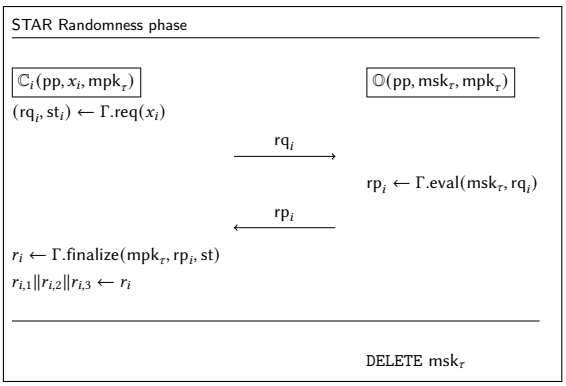
Cryptographic tools:

- Γ is a VOPRF (Section 3).
- $(\mathsf{msk}_\tau, \mathsf{mpk}_\tau)$ is the keypair of $\mathbb{O}$ for Γ.
- Σ is a symmetric encryption scheme satisfying IND-CPA security (Section 3).
- $\Pi_{\kappa,n}$ is a (κ, n)-secret-sharing scheme, and let $\mathbb{F}_p$ be the associated finite field with order $p \in \mathbb{Z}$ (Section 3).
- $\mathcal{A}$ is a malicious PPT adversary.
- $\mathcal{S}$ is a PPT simulator.
- $\mathcal{F}_{\mathcal{P}}$ is the ideal functionality corresponding to protocol $\mathcal{P}$, and L is the leakage function for $\mathcal{P}$ (Section 4.6).
- $\mathcal{F}_\Gamma$ is the ideal functionality corresponding to Γ.

4.2 Design space

We assume a large universe of elements $\mathcal{M}$ (e.g., bitstrings of ≥ 64 bits) representing potential *measurements* that clients send to a single, untrusted aggregation server. For example, such measurements may include profile information about a user (e.g. browser user-agent), or the set of applications installed on a device. Clients may *optionally* send arbitrary additional data with their measurement.

A single encoded measurements is sent during an *epoch* by each available client. The aggregation server should be able to reveal all those encoded measurements (and any associated data) that are received at least κ times. The threshold $\kappa \geq 1$ is agreed publicly from the outset.



STAR Message phase

Inputs : $x_i, \mathsf{aux}_i, (r_{i,j})_{j\in[3]}, \kappa$

Outputs : Client STAR message

$K_i \leftarrow \mathsf{derive}(r_{i,1})$
$s_i \leftarrow \Pi_{\kappa,n}.\mathsf{share}(r_{i,1}; r_{i,2})$
$c_i \leftarrow \Sigma.\mathsf{Enc}(K_i, x_i\|\mathsf{aux}_i)$
$t_i \leftarrow r_{i,3}$
return (c_i, s_i, t_i)

STAR Aggregation phase

Inputs : n Client STAR messages, κ

Outputs : List of measurements that occurred κ times

$\mathcal{Y} = [\,]$;
foreach $\mathcal{E}_\iota = \{(c_j, s_j, t_j) \mid (t_j = t_\iota \; \forall\, j)\}$:
  if $|\mathcal{E}_\iota| < \kappa$: **return** $\perp$
  $(\bar{c_\iota}, \bar{s_\iota}) \leftarrow \{(c_\iota, s_\iota) \mid (c_\iota, s_\iota) \in \mathcal{E}_\iota\}$
  $r_{\iota,1} \leftarrow \Pi_{\kappa,n}.\mathsf{recover}(\bar{s_\mathcal{E}})$
  $K_\iota \leftarrow \mathsf{derive}(r_{\iota,1})$
  foreach $c_j \in \bar{c_\iota}$:
    $x_\iota\|\mathsf{aux}_j \leftarrow \Sigma.\mathsf{Dec}(K_\iota, c_j)$
    if $(\iota \neq 0) \wedge (x_\iota \neq x_{\iota-1})$:
      return $\perp$
    $\mathcal{Y}[x_\iota].\mathsf{push}(\mathsf{aux}_j)$
return $\mathcal{Y}$

Figure 2: The STAR protocol for performing threshold aggregation of measurements. In the Randomness phase clients sample VOPRF randomness from $\mathbb{O}$ (in STARLite, randomness is derived locally by computing $r_{i,1}\|r_{i,2}\|r_{i,3} \leftarrow H(\mathsf{pp}, x_i, \tau)$ for some hash function H). The Message phase sees clients construct an encoded message corresponding to their measurement. In the Aggregation phase, $\mathbb{S}$ learn those measurements (and associated data) that are sent by $\geq \kappa$ clients.

4.3 STAR protocol

The STAR protocol is based upon the principle that clients sharing a measurement can devise compatible secret shares for a (κ, n)-secret-sharing scheme. Such shares could then be combined to reveal the measurement itself (and optionally any

additional data that they send) by an untrusted aggregation server. Once κ clients send a share of the same value, the server will be able to recover the hidden value (and any additional data that is sent).

The algorithmic description of the STAR protocol is given in Figure 2. We provide a description below as an overview of the entire exchange.

Randomness phase. Firstly, each client holding measurement x_i interacts with the randomness server, that runs a VOPRF service. Essentially, the client operates as the client in the VOPRF protocol with input x_i, and the randomness server answers the query and returns the result to the client. The client, after processing the VOPRF output to receive r_i, now has the result (x_i, r_i). Note that any client that shares the measurement x_i will also receive the same output r_i. See Section 3 for a description of the VOPRF exchange.

It is possible to construct a version of STAR that provides weaker security guarantees, in favor of dropping the requirement for the randomness server (which can leads to a much simpler pratical deployment). In particular in STARLite, the client simply samples r_i directly from their measurement (for example $r_i \leftarrow H(x_i)$, where H is a random-oracle model hash function) before proceeding directly to the message phase. The STARLite protocol only retains security when client measurements are sampled from a suitable high-entropy distribution, see Section 7 for more discussion.

Message phase. The message construction phase consists of the following steps.

- The client with (x_i, r_i) parses r_i into three parts $r_{i,1}, r_{i,2}, r_{i,3}$.[5]
- They derive a symmetric key K_i using a pseudorandom generator where $r_{i,1}$ is used as the seed.
- They construct a share s_i of $r_{i,1}$ using a κ-threshold secret-sharing scheme, using $r_{i,2}$ as the local randomness that is used in share generation process.
- They construct the ciphertext c_i as the encryption of their measurement x_i, and any auxiliary data that they would like to attach, using a symmetric encryption scheme with the previously-derived key K_i.
- Finally, they construct their message as (c_i, s_i, t_i), where $t_i = r_{i,3}$.

Aggregation phase. In the final aggregation phase the aggregation server receives a message from each of n clients, and learns which of the encoded measurements are shared by at least κ clients. The steps are as follows.

- The aggregation server groups together messages based on whether they share the same t_ι value into subsets $\mathcal{E}_\iota$.
- For any subset that contains at least κ messages, the aggregation server does the following:
  - runs the share recovery algorithm on each of the share values s_j to output $r_{\iota,1}$;
  - derives the encryption key K_ι from $r_{\iota,1}$;
  - decrypts each of the client ciphertexts c_j using K_ι, and groups together the measurement x_ι with the list of the auxiliary data objects, aux_j, sent by each client.
- Finally, the aggregation server creates a list $\mathcal{Y}$ of all measurements x_ι (along with the attached auxiliary data), that satisfy the threshold κ.

[5] This can be done for example by running $r_{i,j} = H(r_i \| j)$, for a random-oracle model hash function H.

4.4 Security Considerations

We detail a series of considerations related to the security of the STAR protocol design. The formal security model that we will use for proving security is given in Section 4.6, and the proofs are given in Appendix A.

Communication between servers. Note that the randomness and aggregation servers only communicate with the clients in the system, and only one performs the eventual aggregation. This is a significant improvement on existing multi-server solutions for threshold aggregation, where the servers are required to communicate with each other for processing the results of aggregation. Requiring communication between servers quickly drives up costs for both server operators, and tangibly weakens the extent to which both servers are non-colluding. This is because the server operators will have to work together to ensure that their servers can cooperate.

Randomness server key rotations. The usage of the randomness server in STAR ensures that an adversarial aggregation server must communicate with the randomness server to launch attacks on client inputs, but it does not immediately provide security to low-entropy inputs. Therefore, we consider a security model where clients sample randomness in epoch τ, and send their encoded measurement in epoch $\tau + 1$, after the randomness server has performed a key rotation. This limits the aggregation server to only launch online attacks on client inputs before epoch $\tau + 1$, having not yet seen any client messages, or observed any leakage. Once this key is deleted, it is not possible to launch queries that attempt to identify hidden client values. Moreover, by rotating this key before the aggregation phase takes place, this ensures that the $\mathbb{S}$ is only able to make use of any leakage that may occur *before* they witness any client measurements.

In the formal security model defined in Section 4.6, we encode this by forcing the adversary $\mathbb{S}$ to specify up front which values they would like to leak. Importantly, this disables the potential for an adversary to launch a targeted attack based on client identity, or any observed leakage.

Leakage. The leakage in the STAR protocol amounts to the aggregation server learning which clients share the same measurement — regardless of whether the measurement is kept hidden or not. Similarly, the adversary could launch a "Sybil" attack by establishing/corrupting clients with specifically-chosen measurements. As mentioned in Section 2.4, we consider prevention of Sybil attacks a non-goal, since all such threshold aggregation protocols are vulnerable to such attacks. However, we encode the possibility for an adversarial $\mathbb{S}$ to make use of this leakage into the formal leakage function that is defined as part of our security model in Section 4.6.

Predictable input distributions. Practical use-cases of STAR require that client messages remain somewhat unpredictable during the randomness phase of the protocol. If measurements are predictable, then the aggregation server may launch queries the randomness server for all such values during this phase, and then use the leakage to learn which clients are sending predictable values, even if less than κ clients send them. The advantage of STAR (as opposed to STARLite) is that this attack can only be carried out during the time-limited randomness phase (before the key rotation occurs), and that the attack must be carried out online. This allows extra external protection measures at the randomness server, such as identity-based rate-limiting and verification, to be used to make such attacks even more expensive.

Additional data. Before the protocol begins, $\mathbb{S}$ should inform clients of the maximum length of the additional data that should be sent. If aux_i is not equal to that length, then it must be truncated or padded depending on whether it is too long or short, respectively. We make no guarantees on the shape of auxiliary data for client measurements.

Hardening against local attacks in STAR. All hash function invocations in STAR can be replaced with functions that are deliberately slower primitives, such as PBKDF2 [25] and scrypt [33]. Such functions are used in applications handling passwords that hope to provide additional security against password-cracking adversaries. This change only impacts client computation in a small way, and would increase the difficulty for any adversarial aggregation server trying to reverse client encoded measurements. Moreover, such changes similarly increase the difficulty of attacks in case of a breakdown in the trust model used in STAR, or if using STARLite.

4.5 Reducing Leakage Via Oblivious Proxies

Identity leakage. As with many previous designs of threshold aggregation protocols, STAR produces a quantifiable amount of leakage. Importantly, the link between client identity and their input is unbroken.

In some applications maintaining this link is useful. Consider an aggregation server that is attempting to learn which clients may be part of a fraudulent botnet of a threshold size, by having clients submit information about their browser profile. In such cases, it is essential to link client identity to their sent messages, so that the aggregation server can subsequently disqualify malicious clients.

However, if an aggregation server is merely trying to learn client diagnostic information, it is unlikely that maintaining this link is useful or necessary.

Oblivious proxies. One method for eliminating such leakage in STAR is using tools for performing anonymous value submission at the application-layer — destroying the link between client identity and their messages. For example, by submitting measurements via an oblivious/anonymizing proxy that strips client identifying information (such as IP addresses) from HTTP requests containing client measurements, the aggregation server learns nothing about the client identity (Figure 3). Well-known tools exist for this purpose such as

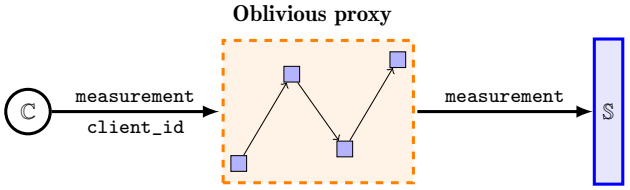


Figure 3: Oblivious proxy for submitting client ($\mathbb{C}$) measurements to the aggregation server ($\mathbb{S}$).

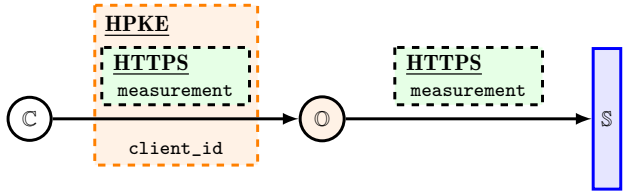


Figure 4: Oblivious HTTP flow including usage of hybrid public key encryption (HPKE) for message encapsulation. Here, $\mathbb{O}$ is the *proxy resource* [38]. This entity can be implemented in STAR using the randomness server $\mathbb{O}$, since the client messages are protected with TLS.

Tor[6] (or certain VPNs) can be used. However, using Tor comes with well-known performance overheads that would slow down client requests in STAR considerably [36].

Oblivious HTTP. An alternative mechanism known as Oblivious HTTP (OHTTP) that has been proposed as a draft standard to the IETF [38] performs similar anonymization of HTTP requests as Tor, but with fewer intermediate hops — promising a smaller performance overhead. The oblivious proxy is a single party known as the *proxy resource*, and the aggregation server plays the part of a *target resource* that receives the client message [38].

Figure 4 provides a diagrammatic representation of the OHTTP flow in the context of STAR. In essence, the client encapsulates a HTTP request containing their message to the aggregation server using *hybrid public key encryption* (HPKE) [3], where encapsulation is performed under the public key of the oblivious proxy. The client sends this encapsulated message as the body of a separate HTTP request to the oblivious proxy. The proxy decapsulates the message and forwards it on to the aggregation server, without including any client identifying information.

Since client messages to the aggregation server are protected by TLS the oblivious proxy has no way of reading the client messages. As a result, this oblivious proxy can be instantiated using the existing randomness server $\mathbb{O}$ in STAR without compromising any of the security goals, and without requiring any additional non-colluding parties. Note that this means that the randomness server must explicitly send a message to the aggregation server, whereas the original STAR

[6] https://www.torproject.org/

protocol requires no communication between these two entities. This communication is minimal and not related at all to the cryptographic logic that is run in the aggregation server. Even so, operators that prefer to avoid any communication taking place between these servers can simply submit data over existing anonymizing proxies like Tor, or would require the oblivious proxy to be run by a different party.

The Oblivious HTTP Internet standards draft defines specific guarantees that must be upheld by the anonymizing proxy, as well as request formats [38, Appendix A]. Such proxies are already intended to be standardized by the IETF, and to be run by independent entities[7] for privacy-preserving measurement aggregation systems [35].

4.6 Formal Security Model

We now provide the security model for the establishing the security of STAR. See Appendix A for proofs of security for the STAR protocol, with respect to the following model.

Ideal functionality. The ideal functionality below represents the inputs, outputs, and internal steps of the threshold aggregation functionality. We will write $\mathcal{F}_{\mathcal{P}}$ to denote this functionality, where $\mathcal{P}$ is the STAR protocol.

- Participants: aggregation server $\mathbb{S}$, randomness server $\mathbb{O}$, clients $\{\mathbb{C}_i\}_{i\in[n]}$.
- Public parameters: upper bound on n.
- Functionality:
  - $\mathbb{O}$ inputs the VOPRF keypair $(\mathsf{msk}_\tau, \mathsf{mpk}_\tau)$.
  - Each client $\mathbb{C}_i$ ($i \in [n]$) provides their input (x_i, aux_i).
  - Let $\mathcal{E}_\iota = \left\{(x_\iota, \{\mathsf{aux}_j\}_{j\in J}, \kappa_\iota) \;:\; (J \subseteq [n]) \wedge (x_j = x_\iota)\right\}$ for each unique x_ι received, where $\kappa_\iota = |\{\mathsf{aux}_j\}|$ is the number of client measurements collected in $\mathcal{E}_\iota$.
  - Let $\mathcal{Y}$ be an empty map.
  - For each $\mathcal{E}_\iota$ where $\kappa_\iota \geq \kappa$, set $\mathcal{Y}[x_\iota] = \mathcal{E}_\iota$.
  - Output $\mathcal{Y}$ to $\mathbb{S}$, output $\{\mathcal{F}_\Gamma(\mathsf{msk}_\tau, x_i)\}_{i\in[n]}$ to $\mathbb{O}$ (where $\mathcal{F}_\Gamma$ is the ideal functionality defined for Γ), and output $\perp$ to each $\mathbb{C}_i$.

Overall, this ideal functionality captures the fact that the aggregation server learns all client measurements that are sent by at least κ clients. The randomness server learns what it would normally learn during the VOPRF exchange, and each client learns nothing.[8]

Leakage function. We use the leakage function (L) defined below to account for additional protocol leakage that occurs while running STAR. Assume that the aggregation server $\mathbb{S}$, and some subset $\mathcal{T} \subset \mathcal{C}$ of all clients is controlled by an adversary $\mathcal{A}$. The view of $\mathcal{A}$ can be simulated using the following leakage function.

- Receive $\mathcal{W} \leftarrow \mathcal{A}$, a set of disqualified clients specified by $\mathcal{A}$.
- Receive $\mathcal{X}_\mathcal{A} \leftarrow \mathcal{A}$, a set of input measurements specified by $\mathcal{A}$
- Let $\mathcal{Q} = \mathcal{C} \setminus \mathcal{W}$ be the set of remaining honest clients.
- Receive (x_i, aux_i) from each $\mathbb{C}_i \in \mathcal{C}$.
- Partition the set $\{(x_i, \mathsf{aux}_i)\}_{i\in[|\mathcal{Q}|]} \cup \mathcal{X}_\mathcal{A}$ into $\mathcal{N}_1, \ldots, \mathcal{N}_\ell$, where $\mathcal{N}_\iota$ is the set of all pairs that share the same measurement x_ι (for ℓ unique measurements).
- Leak $|\mathcal{N}_\iota|$ to $\mathcal{A}$, for each $\iota \in [\ell]$.

We write $\mathsf{L}(\mathcal{X}_\mathcal{H})$, where $\mathcal{X}_\mathcal{H}$ is the set of all measurements received from honest clients, to denote the output of L on $\mathcal{X}_\mathcal{H}$. Overall, this leakage function captures the fact that an adversary that controls $\mathbb{S}$ learns the cardinality of clients that share each unique measurement that is received.

Note that the leakage function explicitly does not capture the notion of client identity, since we assume that client measurements are submitted anonymously. This can be achieved using various practical solutions (Section 4.5).[9]

Security proofs. All correctness and security proofs are described in Appendix A.

5 FUNCTIONALITY AND LEAKAGE COMPARISON

5.1 Ideal functionality

A coarse-grained comparison of the functionality provided in STAR with previous approaches is given in Figure 5. All performance costs are asymptotic, see Section 6 for the concrete costs of running STAR. Overall, the solutions that offer the closest functionality, while still retaining close to practical performance, are the private heavy hitters protocols of [4, 5, 10, 12, 34, 40]. Protocols based on MPC involve very complex cryptographic implementations and expensive overheads [17]. Protocols that utilize trusted proxies and hardware require clients to place trust in computing platforms and entities that are not immune to security failures [32].

5.2 Leakage

An ideal solution to the threshold aggregation problem would provide information that can be derived from the output of the ideal functionality alone. In other words, only those measurements that are received from κ clients. While some schemes are able to achieve this notion [8, 9, 14, 27], they typically fall short of providing practical solutions.

Recent approaches for efficiently learning κ-heavy-hitters [4, 5, 10, 12, 34, 40] incorporate some amount of leakage, that provides additional information to the adversary. Specifically, each scheme leaks all the κ-heavy-hitting prefixes of the eventual κ-heavy-hitter measurements. As an example, consider clients that sent a measurement corresponding to their birth country. Assume that $\kappa = 4$, and that five clients send `"United States of America"`, four send `"United Kingdom"`, and three send `"United Arab Emirates"`. Then the ideal functionality suggests that the aggregation server should only learn that five clients sent `"United States of America"`, and four sent `"United Kingdom"`. However, additional leakage informs the server that twelve clients sent the prefix `"United"`. While such leakage may not always

[7] IETF OHAI: https://datatracker.ietf.org/group/ohai/about/

[8] For STARLite (denoted by $\widetilde{\mathcal{P}}$) we may define an alternative functionality, that takes no input from the randomness server.

[9] The leakage function could also be trivially updated to capture this additional leakage, if anonymous submission is not possible.

Protocol	Single-round interaction with clients	Bandwidth	Client computation	Aggregation computation	Single-server aggregation	Associated data	Negligible correctness errors	Fail-safety
Proxy-based shuffling [8, 13, 31]	✓	$O(n)$	$O(1)$	$O(n)$	✗	✓	✓	✗
Kissner et al. [27]	✗	$O(mn\lambda)$	$O(n^2)$	$O(mn\lambda)$	✓	✗	✓	✗
Blanton et al. [9]	✓	$O(mn\lambda)$	$O(n^2)$	$O(mn^2\lambda)$	✗	✗	✓	✗
Randomized response [4, 5, 12, 34, 40]	✗	$O(n\lambda)$	$O(1)$	$O(n)$	✓	✗	✗	✓
Boneh et al. [10]	✓	$O(mn\lambda)$	$O(\lambda)$	$O(mn\lambda\kappa)$	✗	✗	✓	✗
STAR (Section 4)	✓	$O(n\lambda)$	$O(\lambda)$	$O(n\lambda\kappa^2)$	✓	✓	✓	✓

Figure 5: Coarse-grained comparison of STAR against previous work. We use λ to denote the security parameter, $n = |C|$ to denote the number of clients, and m to denote the number of servers that are used in multi-server settings. Note that we ignore generic MPC techniques for computing threshold aggregation due to well-established performance limitations [17]. We also do not include Prio-like protocols [1, 14] as they are not compatible with string-based data.

be useful, in this example this effectively leaks how many clients also sent the answer `"United Arab Emirates"` (since no other country begins with the `"United"` prefix).

While STAR avoids prefix-based leakage, it leaks the subsets of clients that share equivalent measurements. In other words, the server can separate client messages into groups that all share the same measurement. This can be especially damaging in situations where the adversary launches a "Sybil" attack and injects their own measurements to try and learn how many times the same measurement is submitted. As mentioned previously, "Sybil" attacks are ultimately possible against any threshold aggregation scheme (even those that do not permit any leakage), and so this is not unique to STAR. Separately, such leakage could allow for measurement inference-based attacks that utilize the counts of each received message to attempt to infer encoded measurements.

Finally, it should be noted that the single-server aggregation mechanisms of STAR and those based on randomized response [4, 5, 12, 34, 40] naturally allow linking client messages to revealed measurements. Such leakage can be eliminated using anonymizing proxies for submitting client messages (Section 4.5). This approach has already been recommended for submitting measurements as part of ongoing standardization work in this area [35].

6 PERFORMANCE EVALUATION

We provide an open-source Rust implementation of all the necessary components for establishing the performance of STAR.[10] We benchmark the runtimes for both constructing client messages, and running the server aggregation process. We estimate the overall bandwidth costs as a result of client's interacting with both the aggregation and randomness servers. Finally, we provide runtimes and communication costs for performing anonymization of STAR messages via the Oblivious HTTP framework [38]. Overall, STAR is exceptionally efficient, even when processing 1 million measurements, and orders of magnitude cheaper than competing approaches.

6.1 Implementation Details

Secret-sharing implementation. Our secret sharing implementation is based on the Adept Secret Sharing (ADSS) framework developed by Bellare et al. [6] for achieving stronger guarantees on privacy and authenticity of shares.

As noted previously, we require implementation of a prime-order finite field for secret sharing that is large enough to make the occurrence of collisions a low probability event to ensure correctness. We choose two prime-order fields — one with a modulus of 255 bits in length ($\mathbb{F}_{255}$), and one that is 129-bits ($\mathbb{F}_{129}$) — and provide performance for both. In a practical sense, we consider the change of collisions in either field to be negligible. We assume that all inputs that are shared are 16 bytes in length (randomness for deriving symmetric encryption keys), so that they can be stored in a single share polynomial for either choice of finite field.

Finally, we note that secret share recovery uses only a subset of of κ shares. This means that we do not check whether all client shares are well-formed, but we do perform checks on the decrypted result for *all* of them. For example, if we receive 200 shares for a given measurement, with $\kappa = 100$, we will only perform recovery using a subset of 100 shares.

Oblivious HTTP proxy. We use an open-source Rust implementation for constructing an Oblivious HTTP proxy[11] that is compliant with the most recent IETF standards draft [38], as described in Section 4.5. Our setup assumes that client messages are sent via a *proxy resource*, run by $\mathbb{O}$, to a *target resource*, run by $\mathbb{S}$ [38]. Note that sending such messages via $\mathbb{O}$ is compatible with our approach since such messages are encrypted over a TLS connection that is negotiated with $\mathbb{S}$. This ensures that we do not introduce any additional trust assumptions to the STAR protocol. Encapsulation and decapsulation are performed using HPKE, with ciphersuite `DHKEM(X25519,HKDF-SHA256)` [3].

Other cryptographic machinery. We implement the VOPRF construction detailed by Tyagi et al. [39], with 128 bit security. The VOPRF is implemented using the ristretto255 prime-order group abstraction.[12] All hash functions are implemented using SHA-256. All symmetric encryption is implemented using AES-GCM AEAD with 128-bit keys.

[10] https://github.com/brave-experiments/sta-rs

[11] https://github.com/martinthomson/ohttp

[12] https://github.com/dalek-cryptography/curve25519-dalek

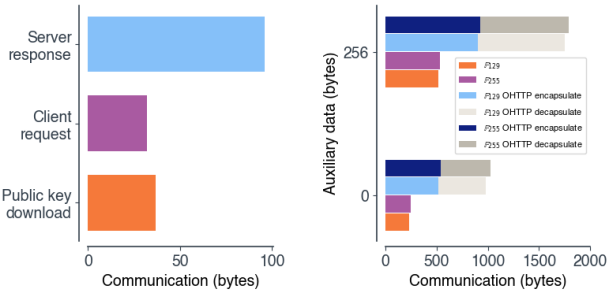


Figure 6: Left: Communication with the randomness server during the randomness sampling phase of STAR. Right: Communication during the STAR aggregation phase. Performance is compared for the two fields $\{\mathbb{F}_{129}, \mathbb{F}_{255}\}$ used in secret sharing, depending on whether OHTTP is utilized, and depending on how much auxiliary data is sent (either 0 or 256 bytes) with each measurement.

Client measurement sampling. All client inputs are sampled as 256-bit strings from a Zipf power-law distribution with a support of $N = 10,000$ and parameter $s = 1.03$. This matches the experimental choices made in [10], and captures a large proportion of applications. This distribution occurs naturally in many network-based settings [28] and, as highlighted in [10], the chosen parameters are chosen conservatively in that the distribution is closer to uniform than would typically be expected. In addition, we measure the costs of STAR in the two cases where clients append either zero or 256 bytes of auxiliary data to the measurement that they send.

Benchmarking. All benchmarks are run using an AWS EC2 `c4.8xlarge` instance with 36 vCPUs (3.0 GHz Intel Scalable Processor) and 60 GiB of memory.

6.2 Communication Costs

Randomness server. In STAR, the client must request randomness from the randomness server, which amounts to requesting a VOPRF evaluation on their measurement. We assume that the epochs are known apriori by both client and server, and that there are seven in total (allowing a daily epoch rotation and weekly full key rotation). This means that the client must download eight compressed curve points for the server public key at the start of the key cycle, and thus amortizes this cost to $(8/7)\cdot$`compressed_ec_point_len` bytes per epoch. The size of a client request is a single compressed elliptic curve point, and the response is a single curve point, plus two field scalars for the DLEQ proof. The total amortized per-client communication costs are given in Figure 6.

Aggregation server. The raw communication costs between clients and the aggregation server consist of a single encrypted ciphertext, a secret share, and a 32-byte tag. The size of the share is dependent on the size of the field that is used. The size of the ciphertext is dependent on the size of the auxiliary data that is appended to the client measurement. If client measurements are sent via the OHTTP proxy, then there are two HTTP requests: one containing an encapsulated HTTP request to the *proxy resource* and another corresponding to the decapsulated request to the aggregation server. We provide per-client communication costs in Figure 6. Note that for constructing OHTTP requests, we use an encapsulated HTTP request containing the following information:

- HTTP status line: e.g. `GET /hello.txt HTTP/1.1`;
- `User-Agent`, `Host`, and `Accept-Language` HTTP headers with default values given for each;
- `X-STAR-Message` header containing base64 encoded STAR protocol message (Section 4).

VOPRF setup	VOPRF evaluation	Proof generation
0.547	0.662	0.166

Figure 7: Randomness server single-threaded runtimes (ms).

VOPRF blind	VOPRF final	VOPRF verification	Aggregation message	
			$\mathbb{F}_{129}$	$\mathbb{F}_{255}$
0.081	0.093	0.301	0.019	0.02

Figure 8: Client runtimes (ms) during the STAR protocol.

6.3 Computational Costs

Client message construction. In Figure 8, we summarize the various costs of the cryptographic operations required for each individual client in STAR. Clearly client-side operations are highly performant. The most expensive client operations are the computation of two exponentiations in the elliptic curve group that is used. Therefore, we can reasonably expect that the STAR protocol can be leveraged even for clients with severely limited computation boundaries. The runtimes of the randomness server in STAR are given in Figure 7.

Aggregation server. Figure 9 considers the cost of the entire server aggregation phase for up to 1,000,000 clients, with κ taken from $\{0.01\%, 0.1\%, 1\%\}$ of this number. For 1,000,000 clients with $\kappa = 0.1\%$, the runtime of the aggregation server is only $20.01s$ using $\mathbb{F}_{129}$, and $73.65s$ for $\mathbb{F}_{255}$. Generally, when reducing the underlying field size ($\mathbb{F}_{129}$) we see runtimes reduce by a factor of around 3×. This clearly indicates that the STAR protocol is suitable for processing aggregations on very regular (sub-daily) reporting schedules. Note that the absolute size of the threshold has a noticeable impact on the runtime performance, due to the quadratic overhead of running Lagrange interpolation. This leads to quadratic growth of runtimes with respect to the threshold.

Oblivious HTTP proxy. Finally, we provide benchmarks in Figure 10 for running HPKE encapsulation and decapsulation of client messages by the OHTTP proxy. The OHTTP proxy is only required for reducing client identity leakage.

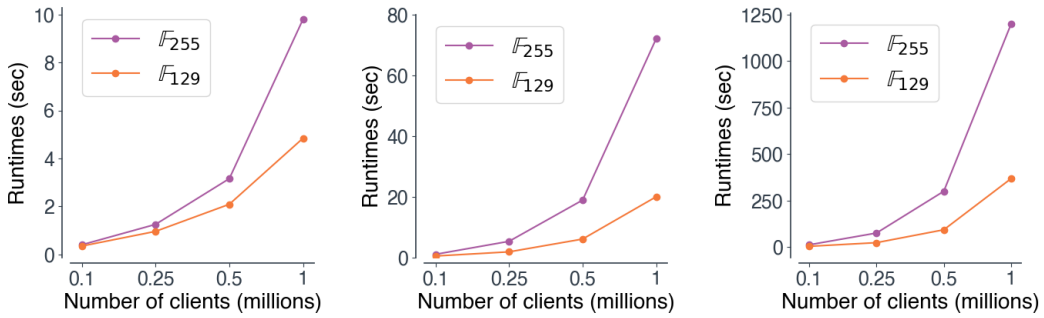


Figure 9: Aggregation server computation runtimes (seconds) based on number of clients. Graphs from left-to-right corresponding to a threshold $\kappa \in \{0.01\%, 0.1\%, 1\%\}$ of total number of client inputs. Performance is compared for both fields $\{\mathbb{F}_{129}, \mathbb{F}_{255}\}$.

Client setup	Server setup	Encapsulate	Decapsulate
0.131	0.106	0.002	0.002

Figure 10: Runtimes (ms) for performing single-threaded HPKE setup, encapsulation, and decapsulation at the OHTTP proxy, using the `DHKEM(X25519, HKDF-SHA256)` ciphersuite.

6.4 Comparison With Prior Approaches

We compare STAR directly with the performance results of the work of Boneh et al. [10], that devises a private heavy-hitters protocols from distributed point functions. As shown in Figure 5 and mentioned previously, alternative approaches (such as those based on randomised response, MPC, and shuffling) do not provide satisfactory performance or functionality.

To ensure that the leakage profile is similar in both STAR and [10], we compare STAR performance whilst including overheads for running the OHTTP proxy. From a functionality perspective, the protocol of [10] does not allow clients to specify auxiliary associated data, and thus is not as expressive as the STAR protocol. For this reason we only consider communication costs when auxiliary data is not sent. Moreover, STAR requires only a single aggregation server, while their aggregation phase requires two server instances. Finally, the client input distribution parameters are identical.

Communication. STAR (using $\mathbb{F}_{129}$) requires: 133 bytes of public key data to be downloaded by the client from the randomness server; 32 bytes to be sent by the client to the randomness server; 983 bytes to be sent from the client to the aggregation server (via the OHTTP proxy), of which only 464 bytes is received by the aggregate server, and 519 bytes is received by the OHTTP proxy. This gives a total 1148 bytes per client. The protocol of [10] requires approximately $70KB$ of communication per client. Therefore, overall communication in STAR is **62.4× smaller** than in [10]. Using $\mathbb{F}_{255}$ instead, per-client communication in STAR only increases by 20 bytes.

Runtimes. STAR vastly improves on the runtimes of [10] — using $\mathbb{F}_{129}$ as the base secret sharing field, and $\kappa = 0.1\%$ (the same value used by [10]) of 100,000 clients, STAR performs server-side aggregation in 0.467 seconds (and 1.03 seconds using $\mathbb{F}_{255}$). Moreover, times scale reasonably: for 500,000 clients, STAR performs server-side aggregation in 6.06s; for 1 million clients, it takes 20s.[13] In contrast, the [10] protocol takes 828.1s to perform an aggregation of data from 100,000 clients, and 54 minutes for 400,000 clients. Thus, the aggregation phase is **1773× faster** in the STAR protocol.

Clients messages take 0.628ms to construct, including interactions with the randomness server and HPKE encapsulation. The randomness server operations take 0.828ms per client input; setup costs occur once and can thus be amortized across all client messages. The cost of running the HPKE proxy is 0.002ms per client input. These times can be distributed across the epoch, and requests can be answered in parallel.

Financial costs. Finally, taking the costs of running an AWS EC2 `c4.8xlarge` at the time of writing, it costs $1.591 per hour of runtime, plus $0.09 per GB of data transferred out, and $0.02 per GB of data transferred in.[14] We summarize the monetary costs for both protocols in Figure 11. Communication costs are calculated by considering all data transferred in and out of EC2 instances, and computation costs by considering computation per hour.[15] The total costs of running all the components in STAR are **$0.00409+$0.037+$0.0053 = $0.04639**, which is more than **24×** cheaper than the cost of running the Boneh et al. [10] protocol (**$1.1152**). Notice that STAR remains cheaper than this benchmark even when aggregating data from 1,000,000 clients, costing **$0.4727** to run. Since the monetary costs of running [10] are expected to scale similarly linearly, we expect that STAR will remain significantly cheaper beyond 1,000,000 clients as well.[16]

[13] Using $\mathbb{F}_{255}$, aggregations of data from 500,000 and 1 million clients take 18.9s and 72.1s, respectively.
[14] February 2022
[15] In [10], computational is doubled due to the two-server setup.
[16] Dominant financial costs for STAR relate to bandwidth usage, which scale linearly, rather than aggregation computation time.

Cost	Boneh et al. [10]	STAR		
	Aggregation	Aggregation	VOPRF	OHTTP proxy
Comms in	$0.6193	$0.00389	$0.00027	$0.00435
Comms out	$0.13	—	$0.00017	$0.00086
Computation	$0.3659	$0.0002	$0.03659	$0.00009
Total cost	**$1.1152**	**$0.00409**	**$0.03703**	**$0.0053**

Figure 11: Monetary costs associated with running both STAR and [10], for aggregating $100,000$ client measurements. All costs include communication from clients and, in the case of [10], communication between aggregation servers. Costs for STAR include the additional costs associated with running the randomness server and OHTTP *proxy resource*. All costs are derived from Amazon EC2 `c4.8xlarge` costs at time of writing (November 2021).

7 DISCUSSION

7.1 Candidate Input Distributions for STARLite

The STARLite protocol must only be used when client inputs that are *not* eventually revealed are sufficiently entropic; client inputs that *are* revealed can be drawn from predictable distributions (Appendix A.3). Large *heavy-tailed* distributions, with correspondingly small thresholds that ensures the distribution tail has sufficient min-entropy, appear suitable for ensuring enumeration attacks are difficult.

It was noted in [8] that full URLs form a large, unpredictable search space. Other wide distributions include the IPv6 address space, which is 64 bits long, and if clients are submitting their own IP addresses these are likely to be unpredictable and not shared by other clients. Finally, STAR allows for multiple messages, sampled from independent distributions, to be concatenated together into a single message. Concatenating enough independently distributed messages can lead to a distribution that derives enough entropy from each of the underlying distributions to construct a secure client message. Finally, the ability of an aggregation server to perform local attacks can be restricted by using deliberately slower cryptographic algorithms, as discussed in Section 4.

We reemphasize that extreme care should be taken when using STARLite, since making categorical arguments about the entropy present in a real-world input distribution is very difficult. In most cases, using STAR is the safest option and comes with very small additional overheads.

7.2 Limitations

One limitation of STAR is that leakage can only be eliminated using application-layer solutions that anonymize client messages to the aggregation server (i.e. via an anonymizing proxy). However, note that some applications (such as those that involve checking for client-side fraud) may not want to elide such leakage, and thus STAR maintains flexibility. A further limitation is that STAR cannot provide security for small message spaces, since this would allow a malicious aggregation server to enumerate all possible client inputs before it has received them, via interaction with the randomness server. This limitation is also possible to exploit in prior systems but with attack complexity equal to $n \cdot \kappa$, rather than n in STAR. Finally, as is the case for all threshold aggregation systems, STAR remains vulnerable to Sybil attacks. Preventing such attacks is out-of-scope for this work, beyond showing that STAR is robust against adversarial clients to the extent that their only power is in choosing arbitrary inputs (Theorem 5).

8 RELATED WORK

We summarize a number of prior approaches that aim to preserve client privacy during threshold aggregation.

Data shuffling. Systems such as Prochlo [8] construct a data pipeline for clients to provide measurements whilst maintaining crowd-based privacy. Clients send their data to an initial server that strips identifying information and collates measurements into groupings.[17] Once groupings are large enough, the data is shuffled and sent to a processing server that can perform general post-processing. Unfortunately, these pipelines rely on honest execution of each of the pipeline steps by non-colluding servers, or by trusted hardware and software enclaves. Similar approaches using mix-nets [13] and verifiable shuffling [31] provide better security guarantees, but require increased interactivity to ensure privacy for thresholds greater than one.

Generic multi-party computation. Generic multi-party computation (MPC) protocols can be leveraged to compute threshold aggregation functionality over data from multiple clients [9, 27]. In this context, the server only learns those values which are shared with it over κ times. Such protocols can be computed directly between clients and servers using generic two-party computation that ensures malicious security. Some proposals focus on performing oblivious RAM computations during client-server interactions [21, 22, 26, 30]. Unfortunately, such protocols remain impractically expensive for real-world systems [17]. Moreover, such schemes require heavily-involved implementations for instituting the online (and multi-round) communication and computation patterns.

Outsourced computation. Private heavy-hitters protocols provide threshold aggregation functionality but with improved privacy: client identity is inherently decoupled from input submission.[18] A promising, recent construction explored by Boneh et al. [10] requires clients to secret-share or *distribute* a point function (evaluating to 1 on their chosen value, and

[17] This process is compatible with adding differential privacy.

[18] Note that such leakage can be addressed in higher-level applications by removing client identifying data from requests that contain input data, see Section 4.5 for more details.

0 elsewhere) between two aggregation servers. These servers then combine shares of multiple point functions obliviously and reveal the heavy-hitters among the client values. Overall, for $400,000$ clients each holding a 256-bit string, it takes the two servers 54 minutes to compute the κ-heavy-hitters (where $\kappa = 0.1\%$ of all clients) in the dataset, requiring $70KB$ total communication per client. The [10] approach leaks all heavy-hitting prefixes, and more generally all information leaked by the multi-set of honest client inputs. This information can be restricted by using local differential privacy.

Outsourcing of said computations had been explored previously in using > 2 servers, which then interact with each other to compute the eventual output [9, 27]. Such constructions lead to computation complexities that are quadratic in the number of client inputs, and require usage of notably heavier cryptographic primitives. While more efficient approaches do exist, such as Prio [1, 14], they only allow numerical inputs, and still incur overheads that are infeasible for building efficient threshold aggregation systems [10].

Single-server frameworks for private heavy-hitters. Randomized response based on local differential privacy (LDP) can provide private heavy-hitter aggregation that is computed only by a single server [4, 5, 12, 34, 40]. The major downside of these approaches is that they do not provide satisfactory correctness guarantees in all situations (a non-negligible amount of errors may occur). In particular, when the number of clients is anything but very large, then the amount of noise introduced is likely to heavily skew the correctness of the aggregation.[19] Furthermore, since the utility of the system is highly dependent on the privacy parameter and the number of clients, a system built upon randomized response requires each operator to make informed decisions about whether the correctness signal is strong enough for their application. In addition, solutions based on randomized response leak a non-negligible amount of information about each client's private value, since they also include prefix-based leakage similar to the heavy hitter protocol of [10]. We prefer to focus on building a system that provides perfect correctness and concrete security guarantees, without having to consider how to make security parameterizations or the number of clients.

More generic approaches for achieving randomized response, such as systems like RAPPOR [19], require clients to send a number of bits that is similar in size to the entire universe of possible input measurements. As a result, such techniques are infeasible for situations where this universe is very large.

Secret sharing of client data. The STAR construction has similar properties to parts of the secret sharing approach used by Apple, in their concurrent work to prevent the spread of Child Sexual Abuse Material (CSAM) on Apple devices [7]. Similarities appear in the manner that clients construct messages to the aggregation server — using a secret sharing approach to share media from each of their devices. However, the Apple approach does not extend to a distributed setting, and only operates across a single client's shares. Our work tackles the broader question of how clients can non-interactively agree on compatible secret shares in a distributed system, allowing recovery of messages that are shared by a threshold number of clients. The wider system and application are also significantly different.

[19] When the number of clients is very large, the noise that is introduced will be relatively small in comparison to the signal.

9 CONCLUSION

In this work we build STAR: a simple, practical mechanism for threshold aggregation of client measurements. We intend STAR to enable privacy-protecting, user-respecting data collection practices that were not practical or affordable given the existing state of the art. STAR is orders of magnitude cheaper, easier to understand, and easier to implement (in terms of code and trust requirements) than existing systems. We provide a tested, open source implementation of STAR[20] in rust that can be used in projects today. We hope that STAR will result in analytics and usage data collection being more private, for more users, benefiting more analytics frameworks.

ACKNOWLEDGEMENTS

The authors would like to thank Eric Rescorla, Subodh Iyengar, Ananth Raghunathan, and anonymous reviewers for their helpful feedback on this work.

REFERENCES

[1] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. 2021. Prio+: Privacy Preserving Aggregate Statistics via Boolean Shares. Cryptology ePrint Archive, Report 2021/576. https://eprint.iacr.org/2021/576.

[2] Martin R. Albrecht, Alex Davidson, Amit Deo, and Nigel P. Smart. 2021. Round-Optimal Verifiable Oblivious Pseudorandom Functions from Ideal Lattices. In *PKC 2021, Part II (LNCS, Vol. 12711)*, Juan Garay (Ed.). Springer, Heidelberg, 261–289. https://doi.org/10.1007/978-3-030-75248-4_10

[3] Richard Barnes, Karthikeyan Bhargavan, Benjamin Lipp, and Christopher A. Wood. 2021. *Hybrid Public Key Encryption*. Internet-Draft draft-irtf-cfrg-hpke-12. IETF Secretariat. https://www.ietf.org/archive/id/draft-irtf-cfrg-hpke-12.txt https://www.ietf.org/archive/id/draft-irtf-cfrg-hpke-12.txt.

[4] Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Thakurta. 2020. Practical Locally Private Heavy Hitters. *Journal of Machine Learning Research* 21, 16 (2020), 1–42. http://jmlr.org/papers/v21/18-786.html

[5] Raef Bassily and Adam D. Smith. 2015. Local, Private, Efficient Protocols for Succinct Histograms. In *47th ACM STOC*, Rocco A. Servedio and Ronitt Rubinfeld (Eds.). ACM Press, 127–135. https://doi.org/10.1145/2746539.2746632

[6] Mihir Bellare, Wei Dai, and Phillip Rogaway. 2020. Reimagining Secret Sharing: Creating a Safer and More Versatile Primitive by Adding Authenticity, Correcting Errors, and Reducing Randomness Requirements. *PoPETs* 2020, 4 (Oct. 2020), 461–490. https://doi.org/10.2478/popets-2020-0082

[7] Abhishek Bhowmick, Dan Boneh, Steve Myers, Kunal Talwar, and Karl Tarbe. July 29, 2021. The Apple PSI System, Apple Inc. https://www.apple.com/child-safety/pdf/Apple_PSI_System_Security_Protocol_and_Analysis.pdf (accessed 19 Aug 2021).

[8] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. 2017. Prochlo: Strong Privacy for Analytics in the Crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 441–459. https://doi.org/10.1145/3132747.3132769

[20] https://github.com/brave-experiments/sta-rs

[9] Marina Blanton and Everaldo Aguiar. 2012. Private and oblivious set and multiset operations. In *ASIACCS 12*, Heung Youl Youm and Yoojae Won (Eds.). ACM Press, 40–41.
[10] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2021. Lightweight Techniques for Private Heavy Hitters. IEEE Security & Privacy. https://eprint.iacr.org/2021/017.
[11] Daniel Bourdrez, Hugo Krawczyk, Kevin Lewi, and Christopher A. Wood. 2021. *The OPAQUE Asymmetric PAKE Protocol*. Internet-Draft draft-irtf-cfrg-opaque-07. IETF Secretariat. https://www.ietf.org/archive/id/draft-irtf-cfrg-opaque-07.txt https://www.ietf.org/archive/id/draft-irtf-cfrg-opaque-07.txt.
[12] Mark Bun, Jelani Nelson, and Uri Stemmer. 2019. Heavy Hitters and the Structure of Local Privacy. *ACM Trans. Algorithms* 15, 4, Article 51 (Oct. 2019), 40 pages. https://doi.org/10.1145/3344722
[13] David L. Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* 24, 2 (Feb. 1981), 84–90. https://doi.org/10.1145/358549.358563
[14] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation* (Boston, MA, USA) *(NSDI'17)*. USENIX Association, USA, 259–282.
[15] Alex Davidson, Armando Faz-Hernandez, Nick Sullivan, and Christopher A. Wood. 2021. *Oblivious Pseudorandom Functions (OPRFs) using Prime-Order Groups*. Internet-Draft draft-irtf-cfrg-voprf-06. IETF Secretariat. https://www.ietf.org/archive/id/draft-irtf-cfrg-voprf-06.txt https://www.ietf.org/archive/id/draft-irtf-cfrg-voprf-06.txt.
[16] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. 2018. Privacy Pass: Bypassing Internet Challenges Anonymously. *PoPETs* 2018, 3 (July 2018), 164–180. https://doi.org/10.1515/popets-2018-0026
[17] Jack Doerner and abhi shelat. 2017. Scaling ORAM for Secure Computation. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 523–535. https://doi.org/10.1145/3133956.3133967
[18] John R. Douceur. 2002. The Sybil Attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. Springer-Verlag, London, UK, 251–260. http://portal.acm.org/citation.cfm?id=687813
[19] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *ACM CCS 2014*, Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM Press, 1054–1067. https://doi.org/10.1145/2660267.2660348
[20] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. 2005. Keyword Search and Oblivious Pseudorandom Functions. In *TCC 2005 (LNCS, Vol. 3378)*, Joe Kilian (Ed.). Springer, Heidelberg, 303–324. https://doi.org/10.1007/978-3-540-30576-7_17
[21] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. 2015. Black-Box Garbled RAM. In *56th FOCS*, Venkatesan Guruswami (Ed.). IEEE Computer Society Press, 210–229. https://doi.org/10.1109/FOCS.2015.22
[22] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. 2012. Secure two-party computation in sublinear (amortized) time. In *ACM CCS 2012*, Ting Yu, George Danezis, and Virgil D. Gligor (Eds.). ACM Press, 513–524. https://doi.org/10.1145/2382196.2382251
[23] Sharon Huang, Subodh Iyengar, Sundar Jeyaraman, Shiv Kushwah, Chen-Kuei, Lee Zutian Luo, Payman Mohassel, Ananth Raghunathan, Shaahid Shaikh, Yen-Chieh Sung, and Albert Zhang. 2021. DIT: De-Identified Authenticated Telemetry at Scale. (2021). https://tinyurl.com/yxt7u2ss.
[24] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. 2014. Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model. In *ASIACRYPT 2014, Part II (LNCS, Vol. 8874)*, Palash Sarkar and Tetsu Iwata (Eds.). Springer, Heidelberg, 233–253. https://doi.org/10.1007/978-3-662-45608-8_13
[25] B. Kaliski. 2000. *PKCS #5: Password-Based Cryptography Specification Version 2.0*. RFC 2898. RFC Editor. http://www.rfc-editor.org/rfc/rfc2898.txt http://www.rfc-editor.org/rfc/rfc2898.txt.
[26] Marcel Keller and Avishay Yanai. 2018. Efficient Maliciously Secure Multiparty Computation for RAM. In *EUROCRYPT 2018, Part III (LNCS, Vol. 10822)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, Heidelberg, 91–124. https://doi.org/10.1007/978-3-319-78372-7_4
[27] Lea Kissner and Dawn Xiaodong Song. 2005. Privacy-Preserving Set Operations. In *CRYPTO 2005 (LNCS, Vol. 3621)*, Victor Shoup (Ed.). Springer, Heidelberg, 241–257. https://doi.org/10.1007/11535218_15
[28] Jon Kleinberg and Steve Lawrence. 2001. The Structure of the Web. *Science* 294, 5548 (2001), 1849–1850. https://doi.org/10.1126/science.1067014 arXiv:https://science.sciencemag.org/content/294/5548/1849.full.pdf
[29] Ben Kreuter, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. 2020. Anonymous Tokens with Private Metadata Bit. In *CRYPTO 2020, Part I (LNCS, Vol. 12170)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Heidelberg, 308–336. https://doi.org/10.1007/978-3-030-56784-2_11
[30] Steve Lu and Rafail Ostrovsky. 2013. Distributed Oblivious RAM for Secure Two-Party Computation. In *TCC 2013 (LNCS, Vol. 7785)*, Amit Sahai (Ed.). Springer, Heidelberg, 377–396. https://doi.org/10.1007/978-3-642-36594-2_22
[31] C. Andrew Neff. 2001. A Verifiable Secret Shuffle and Its Application to e-Voting. In *ACM CCS 2001*, Michael K. Reiter and Pierangela Samarati (Eds.). ACM Press, 116–125. https://doi.org/10.1145/501983.502000
[32] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. 2020. A survey of published attacks on Intel SGX. *arXiv preprint arXiv:2006.13598* (2020).
[33] C. Percival and S. Josefsson. 2016. *The scrypt Password-Based Key Derivation Function*. RFC 7914. RFC Editor.
[34] Zhan Qin, Yin Yang, Ting Yu, Issa Khalil, Xiaokui Xiao, and Kui Ren. 2016. Heavy Hitter Estimation over Set-Valued Data with Local Differential Privacy. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 192–203. https://doi.org/10.1145/2976749.2978409
[35] Eric Rescorla. 2021. (2021). `bofreq-privacy-preserving-measurement-06` https://datatracker.ietf.org/doc/bofreq-privacy-preserving-measurement/.
[36] Sacha Servan-Schreiber, Kyle Hogan, and Srinivas Devadas. 2021. AdVeil: A Private Targeted-Advertising Ecosystem. Cryptology ePrint Archive, Report 2021/1032. https://ia.cr/2021/1032.
[37] Latanya Sweeney. 2002. k-Anonymity: A Model for Protecting Privacy. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* 10, 5 (2002), 557–570. https://doi.org/10.1142/S0218488502001648
[38] Martin Thomson and Christopher A. Wood. 2021. *Oblivious HTTP*. Internet-Draft draft-thomson-ohai-ohttp-00. IETF Secretariat. https://www.ietf.org/archive/id/draft-thomson-ohai-ohttp-00.txt https://www.ietf.org/archive/id/draft-thomson-ohai-ohttp-00.txt.
[39] Nirvan Tyagi, Sofia Celi, Thomas Ristenpart, Nick Sullivan, Stefano Tessaro, and Christopher A. Wood. 2021. A Fast and Simple Partially Oblivious PRF, with Applications. Cryptology ePrint Archive, Report 2021/864. https://eprint.iacr.org/2021/864.
[40] Wennan Zhu, Peter Kairouz, Brendan McMahan, Haicheng Sun, and Wei Li. 2020. Federated Heavy Hitters Discovery with Differential Privacy. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 108)*, Silvia Chiappa and Roberto Calandra (Eds.). PMLR, 3837–3847. http://proceedings.mlr.press/v108/zhu20a.html

A CRYPTOGRAPHIC GUARANTEES

In the following section, we will assume the presence of each of the cryptographic primitives specified in Section 4. The description and security guarantees that we assume follow from Section 3.

A.1 Correctness

We first state the correctness guarantee of the STAR protocol.

Theorem 2. (Correctness) *The protocol $\mathcal{P}$ (similarly $\widetilde{\mathcal{P}}$) is correct with all but negligible probability.*

Proof. The correctness of STAR follows from the fact that $\mathbb{S}$ recovers a symmetric key $K_{\mathcal{E}_\iota}$ for every subset $\mathcal{E}_\iota \in \mathcal{Y}$ of compatible client shares of size greater than κ. In these instances, the server uses the value $t_{\mathcal{E}_\iota}$ to check which shares correspond to each other. It then uses the recover procedure to reveal $r_{\mathcal{E}_\iota,1}$ and derive $K_{\mathcal{E}_\iota}$. Once it knows $K_{\mathcal{E}_\iota}$, $\mathbb{S}$ is able to recover (x_j, aux_j) by decrypting each client message corresponding to $\mathcal{E}_\iota$.

As mentioned in Section 3, this requires that the underlying field $\mathbb{F}_p$ that secret shares are generated within is created with prime order p large enough. This ensures that randomly sampling shares from this field is unlikely to lead to collisions. If a collision occurs and the total number of *different* shares is $\leq \kappa$, then the recovery operation will not succeed. □

A.2 Security

We prove the security of STAR against a malicious adversary, that is allowed to operate in one of the following manners: corrupting the aggregation server and a set of clients together; corrupting the randomness server and a set of clients together; and corrupting only a set of clients. We show that the STAR protocol maintains client privacy (up to leakage specified by L) in the case where either server is corrupted. Furthermore, the computation is shown to be robust against an adversary that controls only a set of clients, and attempts to alter the protocol output. The security proofs for $\mathcal{P}$ are given in Theorems 3, 4, and 5. Throughout, we will use $\mathcal{F}_{\mathcal{P}}$ to refer to the ideal functionality for the STAR protocol, and we will use $\mathcal{F}_\Gamma$ to refer to the ideal functionality for the VOPRF protocol that is used [2].

Random oracle model usage in VOPRF. While the $\mathcal{P}$ protocol itself does not include any explicit usage of random oracles, we require that the internal VOPRF protocol uses a random oracle RO in the final evaluation of the PRF value. This allows the simulation to learn adversarial inputs from queries during the protocol execution. Specifically, we require that the VOPRF scheme produces outputs of the form $\mathsf{RO}(x, f(\mathsf{msk}, x))$. Many well-known OPRF primitives adhere to this requirement [16, 24, 29, 39].

Security proofs. We now detail the various theorems that prove the security of $\mathcal{P}$.

Theorem 3. (Malicious aggregation server) *The protocol $\mathcal{P}$ is secure against any $\mathcal{A}$ that corrupts $\mathbb{S}$ and some subset $C_{\mathcal{A}} \subset C$ of all clients, assuming a secure VOPRF protocol Γ, the* IND-CPA *security of Σ, and the privacy of $\Pi_{\kappa,n}$.*

Proof. Let the current epoch be denoted by τ. We construct our PPT simulator as follows.

- $\mathcal{S}$ runs $\mathsf{pp} \leftarrow \Gamma.\mathsf{setup}(1^\lambda)$ and $(\mathsf{msk}'_\tau, \mathsf{mpk}'_\tau) \leftarrow \Gamma.\mathsf{keygen}(\mathsf{pp})$ and sends pp to $\mathcal{A}$.
- $\mathcal{S}$ handles queries made by $\mathcal{A}$ to Γ by interacting with the ideal functionality $\mathcal{F}_\Gamma$.
- When $\mathcal{S}$ receives queries (x, y) to the random oracle $\Gamma.\mathsf{RO}$, it first checks that $y = f(\mathsf{msk}'_\tau, x)$. If this equality holds, it either returns $\mathsf{RO}[x]$, or samples $z \leftarrow\!\!\$\, \{0,1\}^\ell$, sets $\mathsf{RO}[x] = z_x$, and then returns z_x. If the inequality does not hold, it returns a randomly sampled value.
- When $\mathcal{S}$ receives $(\kappa, (c_i, s_i, t_i)_{i \in C_{\mathcal{A}}})$ from the adversary, it sends all inputs $\mathcal{X}_{\mathcal{A}}$ that it received RO queries for, with the set $\mathsf{aux}_{\mathcal{A}} = \{\perp\}_{l \in |\mathcal{X}_{\mathcal{A}}|}$ and κ, to $\mathcal{F}_{\mathcal{P}}$. It receives $\mathcal{Y}$ as output from $\mathcal{F}_{\mathcal{P}}$, and $\mathsf{L}(\mathcal{X})$.
- Let $\mathcal{N}$ be the collection of subsets of all indices returned by $\mathsf{L}(\mathcal{X})$, let $\mathcal{N}_{\mathcal{A}} \subset \mathcal{N}$ denote all subsets that contain inputs taken from $\mathcal{X}_{\mathcal{A}}$, and let $\mathcal{Z} = \emptyset$.
- For each $(x_j, \mathsf{aux}_j) \in \mathcal{Y}$:
  - If $z_x = \mathsf{RO}[x]$ is not empty, then let:
$$\begin{aligned} K_x &\leftarrow \mathsf{derive}(z_x[1]); \\ c_{x,j} &\leftarrow \Sigma.\mathsf{Enc}(K_x, x \| \mathsf{aux}_j); \\ r_j &\leftarrow\!\!\$\, \mathbb{F}_p; \\ s_{x,j} &\leftarrow \Pi_{\kappa,n}.\mathsf{share}(z_x[1], r_j; z_x[2]); \\ t_{x,j} &\leftarrow z_x[3]. \end{aligned} \tag{2}$$
    Else, sample $z_x \leftarrow\!\!\$\, \{0,1\}^\ell$ and construct $(c_{x,j}, s_{x,j}, t_{x,j})$ as in Equation (3).
  - Let $\mathcal{Z}[j] = (c_{x,j}, s_{x,j}, t_{x,j})$.
- For each j where $\mathsf{aux}_j = \perp$, delete $\mathcal{Z}[j]$.
- For each subset $\mathcal{N} \in \mathsf{L}(\mathcal{X})$ where $|\mathcal{N}| \leq \kappa$:
  - If $\mathcal{N} \in \mathcal{N}_{\mathcal{A}}$: for each $\hat{\iota} \in \mathcal{N}$: let $z_{\hat{\iota}} = \mathsf{RO}[x_{\hat{\iota}}]$, and construct $(c_\iota, s_\iota, t_\iota)$ as in Equation (3).
  - Else, sample $K_{\mathcal{N}} \leftarrow\!\!\$\, \{0,1\}^\lambda$, and then for each $\iota \in \mathcal{N}$ compute:
$$\begin{aligned} c_\iota &\leftarrow \Sigma.\mathsf{Enc}(K_\iota, 0); \\ s_\iota &\leftarrow\!\!\$\, \mathbb{F}_p; \\ t_\iota &\leftarrow\!\!\$\, \{0,1\}^{\ell/3}. \end{aligned} \tag{3}$$

In the following claims, we prove that the simulation is indistinguishable to the adversary from the real protocol via a series of game-hops. For a broad overview of the security proof, see Figure 12.

Claim A.1. $\mathcal{G}_0 \overset{s}{\approx} \mathcal{G}_1$ *due to the random oracle properties of* $\Gamma.\mathsf{RO}$.

Proof. In $\mathcal{G}_0$, the execution is as in protocol $\mathcal{P}$. In $\mathcal{G}_1$, all queries (x, w) for the $\Gamma.\mathsf{RO}$ are handled by first checking that $w = f(\mathsf{msk}_\tau, x)$, which can be done using the master secret key sampled by the simulator. If the check passes, then the query is answered by either returning $\Gamma.\mathsf{RO}[x]$ (if non-empty), or sampling a new value and assigning that to $\Gamma.\mathsf{RO}[x]$, before returning it. If the check does not pass, then the query is answered by simply returning a random value. Note that the pseudorandomness property of Γ ensures that the two games are indistinguishable. □

Claim A.2. $\mathcal{G}_1 \overset{c}{\approx} \mathcal{G}_2$ *by the security of* Γ.

Proof. In $\mathcal{G}_2$ the simulator no longer has access to msk_τ, and only has access to the ideal functionality $\mathcal{F}_\Gamma$. Any blind evaluation query for x' is answered by sending the query to the corresponding interface of $\mathcal{F}_\Gamma$, and returning the response

Step	Γ.RO queries	$i \in \widehat{C_{\mathcal{A}}}$	$(j \notin \widehat{C_{\mathcal{A}}}) \wedge (x_j \in \mathcal{Y})$	$(l \notin \widehat{C_{\mathcal{A}}}) \wedge (x_l \notin \mathcal{Y})$	Hop
$\mathcal{G}_0$	$(x_j, f(\mathsf{msk}_\tau, x))$	(c_i, s_i, t_i)	$(\Sigma.\mathsf{Enc}(K_j, x_j \Vert \mathsf{aux}_j), \Pi_{\kappa,n}.\mathsf{share}(r_{j,1}; r_{j,2}), r_{j,3})$	$(\Sigma.\mathsf{Enc}(K_l, x_l \Vert \mathsf{aux}_l), \Pi_{\kappa,n}.\mathsf{share}(r_{l,1}; r_{l,2}), r_{l,3})$	—
$\mathcal{G}_1$	$(x_j, w \stackrel{?}{=} f(\mathsf{msk}_\tau, x))$	(c_i, s_i, t_i)	$(\Sigma.\mathsf{Enc}(K_j, x_j \Vert \mathsf{aux}_j), \Pi_{\kappa,n}.\mathsf{share}(r_{j,1}; r_{j,2}), r_{j,3})$	$(\Sigma.\mathsf{Enc}(K_l, x_l \Vert \mathsf{aux}_l), \Pi_{\kappa,n}.\mathsf{share}(\widetilde{r_{l,1}}; \widetilde{r_{l,2}}), \widetilde{r_{l,3}})$	ROM
$\mathcal{G}_2$	$(x_j, w \stackrel{?}{=} \mathcal{F}_\Gamma(x_j))$	(c_i, s_i, t_i)	$(\Sigma.\mathsf{Enc}(K_j, x_j \Vert \mathsf{aux}_j), \Pi_{\kappa,n}.\mathsf{share}(r_{j,1}; r_{j,2}), r_{j,3})$	$(\Sigma.\mathsf{Enc}(K_l, x_l \Vert \mathsf{aux}_l), \Pi_{\kappa,n}.\mathsf{share}(\widetilde{r_{l,1}}; \widetilde{r_{l,2}}), \widetilde{r_{l,3}})$	VOPRF
$\mathcal{G}_3$	$(x_j, \mathcal{F}_\Gamma(x_j))$	(c_i, s_i, t_i)	$(\Sigma.\mathsf{Enc}(K_j, x_j \Vert \mathsf{aux}_j), \Pi_{\kappa,n}.\mathsf{share}(r_{j,1}; r_{j,2}), r_{j,3})$	$(\Sigma.\mathsf{Enc}(K_l, x_l \Vert \mathsf{aux}_l), s_l \leftarrow\!\$\, \mathbb{F}_p, \widetilde{r_{l,3}})$	(κ, n)-secret-sharing
$\mathcal{G}_4$	$(x_j, \mathcal{F}_\Gamma(x_j))$	(c_i, s_i, t_i)	$(\Sigma.\mathsf{Enc}(K_j, x_j \Vert \mathsf{aux}_j), \Pi_{\kappa,n}.\mathsf{share}(r_{j,1}; r_{j,2}), r_{j,3})$	$(\Sigma.\mathsf{Enc}(K_l, 0 \ldots 0), s_l \leftarrow\!\$\, \mathbb{F}_p, \widetilde{r_{l,3}})$	IND-CPA

Figure 12: Game hops required to prove security of Theorem 3. $\mathcal{G}_0$ corresponds to the real world execution of $\mathcal{P}$, and $\mathcal{G}_4$ corresponds to the PPT simulator that we use in the ideal world formulation. The third, fourth, and fifth columns correspond to the way that the triples, that are sent to $\mathbb{S}$ by the clients, are constructed in each game hop. The differences between each game are highlighted in blue.

to $\mathcal{A}$. When $\mathcal{A}$ makes a query (x, w) to RO, $\mathcal{S}$ sends (x) to the evaluation interface for $\mathcal{F}_\Gamma$ to check if the queries are admissible in the same way as $\mathcal{G}_1$. Note that the difference between $\mathcal{G}_1$ and $\mathcal{G}_2$ can be simulated by an adversary $\mathcal{B}$ against Γ. □

Claim A.3. *$\mathcal{G}_2 \stackrel{c}{\approx} \mathcal{G}_3$ by the share privacy of $\Pi_{\kappa,n}$.*

Proof. In $\mathcal{G}_3$, the simulator replaces all values (s_l, t_l) sent by honest clients $\mathbb{C}_l$ where $x_l \notin \mathcal{Y}$ and they belong to some $\mathcal{N} \in \widehat{\mathcal{N}} \setminus \mathcal{N}_{\mathcal{A}}$ (i.e., never queried to RO by the adversary), with random values. The distinguishing advantage of the two games can be bounded by an adversary trying to break the privacy requirements of $\Pi_{\kappa,n}$, since there are less than κ such shares. Moreover, the distribution of t_l is already random due to never having learnt the value from the output of Γ. □

Claim A.4. *$\mathcal{G}_3 \stackrel{c}{\approx} \mathcal{G}_4$ by the* IND-CPA *security of* Σ.

Proof. In $\mathcal{G}_4$, the only difference is that any message from an honest client $\mathbb{C}_l$ to the server $\mathbb{S}$ that includes ciphertexts that encode messages x_l that have not previously been queried to Γ.RO are modified. In particular, these messages replace the encrypted ciphertext of the encoded message $(x_l \Vert \mathsf{aux}_l)$ with an encryption of all zeros (matching the length). The difference between these two games can be simulated by an adversary $\mathcal{B}$ attempting to break the IND-CPA security of Σ, since the encryption key is derived from randomness that $\mathcal{A}$ never witnessed. Note that the clients $\mathbb{C}_l$ that belong to this set can be learned from the output $\mathcal{Y}$ and the output of the leakage function $\mathsf{L}(\mathcal{X})$. □

Note that the execution in $\mathcal{G}_4$ is identical to the view described by the simulator above. Therefore, putting Claim A.1, Claim A.2, Claim A.3, and Claim A.4 together, we have that the distinguishing advantage of the real-world execution and the ideal world simulation is negligible and the proof of Theorem 3 is complete. □

Theorem 4. (Malicious randomness server) *The protocol $\mathcal{P}$ is secure against any $\mathcal{A}$ that corrupts $\mathbb{O}$ and some subset $C_{\mathcal{A}} \subset C$ of all clients, assuming the security of* Γ*, and the* IND-CPA *security of* Σ.

Proof. By the security of Γ, the simulator can simulate the view of $\mathbb{O}$ during the randomness phase of the protocol. During the aggregation phase, the server $\mathbb{O}$ also witnesses encrypted client messages that are destined for the aggregation server. Such encrypted messages can be simulated as encryptions of all zeroes in every case by the IND-CPA security of Σ. This simulates the entire view of $\mathbb{O}$.

Note that interactions with the ideal functionality can be made without submitting any adversarial client inputs, since these may be arbitrarily corrupted. Note that the simulator can thus only maintain correctness for $\mathbb{S}$ up to the output learnt purely from honest clients. □

Theorem 5. (Malicious clients) *The protocol $\mathcal{P}$ is secure against any adversary $\mathcal{A}$ corrupting some subset $C_{\mathcal{A}} \subset C$ of all clients, assuming the security of* Γ*, the* IND-CPA *security of* Σ*, and the privacy of $\Pi_{\kappa,n}$).*

Proof. This proof follows an almost identical set of transitions to the proof of Theorem 3. Note that the simulator can simulate all messages in the same way, except that it only sends adversarial client messages to the ideal functionality that are *well-constructed*. It can check whether messages are well-constructed by checking that the ciphertext c_i encrypts a value x_i that was received in the queries to Γ.RO, and using a correctly derived key. Note that this can be checked using the combination of inputs and outputs derived from Γ.RO. Similarly, the simulator can check whether s_i and t_i are consistent with the value of x_i that is encrypted. The simulator is then able to construct a set of messages using the output of $\mathcal{F}_{\mathcal{P}}$ that provides the same correctness guarantees as in the real-world execution. The simulation for these messages is identical to the simulation in the proof of Theorem 3. □

A.3 Security for STARLite

The security of the STARLite protocol only holds when the client input distribution has sufficient min-entropy. Simulating security of unrevealed measurements against a malicious aggregation server is fairly trivial since the simulator can simply construct dummy-encodings, and rely upon the fact that the aggregation server is unable to guess which measurement is encoded with anything other than negligible probability. Otherwise, the simulation follows a similar model as the proof of Theorem 3. In the case of malicious clients, the security of the system can be ensured by modelling the randomness derivation process as interacting with a random oracle model. This allows constructing a proof with identical dynamics to Theorem 5.

In both cases, it should be reacknowledged that finding practical input distributions that demonstrate such behavior is difficult. We provide further discussion on this problem in Section 7.1.