Zero-Knowledge Cloud Analytics

Zeying Zhu, Clarence Lam, Alexander Frolov, Ian Miers, Zaoxing Liu
University of Maryland
College Park, Maryland, USA

Abstract

We present **zk-Analytics**, a distributed cloud analytics system that enables publicly verifiable analytics without revealing raw logs or relying on trusted hardware in analytics providers' infrastructure. Today's cloud analytics are largely self-assertive: providers collect telemetry, perform aggregation, and report results, leaving external parties unable to verify correctness without access to sensitive data or trusted execution environments. zk-Analytics addresses this gap by augmenting analytics pipelines with lightweight append-only log commitments and verifiable aggregation and query execution using zero-knowledge proofs. The system cleanly separates online log commitment from offline, distributed batch aggregation and query verification, enabling scalability while keeping online overhead low. We implement zk-Analytics using a zkVM-based execution environment and evaluate it on real-world and synthetic workloads, demonstrating that verifiable, privacy-preserving cloud analytics is feasible for real-world cloud workloads. zk-Analytics is open-sourced at https://github.com/Froot-NetSys/zk-Analytics.

CCS Concepts

• **Networks** → **Cloud computing**; • **Security and privacy** → **Cryptography**.

Keywords

Cloud analytics, Zero-knowledge proofs, Verifiable computation, Distributed analytics, Privacy-preserving systems

ACM Reference Format:

Zeying Zhu, Clarence Lam, Alexander Frolov, Ian Miers, Zaoxing Liu. 2026. Zero-Knowledge Cloud Analytics. In *ACM SIGCOMM 2026 Conference (SIGCOMM '26), August 17–21, 2026, Denver, CO, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3789240.3829157

1 Introduction

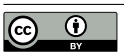
Modern cloud service sectors, ranging from video streaming and broadband delivery to security-metric reporting, increasingly rely on cloud-based analytics to demonstrate performance and regulatory compliance. These measurements underpin service-level agreements (SLAs), industry benchmarks, and public accountability [6, 10, 16, 33, 93, 102]. However, in today's landscape, these analytics remain fundamentally *self-asserted*: the same provider pulls the data from sources, aggregates them, and publishes performance reports. This creates a critical "trust gap" where external stakeholders, such as end users, competing streaming platforms, or regulatory auditors, lack the technical means to independently query and verify the correctness of reported metrics without accessing proprietary raw logs [28, 75].

A natural response is transparency through *disclosure*. However, raw logs often contain proprietary business and user information (e.g., IPs, cookies, request URLs) and proprietary operational details (e.g., routing or peering policies), making disclosure incompatible with privacy and confidentiality requirements [44, 80]. Worse, *disclosure does not mean correctness*: a provider can selectively summarize or falsify the aggregation of measurements [75]. A line of hardware-based *confidential computing* designs, such as Trusted Execution Environments (TEEs) [3, 11, 59], can offer runtime integrity, but TEEs run on the untrusted provider's own servers: attestation does not eliminate trust but instead shifts it to the hardware vendor and its attestation infrastructure (e.g., AWS Nitro [5]), rather than yielding an independently verifiable proof of correctness. TEEs also offer limited support for long-term auditability, as attestation verification depends on vendor-maintained trust infrastructure that evolves over time [72, 91]. Alternatively, measurements can be performed from the user end (e.g., probe [61] and speed tests [81]), which provide only indirect visibility and are vulnerable to preferential treatment or manipulation [18, 75, 81]. In summary, existing approaches that tackle the transparency problem therefore force an undesirable trade-off among verifiability, privacy, and deployability.

This paper asks: **how can an analytics provider make its analytics *publicly verifiable* and *privacy-preserving*, without revealing raw logs?** Our goal is accountability, not enforcement. We assume that data sources (e.g., routers or edge devices) correctly generate and commit logs according to declared measurement semantics. Thereafter, we do not trust the analytics provider. Instead, we seek guarantees that committed logs cannot be altered, omitted, reordered, or miscomputed, and that reported analytics results faithfully reflect the committed data.

We present **zk-Analytics**, a distributed cloud analytics framework that transforms performance claims from self-asserted reports into cryptographically verifiable statements. zk-Analytics augments existing analytics pipelines with two complementary mechanisms: (i) lightweight, tamper-evident cryptographic commitments over data collection, and (ii) zero-knowledge proofs (ZKPs) [26, 27, 50, 63, 98] that attest to the correct execution of aggregation and query logic over the committed data. Together, these mechanisms link reported analytics to an immutable set of authentic logs, while preserving data confidentiality. As a result, any verifier can independently validate reported query results, *offline* and *without trusting* the analytics provider, *without accessing* raw logs or proprietary infrastructure. zk-Analytics separates the analytics pipeline into three stages: *online log commitment*, *offline aggregation*, and *offline query verification*.

First, telemetry and analytics metrics are collected inside various *data sources* (e.g., video streaming servers, network routers, or devices) and recorded as append-only time-series logs. As logs are



SIGCOMM '26, Denver, CO, USA

ACM ISBN 979-8-4007-2467-1/26/08
https://doi.org/10.1145/3789240.3829157

produced, each data source incrementally commits to them using a lightweight cryptographic hash chain, binding every new batch to the entire prior history with constant per-log overhead. The committed log batches themselves are streamed to the analytics provider, which organizes them into fixed-size epochs. To make these commitments non-repudiable and independently checkable, each data source publishes the resulting commitment digests to a public, append-only bulletin board (e.g., a transparency log [39, 56, 64, 95]), so that any party, including external verifiers, can later confirm that a given log batch was committed at a particular point in time and was not retroactively altered.

Next, in the offline aggregation stage, zk-Analytics executes verifiable aggregation logic across a number of zero-knowledge proofs as a distributed system to transform raw logs into structured, query-ready summaries (e.g., per-key aggregates, histograms, or sketches for top-k, frequency, and distributed analytics [31, 66, 67, 107]), while simultaneously producing cryptographic commitments to each epoch's aggregated state. Each epoch is linked to the previous one via an append-only chain, ensuring completeness and preventing reordering or omission for verification. These epoch chains are maintained independently by multiple shared-nothing aggregators, allowing the aggregation stage to scale horizontally while keeping each chain independently verifiable.

Finally, when a verifier issues a query, the analytics provider generates a zero-knowledge proof showing that they executed the query logic over the relevant epochs of data. The verifier can independently validate this proof offline without accessing raw logs, intermediate state, or provider infrastructure, obtaining cryptographic assurance that the reported analytics faithfully reflect the underlying data.

We implement zk-Analytics using the RISC Zero zkVM [27] and evaluate it on real-world and synthetic datasets. Our evaluation shows that zk-Analytics provides strong verifiability while remaining feasible for moderate-scale cloud analytics workloads. For example, aggregating 131K logs with 8 aggregators, proof generation completes within 1.5 hours for histogram and sum-per-key aggregations, and within 4 hours for Count-Min Sketch aggregation. Both aggregation and query proof verification take less than 100 ms. Query execution on a single machine varies with queried data size; querying 131K aggregated logs completes in under 25 minutes for aggregation with sum-per-key, histogram-per-key, and Count-Min Sketch. These results demonstrate the feasibility of general verifiable, privacy-preserving analytics and provide a concrete step toward practical deployment. In summary, this paper makes the following contributions:

- We identify a fundamental lack of general, externally verifiable correctness guarantees in today's cloud analytics providers, and formulate verifiable analytics under an *untrusted analytics provider*.
- We design zk-Analytics, an end-to-end distributed framework for verifiable, privacy-preserving cloud analytics, combining append-only time-series log commitments and zero-knowledge proofs, with a scalable separation of online commitment from offline distributed aggregation and query verification.
- We implement and evaluate zk-Analytics using a zkVM-based proof system on real-world and synthetic workloads, showing that general verifiable analytics can be realized for moderate-scale cloud workloads.

2 Threat Model and Motivation

In this section, we discuss our threat model along with motivating use cases. We then explore existing solutions and their limitations.

2.1 Threat Model

Parties. We consider three types of parties: (1) **Data sources**, such as routers or edge devices, who generate raw logs within their infrastructure. (2) **Analytics providers**, such as video analytics platforms, who collect raw logs and commitments from data sources, aggregate logs into batches, and execute queries. (3) **Verifiers**, such as regulators, customers, or peer providers, who seek credible evidence for analytics results (e.g., performance or compliance claims) without having direct access to raw logs, analytics provider internals, or provider-controlled infrastructure.

Adversary model. The *analytics provider* is the primary adversary and may attempt to *misreport* analytics results by fabricating, omitting, or reordering generated log records, or by manipulating aggregation and query execution to produce incorrect results that falsely appear compliant with performance or resource usage claims. We assume *data sources* are trusted to correctly deploy and operate log collection and instrumentation once in place, relying on standard, widely deployed mechanisms such as NetFlow on network devices [12, 33] or systemd-based logging [100] in cloud operating systems. *Verifiers*, however, do not trust analytics results in the absence of cryptographic evidence. We further assume the security of standard cryptographic primitives, such as hash functions or zk-SNARK systems [22, 27, 63, 89], meaning that the adversary cannot violate the hiding/binding of commitments or forge valid cryptographic proofs.

Security goals. zk-Analytics is designed to detect *misreporting* by *analytics providers*, who we assume have access to raw logs. To ensure end-to-end verifiability of analytics results, the system must satisfy three properties: (1) *Input integrity*: raw logs cannot be manipulated or dropped after being generated by data sources. (2) *Confidentiality*: raw logs and intermediate computational states (e.g., full keys with private identifiers or individual values) must not be revealed to verifiers due to privacy concerns of sensitive data. The required level of privacy varies across use cases and is defined by the party generating the data. zk-Analytics should protect sensitive information processed by the analytics provider, such as user-identifying information (e.g., IP addresses, cookies, and request URLs), proprietary operational details (e.g., routing topologies and peering policies), and raw log data from being disclosed to verifiers. (3) *Computational integrity*: the query results produced by the system must be correctly computed over the correct set of raw original logs.

Security-non-goals. For zk-Analytics, our goal is to allow verification of existing queries in deployed systems without exposing the underlying logs to the verifier. As such, the provider remains a trusted viewer of the analytics data. Similarly, the data source is trusted to accurately produce and commit to accurate logs at recording time. Privacy in zk-Analytics refers to hiding raw logs

and intermediate analytics state from verifiers; it does not guarantee that query outputs themselves reveal no information. Similar to other zero-knowledge systems [65], information may still be inferred from the public outputs of approved queries.

2.2 Motivating Scenarios

With our threat model, we summarize three motivating scenarios, as illustrated in Figure 1.

Video streaming performance comparison. Video streaming services (e.g., Netflix, Hulu, and Disney+) commonly rely on third-party analytics providers to monitor and analyze user quality of experience metrics (QoE), such as bitrate, resolution, startup latency, and rebuffering [60]. By aggregating measurements across many providers worldwide, these providers (e.g., Conviva [36]) can offer industry-wide insights and enable cross-provider performance comparisons. Some examples of relevant queries for this application are *"Do a provider's QoE metrics (e.g., startup latency or rebuffering ratio) rank among the global top two?"*, or *"Does Provider A outperform Provider B during a specific time period?"*. However, to protect proprietary metrics and sensitive business information, analytics providers must not reveal the identities of participating providers or disclose individual customers' raw QoE data. At the same time, given the for-profit nature and growing influence of these platforms, it is important to independently verify the correctness and fairness of the insights they provide.

Verifiable broadband performance. Regulatory frameworks can require Internet Service Providers (ISPs) to measure and report network accessibility and performance metrics in order to demonstrate compliance with connectivity and quality-of-service requirements [68]. Prior work shows that ISPs can self-report Internet access levels comparable to those in urban regions while delivering substantially lower performance in practice [1, 68]. In this setting, end users control their traffic, whereas ISPs control the underlying network infrastructure; neither party is willing to disclose raw packet traces or internal network state to auditors or the public due to privacy and proprietary concerns. Meanwhile, auditors require trustworthy evidence of Internet connectivity and performance, such as achievable bandwidth, to ensure that reported analytics are not falsified and that ISPs comply with regulatory obligations. An example of a relevant query for this application is *"Does the achieved throughput in a given rural area meet the reported performance claims?"*.

Verifiable security metric reporting. Managed security providers (e.g., CrowdStrike [43], Arctic Wolf [15]) monitor their customers' networks and endpoints and report aggregate security metrics, such as the number of connections to known malicious destinations, or daily counts of critical detections [14, 40]. Such metrics are computed by applying a detection criterion, e.g., matching traffic against a public threat-intelligence blocklist, to timestamped telemetry such as network-flow and connection records generated within the customer's infrastructure (e.g., banks, hospitals). These reports serve as evidence in compliance audits [35, 84] and inform cyber-insurance underwriting [34, 88]. Here, the customer's network instrumentation (e.g., DNS and HTTP metrics) is the data source, the managed security provider is the analytics provider, and an auditor, regulator, or insurer is the verifier. The raw connection

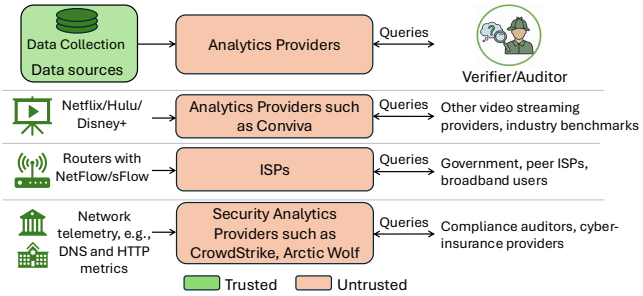


Figure 1: Overview of threat model with motivating use cases: data sources, an analytics provider, and an external verifier. Raw logs remain within the data sources and analytics provider, while only derived and verifiable query results are shared externally.

records cannot be disclosed, as they would expose hostnames, user identities, or internal network structure; yet the verifier requires trustworthy evidence that a reported metric is faithfully computed over the actual telemetry rather than fabricated. An example query is "*Over the reporting period, how many outbound connections from hosts in subnet S matched a specified threat intelligence indicator set?*".

Summary. Across video streaming analytics, broadband performance reporting, and security metric reporting, we observe a *common analytics structure* and *trust gap*. In each setting, raw data is generated by trusted data sources and transmitted to an analytics provider for aggregation and analysis. External verifiers lack the technical means to verify that these reports were computed correctly from complete and authentic data. These scenarios motivate the need for an end-to-end analytics framework that enables verifiable correctness of analytics results without revealing raw data. Such a system must ensure that reported analytics are faithfully derived from complete committed data and that queries are executed correctly, while keeping raw logs and intermediate analytics state hidden from verifiers.

2.3 Existing Solutions and Limitations

Despite extensive efforts in cloud analytics [41, 57, 96], *verifiable and privacy-preserving analytics* remain limited. We explore two existing approaches and their limitations

Confidential computing. Confidential computing approaches rely on trusted hardware (often TEEs [3, 11, 59]) to protect data aggregation and query execution [30, 32, 106]. In our setting, where data sources are trusted but the analytics provider is not, a TEE would run on the provider's own aggregation and query servers. This does not eliminate trust but instead shifts it to the hardware vendor's attestation root and the platform-managed attestation infrastructure (e.g., AWS Nitro [5]). Since the analytics provider operates on top of this infrastructure, verifying aggregation via attestation requires trusting the underlying platform and its attestation mechanisms rather than obtaining an independently verifiable proof of correctness. Confidential computing also offers limited support for long-term auditability: attestation verification relies on vendor collateral (certificates, TCB version and revocation data,

sometimes a live attestation service) that evolves over time, so previously valid attestations may stop verifying as hardware is patched or decommissioned [72, 91]. Finally, TEEs remain vulnerable to a range of microarchitectural and side-channel attacks that can leak enclave secrets or undermine their integrity guarantees [76, 101].

Verifiability at the cost of privacy. Verifiability can be achieved by disclosing *digested logs* and application code to auditors [75, 80, 105, 106]. While such disclosure enables verification, it comes at the cost of privacy and confidentiality. Cloud analytics data often includes detailed application and system metrics that can reveal sensitive business information, workload behavior, or internal system configurations, limiting adoption in commercial cloud environments [80]. Consequently, verifiability-through-disclosure conflates verifiability with data exposure and remains incompatible with privacy requirements.

3 Cryptographic Background

In this paper, we explore a novel architecture using zero-knowledge proofs to achieve verifiable, privacy-preserving cloud analytics. Zero-knowledge proofs [54] are cryptographic protocols that allow a prover to convince a verifier of the correctness of a computation without revealing the underlying data. In this paper, we work with zk-SNARKs (zero-knowledge Succinct Non-interactive ARgument of Knowledge) [23, 52, 74], which are a variant of zero-knowledge proofs with the additional properties that the proofs are succinct and do not require interaction to verify. As is common in the applied cryptography literature [29, 58, 90], we use these terms interchangeably, though this is somewhat colloquial. A zk-SNARK allows a prover to convince a verifier that, given a function $F(x, w)$ (represented by an arithmetic circuit), with input x public and input w private to the verifier, that the verifier knows w such that $F(x, w) = y$ for a public output y. We sketch this in Figure 2. The function F is parameterized by a set of *public inputs* x, which are known to the verifier and may include public data, commitments, or metadata required for verification, and a *private input* w (the witness), which contains secret inputs or state known only to the prover. The prover generates a proof, typically denoted π, attesting that there exists a witness w such that the circuit evaluates to y, without revealing w itself.

Below, we describe how various properties of zk-SNARKs are useful for zk-Analytics' security/efficiency properties. We defer formal definitions of zk-SNARKs and their properties to Appendix A. zk-SNARKs give zk-Analytics three essential properties: public verifiability, succinct verification, and privacy-preserving verification.

Public verifiability and auditability. zk-SNARKs are *non-interactive* and, by definition, *publicly verifiable*. This means that anybody can verify a zk-SNARK proof without interacting with the prover. In the context of cloud telemetry, verification does not require access to the prover's execution environment and can be performed offline, enabling long-term auditability of computation results. These computation results can be archived and re-verified at any point in the future without relying on the availability of the original execution environment.

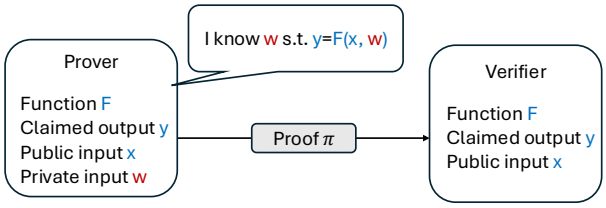


Figure 2: Outline of a zk-SNARK.

Privacy-preserving verification. Beyond being publicly verifiable, zk-SNARKs are also *zero-knowledge*. The verifier of a zk-SNARK can verify a proof while learning no information about the private witness w for F. Raw cloud metrics and logs often contain sensitive information, such as IP addresses, traffic patterns, or infrastructure identifiers. In the context of zero-knowledge proofs about cloud telemetry, these sensitive values would be contained in the private witness, and verifiers would learn nothing about these values from verifying proofs about computations on these values. When combined with cryptographic commitments (described below) in our system, zk-SNARKs can be used to show that a given set of logs was processed while keeping the values of the logs themselves private.

Succinct verification. zk-SNARKs are *succinct*. This means that proofs π are short (sublinear) in the size of the computations they prove, and take time sublinear in the size of the computation they prove to verify. In the context of our system, this means that the validity of a query on a database can be verified substantially faster than computing it. Concrete verification times for zk-SNARKs are generally hundreds of milliseconds or less. This allows for verifiers without large computational resources to verify queries over large data sets.

zkVMs. There has been a great deal of recent work making zk-SNARKs more efficient and scalable [21, 27, 63, 89]. One recent line of work is in building zero-knowledge Virtual Machines (zkVMs), such as RISC Zero [27] or Jolt [17]. These enable correctness proofs for arbitrary programs written in traditional high-level programming languages like Rust by encoding a CPU architecture in the arithmetic circuits for a zk-SNARK. These provide a better developer experience and simplify integration with existing systems. Our prior work [12], also based on zkVMs, has demonstrated the potential of zero-knowledge proofs for verifiable network telemetry. zkVMs have some practical advantages over more traditional circuit-based approaches to zero-knowledge proofs. They more naturally support computations with variable length inputs, and can distribute computations over multiple provers more naturally. Both of these properties more naturally lend zkVMs to runtime auto-scaling.

Commitments. We make use of cryptographic commitment schemes in this work. A commitment scheme is a primitive that allows a party to *commit* to a value, producing a *commitment*. Later, the user can *open* the commitment and demonstrate that they committed to some value. A secure commitment scheme is *hiding*, meaning that it reveals nothing about the underlying value, and *binding*, meaning

that users can only successfully open a commitment to the value they actually committed to. Commitments are often instantiated by cryptographically hashing the committed value concatenated with a random value [24]. We give a formal definition in Appendix A.

4 System Overview

4.1 Design Challenges and Key Insights

In designing zk-Analytics, we must balance three key requirements: verifiability, privacy, and scalability.

Challenge 1: Verifiable analytics without trusting the provider or sacrificing privacy. Enabling third-party verification of analytics results traditionally requires access to raw logs or trusted execution environments inside the analytics provider. Disclosing raw logs leaks sensitive and proprietary information, while a TEE-based approach does not remove trust but relocates it to the provider's own hardware and platform-managed attestation infrastructure. Consequently, existing approaches struggle to provide strong correctness guarantees without either trusting the analytics provider or compromising data privacy.

Key Idea. zk-Analytics leverages recent advances in zk-SNARKs to enable verifiable analytics over private logs. By combining cryptographic commitments with zero-knowledge proofs of correct aggregation and query execution, zk-Analytics allows verifiers to validate results via a query interface without accessing raw logs or trusting provider-controlled infrastructure.

Challenge 2: Scalable verifiable aggregation. With ZKPs, our cloud analytics pipeline consists of three stages: continuous online log collection from distributed data sources at high volume, followed by aggregation and query analytics that can be executed offline for auditing and reporting. While online log collection must remain lightweight to avoid disrupting performance-critical workflows, the *primary computational bottleneck* arises in the aggregation stage, as observed in our position paper [12], where large volumes of distributed and unordered logs must be organized into per-key time-series state or summarized into structured data representations (e.g., hash table of time series key-value pairs). Adding verifiability here can substantially increase overhead if it is not carefully designed.

Key Idea. zk-Analytics decouples lightweight online commitment from expensive offline aggregation and query computation, and targets scalability at the aggregation layer. Raw logs are treated as append-only time series and committed at data sources using lightweight hash chains [92], which naturally match sequential log streams and are cheaper than tree-based commitments such as Merkle trees [73]; this substitution is sound because membership proofs are not required at the data-source stage. Committed logs are then stored internally and processed by a *distributed batch aggregation* stage that parallelizes the costly task of organizing unordered log streams into per-key time-series stores or compact summaries (e.g., histograms and sketches). Keeping commitment lightweight and deferring aggregation offline bounds online overhead, while distributing aggregation across machines addresses the dominant computation cost in verifiable analytics pipelines.

Challenge 3: Supporting a diverse set of analytical queries with low overhead. Analytics providers must support a wide range of queries over raw data, including simple aggregations, per-key statistics, distributional metrics, and heavy-hitter detection. However, directly proving arbitrary queries over raw logs is computationally expensive and does not scale to high-volume workloads.

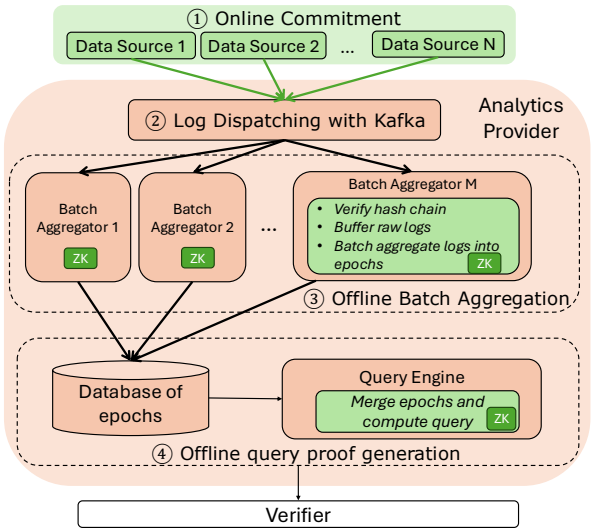


Figure 3: System overview of zk-Analytics.

Key Idea. zk-Analytics supports diverse analytical queries by organizing aggregated data into multiple summary representations tailored to different query classes. Specifically, zk-Analytics supports queries over raw logs for fine-grained analysis, histogram-based summaries for distribution and percentile queries, and Count-Min Sketches for frequency estimation and Top-K queries. By selecting appropriate summary structures during aggregation, zk-Analytics enables efficient and scalable verification of a broad range of analytics queries without sacrificing verifiability or performance.

4.2 zk-Analytics Workflow

zk-Analytics is an end-to-end distributed verifiable analytics framework that enables trustworthy analytics without exposing raw logs. As illustrated in Figure 3, the system consists of three logically separated stages: *online commitment*, *offline distributed batch aggregation*, and *offline query and verification*. Data sources are trusted to generate raw logs correctly, while the analytics provider, including log transmission, aggregation, and query execution, is not trusted and must provide cryptographic evidence of correctness.

① Online commitment. Log collection occurs inside data sources (e.g., video streaming services, ISPs) using system monitors or application-level instrumentation defined by providers. Each log entry is a time series key value pair, defined as $l = (ts,\ key = (k_1, \ldots, k_n),\ value)$, where the key encodes n dimension identifiers (e.g., client location, OS version, bitrate metric for video streaming use case) and the value is a numerical measurement. For example, $l = (ts,\ (\texttt{US-West},\ \texttt{bitrate}),\ 4800)$ records a 4800 kbps bitrate from a US-West client. The data source records raw log entries and treats them as an append-only time series, committing to each log batch at recording time using a lightweight hash chain [92] that

links it to the previous batch. Optionally, per-batch zero-knowledge proofs can be generated to attest to correct local preprocessing (e.g., filtering [83]). The outputs of this stage consist of raw log batches and their corresponding hash-chain commitments. Each data source also periodically publishes hash chain checkpoints to a public transparency log [39, 64].

② **Log dispatching.** Committed log batches are transmitted to the analytics provider via a reliable dispatcher, such as Apache Kafka [13]. The dispatcher partitions committed logs based on data source or key, ensuring that all log batches belonging to the same source are consistently routed to the same aggregator. This guarantees ordered delivery of commitments while enabling scalable, parallel processing across multiple aggregators.

③ **Offline distributed batch aggregation.** The batch aggregation stage runs on multiple analytics-platform–controlled servers and is decoupled from online commitment. Each aggregator first buffers incoming log batches and their associated hash-chain commitments in a local persistent database, such as a key-value store. Aggregators continuously accumulate recent log batches until a configured threshold is reached, at which point the buffered logs form a fixed-length epoch. Once an epoch is formed, the corresponding logs and commitments are fetched from the local database and provided as input to a zk-SNARK for aggregation. The zk-SNARK verifies commitment integrity and computes organized and summarized data structures required by supported queries, such as key-value hash table for time series, per-key aggregates, histograms, or sketches. By distributing aggregation across multiple machines, zk-Analytics parallelizes the most computationally expensive stage of the analytics pipeline while ensuring verifiable execution. Raw logs and intermediate aggregation state remain confined to analytics-platform–controlled infrastructure and are never exposed externally. Each aggregator likewise publishes epoch chain checkpoints to a public transparency log, committing to a single, non-equivocating epoch history.

④ **Offline query and verification.** Queries are executed by a query engine running on analytics-platform-controlled servers that are logically separated from data sources. Given an analytics query (e.g., compliance checks or usage summaries), the query engine operates over epoch-level aggregation results and executes the query logic inside a zk-SNARK to generate a cryptographic proof attesting to the correctness of the reported result with respect to the committed logs. Each query returns both the query result and a succinct proof, which verifiers can independently validate without accessing raw log entries, internal databases, or analytics-platform-controlled infrastructure. Validation anchors the epoch hash chain and per-source boundaries to the public transparency logs, extending the trust chain from the query result back to the logs each data source committed to.

5 Detailed Design

This section presents the design of zk-Analytics, focusing on how the system achieves verifiable online commitment, scalable offline distributed aggregation, and verifiable query execution, while preserving the privacy of raw logs and intermediate states.

Algorithm 1 Log Commitment (`data source`)

Input: source id s; streaming log entries $\{l_j\}_{j\geq 1}$, $l_j = (ts_j, key_j, value_j)$; initial chain hash h_0 (per source); log batch size B, publication interval P, transparency log TLog_s

Output: per-batch tuples $B_{seq} = (s, seq, \text{logs}, h_{seq})$ sent to dispatcher

```
logs ← [ ];  seq ← 0
for j ← 1, 2, 3, ... do
    logs.append(l_j)
    if |logs| ≥ B then
        seq ← seq + 1
        h_seq ← CommitHash(h_{seq−1}, logs)
        Send batch B_seq = (s, seq, logs, h_seq) to dispatcher
        logs ← [ ]
        if seq mod P = 0 then
            Publish checkpoint (seq, h_seq) to TLog_s
```

5.1 Online Verifiable Commitment

In zk-Analytics, data sources commit to logs as they are produced. The ultimate goal of these commitments is allowing verifiable queries over the logs. While the rest of these steps in the zk-Analytics pipeline are handled offline by other parties, commitments must be lightweight since data sources are often performance-critical and resource constrained.

Append-only log streams. To minimize overhead, zk-Analytics models logs as append-only time series of key-value pairs, where keys may come from a large range and be multi-dimensional. Each log record is represented as $(ts, key, value)$ [6], where ts denotes the timestamp, key identifies the log series (e.g., host identifier, flow ID, or metric name), and $value$ is the recorded measurement. Data sources emit raw logs and hash commitments as they are generated. Deferring aggregation to later stages ensures that online commitment remains lightweight and scalable.

Tamper-evident log commitments via hash chains. Because logs arrive as an ordered stream, zk-Analytics commits to log records using a hash-chain construction [92]. For each batch of logs, the data source updates a running commitment that binds the entire prefix of the log stream. Hash chains incur constant per-log computation and require only constant-size state, making them well suited for continuous, high-volume log streams. Once committed, log records cannot be reordered, removed, or modified without detection. Algorithm 1 illustrates the online commitment process executed at each data source. As log entries arrive, the source buffers consecutive log entries into fixed-size batches. When a batch reaches the configured size, the data source computes a new hash-chain value by hashing the previous chain value together with the buffered log entries, and emits the batch along with its commitment to a downstream dispatcher. By verifying the continuity of the hash chain, downstream aggregators can validate the integrity, ordering, and completeness of log entries produced by each data source. Periodically, each data source publishes hash-chain checkpoints to a public transparency log [39, 64] (e.g., Trillian [56]).

Algorithm 2 Epoch batch aggregation (`single aggregator`)

Input: batches $\{B_i = (s_i, seq_i, \text{logs}_i, h_i)\}_{i\geq 1}$ from dispatcher; epoch threshold N; prev. epoch tip h^E_{t-1}; per-source incoming epoch boundary $inTip[\cdot]$; epoch publication interval P_E

Output: $\mathcal{E}_t = (t, h^E_{t-1}, h^E_t, c^E_t, \{(s, inTip[s], outTip[s])\}_{s\in\mathcal{S}_t}, \text{state}_t, r_t, \pi_t)$

buf ← [ ]; lastFlush ← now()
for each incoming B_i **do**
  buf.append(B_i)
  if |buf| ≥ N **then**
    — Beginning of zk-SNARK Circuit —
    public input $x_t = (h^E_{t-1}, \{inTip[s]\}_{s\in\mathcal{S}_t})$; witness $w_t = (\text{buf}, r_t)$; public output $out_t = (h^E_t, c^E_t, \{outTip[s]\}_{s\in\mathcal{S}_t})$
    for all source $s \in \mathcal{S}_t$ with batches in buf **do**
      $acc \leftarrow inTip[s]$
      for all $(s_i, seq_i, \text{logs}_i, h_i)$ of s in buf, in seq order **do**
        $acc \leftarrow \text{CommitHash}(acc, \text{logs}_i)$
        assert $acc = h_i$
      $outTip[s] \leftarrow acc$
    $\text{state}_t \leftarrow \text{AggregateFunc}(\text{buf})$
    $r_t \xleftarrow{\$} \{0,1\}^\lambda$ ▷ commitment randomness
    $c^E_t \leftarrow \text{Commit}(\text{state}_t; r_t)$
    $h^E_t \leftarrow \text{CommitHash}(\text{TAG\_EPOCH} \,\|\, h^E_{t-1} \,\|\, c^E_t)$
    — End of zk-SNARK Circuit —
    $\pi_t \leftarrow \text{Prove}(x_t; w_t)$; EpochDB.put($\mathcal{E}_t$)
    if $t \bmod P_E = 0$ **then** publish epoch checkpoint (t, h^E_t) to transparency log TLog^E
    $inTip[s] \leftarrow outTip[s]$ $\forall s \in \mathcal{S}_t$
    buf ← [ ]; $h^E_{t-1} \leftarrow h^E_t$; $t \leftarrow t+1$

5.2 Distributed Verifiable Aggregation

zk-Analytics introduces *distributed batch aggregation* as an intermediate layer of computation between data sources and the verifiable query engine. Aggregators transform raw logs into committed epoch states that are later used by the query engine. Since aggregation is the dominant offline cost in the pipeline, zk-Analytics scales this layer horizontally across independent aggregators.

Dispatcher and log partitioning. Committed log batches emitted by data sources are transmitted to the analytics provider via a reliable dispatcher (e.g., Apache Kafka [13]). The dispatcher partitions logs by source ID or key ID and routes each partition to a specific aggregator in a *share-nothing* architecture, ensuring that each aggregator processes a disjoint subset of logs. Reliable delivery is critical: hash-chain verification at aggregators requires that every committed log batch be delivered *once and only once*. In practice, common dispatchers may provide at-least-once delivery semantics; therefore, each aggregator buffers received log batches in a local persistent store and performs deduplication before aggregation to ensure consistency with the data-source hash-chain commitments. If log delivery were incomplete or reordered, hash-chain verification would fail, preventing verifiers from validating downstream analytics results.

Distributed aggregation and epoch formation. Each aggregator operates independently and maintains no shared state with others. It buffers verified log batches in a local persistent database (e.g., a key-value store) and organizes them into fixed-size epochs once a configurable size threshold or timeout is reached, at which point it executes a verifiable aggregation procedure inside a zk-SNARK, as shown in Algorithm 2.[1] Within this procedure, the aggregator verifies data-source hash-chain continuity for each source over the buffered batches, aggregates the logs into an epoch state, derives a hiding commitment to that state, and links the epoch to the previous one through an append-only hash chain it maintains over epoch commitments. The resulting epoch is persisted to the provider's database, and the aggregator publishes, for each epoch, its public outputs: (i) the per-source incoming and outgoing hash-chain boundaries of the logs it has processed, (ii) the aggregator's epoch hash-chain values, and (iii) the hiding commitment to the epoch's aggregated state, together with a proof attesting that the epoch state is the correct aggregation of those logs and that the per-source boundaries are consistent with the source-committed batches. Periodically, each aggregator also publishes epoch hash-chain checkpoints (t, h^E_t) to a public epoch transparency log TLog^E [39, 64], enabling verifiers to confirm that the analytics provider commits to a single, non-equivocating epoch history. This design enables independent verification of per-source log continuity and ordering at each aggregator, with completeness anchored to the data sources' transparency logs, as well as per-aggregator epoch-chain integrity across distributed shared-nothing aggregators. Raw logs and intermediate aggregation state remain confined to the analytics provider's infrastructure and are never exposed externally; verifiers observe only the public commitments and hash-chain values needed for verification.

Aggregation state and queries. zk-Analytics exposes the aggregation data structure as a configurable choice, ranging from raw-log processing to per-series summaries and compact sketches, trading off the supported query statistics against performance. For statistical queries, it supports single statistics such as sums, averages, min, and max, exponential histograms [82] based on $\log_2$ bucketing for quantile and distribution queries, and Count-Min Sketches [31, 37] for frequency and Top-K estimation.

Handling elasticity. Our implemented system achieves horizontal scaling via sharding, where data sources are assigned to fixed shards. Skewed workloads can limit parallelism when a few sources dominate the input stream. zk-Analytics can be extended to support dynamic resharding, reassigning data-source ownership across aggregators at epoch boundaries. Since each aggregator persists a chain-tip commitment at every epoch's end, a newly assigned aggregator can resume from the latest committed tip. This preserves integrity: the new aggregator must continue from the exact committed tip, and the linkage is verified during proof generation, preventing divergence or omission of previously committed logs.

5.3 Offline Verifiable Query Engine

zk-Analytics exposes an offline verifiable query service to external verifiers through a standard HTTP interface. Verifiers submit analytics queries to the query engine, which returns both the query

[1] Note: When we use the notation $\xleftarrow{\$}$, we mean that the prover samples r_t out of circuit.

Algorithm 3 Verifiable query (`query engine`)

Input: query function q, query window W, EpochDB
Output: (out, π)

$(h_0^E, \{(h_k^E, c_k^E)\}_{k=1}^n, \{(\text{state}_k, r_k)\}_{k=1}^n) \leftarrow \text{LoadEpochs}(W)$ ▷ host: fetch and unpack epochs

— Beginning of zk-SNARK Circuit —

Let the epochs in $W = [start, end]$ be renumbered as $k = 1, \ldots, n$, where $h_1^E = h_{\text{start}}^E$, $h_n^E = h_{\text{end}}^E$, and h_0^E is the tip of the epoch preceding $start$.
public input $x = (h_0^E, \{(h_k^E, c_k^E)\}_{k=1}^n)$; witness $w = \{(\text{state}_k, r_k)\}_{k=1}^n$; public output $out = (h_{\text{start}}^E, h_{\text{end}}^E, ans)$

for $k = 1, \ldots, n$ **do**
  $h_k'^E \leftarrow \text{CommitHash}(\text{TAG\_EPOCH} \,\|\, h_{k-1}^E \,\|\, c_k^E)$
  assert $h_k'^E = h_k^E$ ▷ verify epoch hash and chain linkage
  $c' \leftarrow \text{Commit}(\text{state}_k; r_k)$
  assert $c' = c_k^E$ ▷ verify epoch commitment
$\text{merged} \leftarrow \text{Merge}(\{\text{state}_k\})$; $ans \leftarrow q(\text{merged})$
$out \leftarrow (h_{\text{start}}^E, h_{\text{end}}^E, ans)$

— End of zk-SNARK Circuit —

$\pi \leftarrow \text{Prove}(x; w)$
return (out, π)

Algorithm 4 Proof verification (`verifier`)

Input: window $W = [start, end]$; result (out, π) with anchor h_0^E; per-epoch public data $\{(h_k^E, c_k^E, \{(s, inTip_k[s], outTip_k[s])\}_s, \pi_k)\}$; logs TLog^E, $\{\text{TLog}_s\}$
Output: accept / reject

fetch latest $(t_c, h_{t_c}^E) \in \text{TLog}^E$ with $t_c \le start$; $g \leftarrow h_{t_c}^E$
for $j = t_c{+}1, \ldots, start{-}1$ **do**
  $g \leftarrow \text{CommitHash}(\text{TAG\_EPOCH} \,\|\, g \,\|\, c_j^E)$
  assert $g = h_j^E$
assert $g = h_0^E$ ▷ h_0^E descends from the checkpoint
for all window epoch k, $t_k \in [start, end]$ **do**
  assert $\text{VerifyEpoch}(\pi_k)$
for all source s in W **do**
  assert $outTip_k[s] = inTip_{k+1}[s]$ across s's window epochs
  assert s's first $inTip$ and last $outTip$ in $W \in \text{TLog}_s$ ▷ anchor to source checkpoints
assert $\text{Verify}(\pi,\ x{=}(h_0^E, \{(h_k^E, c_k^E)\}_{k=1}^n),\ out)$ ▷ window chain, commitments, query
return accept

result and a succinct cryptographic proof attesting to the correctness of the query computation, as shown in Algorithm 3. Upon receiving a query, the query engine retrieves the relevant per-epoch aggregation states, their commitment openings, and associated chain metadata for the specified query window from the analytics provider's internal persistent storage. The query is then executed entirely inside a zk-SNARK (in our system a zkVM guest program), which ensures that all verification and computation steps are performed deterministically and are fully auditable. The guest program first verifies aggregation integrity by checking the per-aggregator epoch hash chain. Specifically, it recomputes each epoch hash, validates hash-chain continuity, and verifies an opening for each epoch state commitment for the witnessed state. This process guarantees that the retrieved epoch states are authentic and consistent up to the query end time. After successful hash-chain verification, the guest program merges the verified per-epoch payloads into a single summary data structure according to the configured aggregation mode and query types. It then applies the specified query function over this merged state to compute the query result. Along with the result, the zk-SNARK program outputs the window's start and end epoch hash chain values, binding the answer to the verified aggregation state; the prover then produces the proof, which any verifier can validate to obtain assurance of both aggregation integrity and correct query execution.

5.4 Access Control and Allowed Queries

Not any query should be allowed in zk-Analytics due to data leakage considerations. While zk-Analytics provides cryptographic integrity and correctness guarantees via zero-knowledge proofs, it does not provide access control. A zero-knowledge proof alone does not prevent a query from intentionally or unintentionally encoding sensitive information into its public output. For example, a query such as "Does flow A exist in the ISP?" would directly reveal sensitive flow-level information. Such queries should be rejected or sanitized prior to execution.

Queries in zk-Analytics can be regulated using a combination of static and dynamic checks. Under a simple allowlist approach, the system publishes a set of pre-approved query functions whose privacy properties have been manually audited. While secure, this restricts the system to a fixed set of pre-vetted queries. To support dynamic queries, the system can enforce run-time checks, e.g., a blocklist mechanism that detects and rejects query functions exhibiting known leakage patterns, following prior work on privacy enforcement [19, 53, 86, 103, 104]. Regardless of the approach, the overhead of access control is largely orthogonal to the overhead of producing the zero-knowledge proof, since checking if the query is allowed is independent of proving its correctness.[2]

5.5 Verifier

The query proof π alone attests only that the answer was computed correctly over a set of epoch commitments forming a valid epoch hash chain; it does not establish that those epochs faithfully reflect the sources' logs, nor that the provider committed to a single epoch history. zk-Analytics's verification therefore adds checks over two transparency logs: the per-source logs $\{\text{TLog}_s\}$ for *data authenticity*, and the aggregator epoch log TLog^E for *non-equivocation* (Algorithm 4). First, the verifier anchors the epoch chain to a publicly agreed value: it fetches the latest checkpoint $(t_c, h_{t_c}^E)$ from TLog^E with $t_c \le start$, re-derives the chain to the window predecessor, and confirms the query's public input h_0^E descends from it; since TLog^E is append-only and public, the provider

[2] Looking ahead, our prototype implementation runs the entire query engine inside a zkVM to produce a proof of query correctness. Dynamic access checks, if naively added, would also run in the VM or as a separate re-execution of the query with checks. But this is avoidable with more engineering work.

cannot show divergent histories to different verifiers. Second, for each window epoch it checks the aggregation proof π_k (Section 5.2), binding the epoch's committed state and per-source boundaries ($inTip_k[s], outTip_k[s]$) to the source-claimed batch hashes. Third, for every source it checks boundary continuity across consecutive epochs ($outTip_k[s] = inTip_{k+1}[s]$) and matches the window's start/end boundaries against that source's checkpoints in TLog_s, anchoring the aggregated data to exactly the committed logs, no omission, insertion, or reordering. Fourth, it verifies the query proof π against its public input $(h_0^E, \{(h_k^E, c_k^E)\}_{k=1}^n)$ and claimed output; the in-circuit checks of Section 5.3 establish epoch contiguity, commitment correctness, and correct query evaluation. Every link is verified against public values, so no parties beyond the data sources and a transparency-log operator need be trusted. Appendix B states and sketches proofs of zk-Analytics's end-to-end integrity and privacy guarantees, with their assumptions and limits.

6 Implementation

zk-Analytics comprises three components: a data-commitment component in data sources, a set of distributed batch aggregators fed by an Apache Kafka [13] dispatcher, and a zero-knowledge query engine, together forming an end-to-end verifiable, privacy-preserving analytics pipeline. It is implemented in 13K lines of Rust using the RISC Zero zkVM [27], which instantiates zk-Analytics's zk-SNARKs by expressing the aggregation and query circuits as guest programs.

Log commitment. Each data source acts as a Kafka producer, submitting logs in batches grouped by key. We use SHA-256 for log commitment, as it runs efficiently in RISC Zero [87]. Each log entry is a 15-byte key, a 32-bit value, and a 32-bit timestamp, 23 bytes total; the key and value lengths are configurable per application. A 15-byte key suffices for most real-world datasets: an IPv4 flow identifier as a 5-tuple needs only 13 bytes. We partition committed logs across aggregators by source ID in Kafka, so each source's chain lands on a single aggregator.

Batch aggregator and query engine. Aggregator hosts run Kafka consumers that receive log batches from distributed sources, buffering raw log epochs and hash commitments in a local RocksDB instance [8, 45]. Aggregated output epochs are persisted in FoundationDB [4], the analytics provider's internal key-value store from which query engines fetch epoch data and proofs. Query engines support multiple query types over aggregated epochs; our prototype implements Count-Min sketch (3 rows × 1024 columns) estimates and Top-K queries, histogram bucket counts and percentiles, and per-key sum/average aggregations over hash tables.

7 Evaluation

We evaluate zk-Analytics on real-world and synthetic datasets. zk-Analytics supports a broad range of cloud analytics queries with end-to-end verifiable correctness at practical cost (§7.1); its distributed design targets the dominant bottleneck, batch aggregation, achieving near-linear speedup as aggregator machines increase; and offline query proof generation stays tractable at realistic scales (§7.2). Across all workloads, aggregating 131K log entries with 8 aggregators completes proof generation within 1.5 hours for histogram and hash table aggregations and within 4 hours for Count-Min Sketch, with both aggregation- and query-proof verification under 100 ms, and querying over 131K aggregated log entries completes in under 25 minutes across all aggregation data structures.

Experimental setup. Our prototype testbed is deployed on CloudLab [99] and consists of collectors, aggregators, and query engines. We use machines equipped with Intel Xeon Gold 5512U processors running at 2.1 GHz, with up to 56 cores, 128 GB of memory, and 800 GB NVMe SSDs, interconnected via 10 Gbps NICs. In our implementation, RISC Zero generates succinct receipts [7], whose proof size and verification time are independent of the length of the guest program execution. Publishing checkpoints involves posting one hash per interval, orders of magnitude cheaper than the per-epoch proving it accompanies, so we do not measure it separately; the transparency-log service itself is external infrastructure outside our scope. Our reported verification times separate the aggregation proof verification and query proof verification.

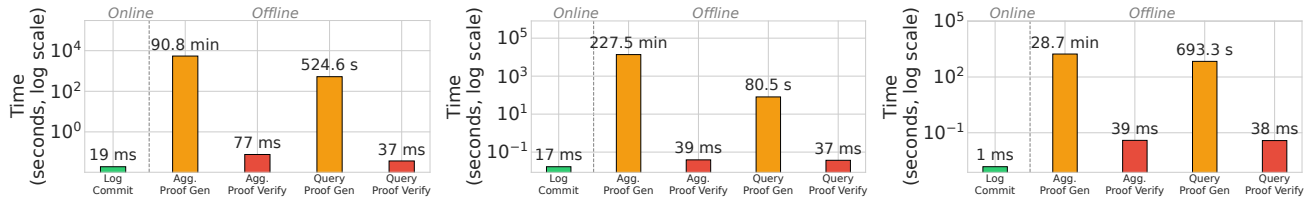
Datasets. We generate synthetic traces with keys (time series) drawn from either a Zipf distribution with skewness parameter $s = 1.2$, where $P(k) \propto k^{-1.2}$, or a uniform distribution. For both key distributions, we apply the same Zipf distribution to values within each time series, while varying the number of distinct keys to control workload scale. We also evaluate zk-Analytics on real-world datasets, including CAIDA flow traces [2] collected from a backbone link of a Tier-1 US ISP. We extract source IP and destination IP from each packet as a key, and packet length as value, as each log, as well as Google Cluster dataset v3 [55] (*Google*), where we use start_time as time, average_usage.memory as memory_usage. We also use a vehicle CO_2 emission dataset measuring the emission of commercial vehicles from 2015 to 2025 in Canada [79].

7.1 End-to-end Experiments

We evaluate zk-Analytics's end-to-end performance on three real-world datasets that capture diverse application domains and data distributions. Figure 4 breaks down online and offline costs across log commitment, aggregation- and query-proof generation, and verification; Table 1 reports dataset characteristics and proof sizes.

Google cluster dataset. The Google cluster traces collect cloud resource usage workloads with largely uniform key distributions as machine IDs and fixed time-series collection interval. Each machine reports memory usage metrics, resulting in evenly distributed per-key logs. In our evaluation, we use 8,192 unique keys and 131,072 logs. Figure 4(a) shows that online log commitment incurs negligible overhead (19 ms), while offline aggregation proof generation dominates end-to-end latency (90.8 min). When querying the sum of all memory-usage values to reveal aggregate memory demand, query proof generation takes 524.6 s, while both aggregation and query-proof verification incur sub-100 ms overhead.

Network traces. Network telemetry is characterized by highly skewed key distributions, where a small fraction of flows accounts for a disproportionate share of traffic [69]. We evaluate zk-Analytics using CAIDA backbone traces, treating source and destination IP addresses as keys and query the Top-10 total packet lengths per key. The workload contains up to 100,000 distinct keys and 131,072 log records. As shown in Figure 4(b), aggregation proof generation is more expensive than in the Google cluster workload, taking 227.5 min offline, due to the more complex Count-Min (CM) Sketch



(a) Google cluster memory usage of sum aggregation over all values based on hash table aggregation (8 aggregators)

(b) CAIDA of Top-10 query of (source IP, destination IP)'s total packet lengths based on Count-Min Sketch aggregation (8 aggregators)

(c) Vehicle emission of histogram distribution of CO_2 emission of a certain vehicle manufacturers based on histogram aggregation (4 aggregators)

Figure 4: End-to-end zk-Analytics performance on real-world datasets with 4 to 8 distributed aggregator nodes and 1 query node: zk-Analytics finishes proof generation for aggregation and query within reasonable time, with small verification time.

Datasets/Metrics	Google Cluster	CAIDA	Vehicle Emissions
Number of keys	8,192	100,000	9,727
Number of log entries	131,072	131,072	10,058
Key distributions	Uniform	Skewed	Uniform
Total Aggregator Proof	2 MB	1.6 MB	962 KB
Query proof	225 KB	222 KB	222 KB
Public output (Agg. + Query)	15.8 KB	16 KB	11.8 KB

Table 1: Real-world datasets used for end-to-end evaluation and proof sizes results: zk-Analytics introduces small verification overhead from proof sizes.

Metric	Google Cluster		CAIDA		Vehicle Emissions	
	Vanilla	ZK	Vanilla	ZK	Vanilla	ZK
RocksDB insert (per epoch)	444 ms		448 ms		68 ms	
RocksDB read (per epoch)	2.9 ms		1.8 ms		0.3 ms	
FDB write (per epoch)	4.3 ms	7.4 ms	3.6 ms	6.5 ms	3.9 ms	7.6 ms
FDB read (per query)	18 ms	18 ms	3 ms	3 ms	3 ms	3 ms
Total Aggregation time (in parallel)	3.6 ms	90.8 min	5.3 ms	227.5 min	1.5 ms	28.7 min
Query time (per query)	18 ms	524.6 s	3 ms	80.5 s	3 ms	693.3 s
Peak mem (per agg. node)	34 MB	9.48 GB	32 MB	9.51 GB	31 MB	9.43 GB
Peak mem (per query node)	33 MB	9.44 GB	29 MB	9.42 GB	29 MB	9.44 GB

Table 2: Vanilla baseline vs. zk-Analytics (ZK) on the end-to-end setups (Figure 4): ZK aggregation is approximately $10^6\times$ slower than vanilla, while ZK query processing is 10^4–$10^5\times$ slower.

computation logic. Query proof generation completes in 80.5 s, while proof verification remains lightweight (tens of milliseconds).

Vehicle emission dataset. The vehicle emission dataset uses vehicle attributes (e.g., manufacturer, model, fuel type) as keys and CO_2 emission values as measurements. Compared to the cloud and network workloads, this dataset has 9,727 unique keys and a smaller overall size (10,058 records). We issue histogram queries over CO_2 emissions grouped by vehicle manufacturer. Figure 4(c) shows that aggregation proof generation completes in 28.7 min, while query proof generation takes 693.3 s, with small verification overhead.

Vanilla baseline. To isolate the cost of the analytics architecture from that of zero-knowledge proving, we re-run zk-Analytics's pipeline with a vanilla (non-ZK) baseline for aggregation and query computation (Table 2). It shows ZK proving dominates: ZK aggregation is $\sim 10^6\times$ slower than vanilla, and ZK query processing 10^4–$10^5\times$ slower, with peak memory per node growing from ~33 MB to ~9.4 GB (~280×), driven by the prover, not the pipeline. The pipeline itself adds only modest, ZK-independent overhead: RocksDB insert and the subsequent read are identical across baselines; FDB writes roughly double under ZK, which also persists the ~225 KB proof. These overheads match prior measurements of general-purpose zkVMs [51], and ZK-specific optimizations, e.g., custom circuits, are orthogonal to our design and can further reduce proving cost without changing the architecture.

Summary. Across all datasets, offline aggregation proof generation dominates end-to-end latency, while online costs and proof verification overhead remain consistently low. Aggregation and query proof sizes stay compact. These results demonstrate that zk-Analytics enables feasible, end-to-end verifiable analytics over diverse moderate-size real-world workloads without significant verification overhead on verifiers.

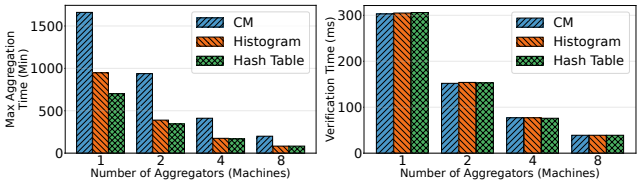
7.2 Microbenchmarks

Online log commitment. We benchmark the per-log cost of our SHA-256-based online commitment library under batched ingestion using a single thread. With batch size 1, the system can commit log entries at 1.6 million log entries/second, and with batch of 8 log entries, the throughput of commitment is 6.7 million log entries/second. As the batch size increases, the commitment throughput increases with batch sizes due to amortization of per-batch overheads of SHA-256 hash computation. Batching reduces commitment bandwidth overhead by amortizing a single 32-byte SHA-256 commitment across multiple log entries. Regarding the effective commitment bytes per log entry as a function of batch size, at batch size of 1 log entry, each log entry carries a full 32-byte commitment (139% overhead relative to a 23-byte log entry), whereas at batch size of 8 log entries, the amortized commitment cost falls to 4 bytes/log entry (≈17% overhead).

Distributed batch aggregation. We evaluate the scalability of zk-Analytics's offline batch aggregation by varying aggregators from 1 to 8, with a commitment batch of 8 logs, an epoch of 16,384 logs, and 131,072 log entries across 4,096 distinct series. Figure 5 reports the maximum aggregation time, verification time, memory usage, and total proof size across aggregators, under three modes: hash table of raw logs, histograms, and Count-Min (CM) sketches. Figure 5(a) shows aggregation time decreases nearly linearly with the number

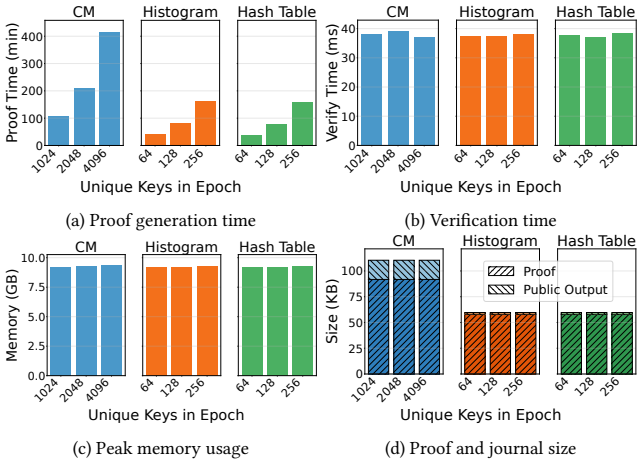
Aggregation Type	Metric/Number of Aggregators	1	2	4	8
Count-Min (CM)	Proof Size (KB)	2096.6	1956.6	1886.6	1851.6
	Public Output Size (KB)	44.1	26.6	17.9	13.5
	Peak Memory (GB)	9.27	9.35	9.35	9.34
Histogram	Proof Size (KB)	2064.6	1924.6	1854.6	1819.6
	Public Output Size (KB)	40.1	22.6	13.9	9.5
	Peak Memory (GB)	9.30	9.26	9.25	9.25
Hash table of Raw Logs	Proof Size (KB)	2064.6	1924.6	1854.6	1819.6
	Public Output Size (KB)	40.1	22.6	13.9	9.5
	Peak Memory (GB)	9.24	9.24	9.25	9.25

Table 3: Distributed total aggregator proof size, public output size, and peak memory usage during runtime.



(a) Max proof generation time (b) Max verification time

Figure 5: Distributed offline batch aggregation performance: Increasing the number of batch aggregation machines reduces proof generation time approximately linearly.



(a) Proof generation time (b) Verification time

(c) Peak memory usage (d) Proof and journal size

Figure 6: Single machine batch aggregation performance with varying distinct series (Zipf distribution of keys) and epoch sizes across three epoch types using all threads: Aggregation proof generation time grows linearly with the number of input log entries, with small, constant-size proofs per aggregation type and constant-time verification.

of aggregators, indicating effective parallelization; with 8 aggregators it drops more than 7× versus a single machine. CM sketches incur the highest cost due to their more complex update logic (3 rows × 1024 columns, with a top-100 heap), while histograms and raw logs complete faster. Figure 5(b) shows aggregation proof verification completes within hundreds of milliseconds and scales favorably with more aggregators. Table 3 shows proof sizes stay compact (below a few megabytes) and largely independent of aggregator count, since the epoch partition is consistent across configurations. Total public output shrinks with more aggregators: each epoch's output includes the hash-chain tips of its assigned sources, so spreading sources across more aggregators reduces the tips maintained per epoch. Peak memory per aggregator remains stable across configurations and modes. Overall, zk-Analytics's distributed aggregation scales effectively while keeping proof sizes practical.

Single-machine batch aggregation. We benchmark batch aggregation on a single machine using all 56 Rayon threads [71], varying the distinct keys per epoch across three modes: Count-Min (CM) sketches, histograms, and hash tables of raw logs. These suit different epoch shapes: CM sketches for many distinct keys, histograms and hash tables for fewer keys with longer series histories. Figure 6 reports proof generation time, verification time, memory usage, and proof size; the largest configuration processes up to 32,768 log entries per epoch, with larger epochs containing proportionally more input data. Figure 6(a) shows proof generation time scales roughly linearly with epoch size and aggregation complexity, reaching ~400 min for CM sketches at 4,096 keys, while histogram and raw-log aggregation finish within 180 min at 256 keys. Verification time, proof size, and peak memory stay largely constant (Figures 6(b–d)): verification completes in tens of milliseconds, proofs stay compact thanks to their succinctness [7], and memory stays around 9–10 GB. Thus input size primarily drives aggregation proof generation cost, while verification and resource usage remain stable.

Query processing. We evaluate the offline query component on synthetic datasets with 8,192 distinct keys for Count-Min Sketch epochs and 1,024 keys for Histogram and Hash table epochs, each epoch holding 8,192 logs, varying the queried epochs from 1 to 256. Figure 7 reports global aggregation (sum over all selected epochs), per-key sum, Top-K (hash table and Count-Min Sketch epochs), single-key frequency estimation (Count-Min Sketch), and percentile queries (Histogram). Figure 7(a) shows that proof generation time grows with the number of queried epochs for all query types as the input data grows. Complex queries (Top-K, percentile) cost more, and Count-Min Sketch queries are generally slower than raw-log and histogram queries at scale. Verification time stays stable at tens of milliseconds across all settings (Figure 7(b)). Proof sizes remain nearly constant across epoch counts and query types (Figure 7(c)), while public output grows with the queried epochs (Figure 7(d)).

8 Related Work

Multi-Party Computation. Prior work has applied secure Multi-Party Computation (MPC) to privacy-preserving analytics. NetQuery [94] proposes using MPC and trusted hardware for cross-domain network queries, where operators of different domains may be mutually distrusting. Private analytics systems [38, 85] combine MPC with client-side proofs to compute aggregate statistics over user data without revealing individual inputs. These approaches typically assume that multiple non-colluding parties aggregate data, or that clients participate in relatively heavy-weight cryptographic protocols. zk-Analytics targets a different security model, where analytics are collected for one domain/platform, the aggregation

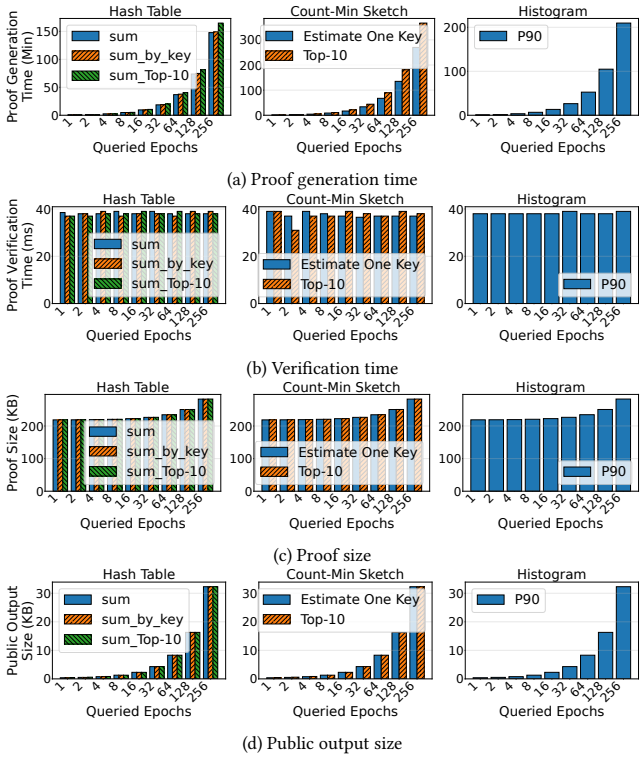


Figure 7: Offline query performance with varying number of epochs across three epoch types: Proof generation time grows linearly with queried data size, with small proofs and constant-time verification.

server can see raw data, and we have weaker security assumptions for collusion/bad behavior from clients/servers.

Zero-Knowledge Databases. Zero-knowledge databases [65, 108] enable verifiable SQL queries whose results are provably correct without revealing the underlying records, but focus on proving query execution over a *pre-committed* database state. zk-Analytics instead addresses the broader analytics pipeline: continuous log generation, commitment, aggregation, and query execution. [30] verifies cloud SLAs via TEE-based monitoring, blockchain-backed storage, and zero-knowledge proofs; zk-Analytics minimizes trusted hardware, replaces blockchain storage with lightweight append-only log commitments, and supports a broader class of verifiable analytics over distributed telemetry. zkStream [97] verifies streaming-operator execution and authenticates inputs with per-message signatures, whereas zk-Analytics targets end-to-end verifiability across the telemetry pipeline.

Differential Privacy. Differential privacy (DP) and zero-knowledge proofs are complementary techniques. DP provides statistical privacy guarantees by adding calibrated noise to query results under a fixed privacy budget [46, 47], whereas zk-SNARKs cryptographically guarantee correct computation. While zk-SNARKs hide raw logs and intermediate analytics state from verifiers, they do not prevent information leakage through query outputs themselves. Prior work has explored combining DP with zk-SNARKs to provide both privacy and verifiability [25, 49, 78]. zk-Analytics is complementary to these approaches and could incorporate DP mechanisms to provide stronger output-privacy guarantees.

End-user-driven measurement. Another line of work proposes that verifiers participate as end users to independently collect logs and perform analytics, using techniques such as active probing (e.g., Ookla Speedtest [81]) or synthetic traffic replay to measure end-to-end performance, rather than relying on centralized analytics providers. While these approaches reduce dependence on self-reported measurements from analytics providers, they suffer from several limitations. Probe workloads may not accurately reflect real application behavior [18]; moreover, probing traffic can be treated preferentially by the infrastructure (e.g., ISPs), leading to biased or misleading results [1]. Finally, end-user-driven measurements provide only a partial view, as they cannot capture analytics over the full set of workloads traversing the analytics provider.

Streaming sketches and approximate analytics. Streaming sketches provide compact summaries for high-volume data streams, enabling approximate frequency estimation [31, 37], heavy-hitter detection [20, 31], cardinality estimation [48], and quantile/distribution queries with bounded memory [62, 70]. A rich line of work has developed sketches for network and systems analytics, including universal monitoring with UnivMon [67], robust high-speed software sketching with NitroSketch [66], partial-key query support with CocoSketch [109], programmable switch sketch libraries with SketchLib [77], and distributed sketch execution with OctoSketch [107]. These systems show that sketch-based analytics can provide accurate, resource-efficient analytics across various distributed network settings [9]. zk-Analytics is complementary to this line of work: it does not introduce new sketch algorithms, but can make sketch construction and query execution verifiable by proving that the published sketch state was correctly derived from committed logs.

9 Conclusions

We present zk-Analytics, a verifiable cloud analytics framework that enables cryptographically verifiable analytics without disclosing raw data or provider infrastructure. zk-Analytics combines lightweight append-only log commitments with zero-knowledge proofs to attest to the correctness of aggregation and query execution over authentic telemetry. By decoupling online log commitment from offline distributed aggregation and query verification, zk-Analytics achieves scalable verification with feasible overheads, demonstrating that general verifiable, privacy-preserving analytics is within our reach for real-world cloud workloads.

Ethics: This work does not raise any ethical issues.

Acknowledgments

We thank the anonymous reviewers, our shepherd, and Vyas Sekar for their valuable feedback. We also thank Amol Deshpande and Daniel Abadi for helpful discussions on distributed and streaming analytics. This work is supported in part by NSF grants IIS-2544434 and CNS-2415754.

References

[1] 2014. T-Mobile forced to stop hiding slow speeds from throttled customers. https://arstechnica.com/information-technology/2014/11/t-mobile-forced-to-stop-hiding-slow-speeds-from-throttled-customers/.

[2] 2019. The CAIDA UCSD Anonymized Internet Traces. https://www.caida.org/catalog/datasets/passive_dataset/. Online.

[3] 2023. Intel Trust Domain Extensions. https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-v4.pdf. Online.

[4] 2024. FoundationDB. https://www.foundationdb.org/. Distributed multi-model database with ACID transactions.

[5] 2025. AWS Nitro System. https://aws.amazon.com/ec2/nitro/.

[6] 2025. Prometheus – Monitoring system & time series database. https://prometheus.io/. https://prometheus.io/ Open-source systems monitoring and alerting toolkit; Cloud Native Computing Foundation graduated project.

[7] 2026. Proof System Overview — Types of Receipts. https://dev.risczero.com/proof-system#types-of-receipts.

[8] 2026. RocksDB: A persistent key-value store. https://rocksdb.org/.

[9] Anup Agarwal, Zaoxing Liu, and Srinivasan Seshan. 2022. HeteroSketch: Coordinating network-wide monitoring in heterogeneous and dynamic networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 719–741.

[10] Anette Alén-Savikko. 2019. Network neutrality in the era of 5G–a matter of faith, hope, and design? *Information & Communications Technology Law* 28, 2 (2019), 115–130.

[11] AMD. 2023. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. https://www.amd.com/en/processors/amd-secure-encrypted-virtualization. Online.

[12] Jaechan An, Zeying Zhu, Ian Miers, and Zaoxing Liu. 2025. Towards Verifiable Network Telemetry without Special Purpose Hardware. In *Proceedings of the 24th ACM Workshop on Hot Topics in Networks (HotNets '25)*. Association for Computing Machinery, New York, NY, USA, 411–418. doi:10.1145/3772356.3772392

[13] Apache Software Foundation. 2026. Apache Kafka. https://kafka.apache.org/.

[14] Arctic Wolf. 2026. Dashboards. https://docs.arcticwolf.com/en-us/arctic-wolf-unified-portal/unified-portal/reporting/dashboards. Describes dashboards summarizing security-relevant data that an organization sends to Arctic Wolf.

[15] Arctic Wolf. 2026. Managed Detection and Response (MDR). https://docs.arcticwolf.com/bundle/arctic_wolf_solutions/page/managed_detection_and_response.html.

[16] Katerina Argyraki, Petros Maniatis, Olga Irzak, Subramanian Ashish, and Scott Shenker. 2007. Loss and delay accountability for the Internet. In *2007 IEEE International Conference on Network Protocols*. IEEE, 194–205.

[17] Arasu Arun, Srinath Setty, and Justin Thaler. 2023. Jolt: SNARKs for Virtual Machines via Lookups. Cryptology ePrint Archive, Paper 2023/1217. https://eprint.iacr.org/2023/1217

[18] Arjun Balasingam, Manu Bansal, Rakesh Misra, Kanthi Nagaraj, Rahul Tandra, Sachin Katti, and Aaron Schulman. 2019. Detecting if lte is the bottleneck with bursttracker. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–15.

[19] Gilles Barthe, Pedro R D'argenio, and Tamara Rezk. 2004. Secure information flow by self-composition. In *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.* IEEE, 100–114.

[20] Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. 2016. Heavy hitters in streams and sliding windows. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. 1–9.

[21] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. 2013. SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*. 90–108. doi:10.1007/978-3-642-40084-1_6 https://doi.org/10.1007/978-3-642-40084-1_6.

[22] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, San Diego, CA, 781–796. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson

[23] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2012. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference* (Cambridge, Massachusetts) *(ITCS '12)*. Association for Computing Machinery, New York, NY, USA, 326–349. doi:10.1145/2090236.2090263

[24] Dan Boneh and Victor Shoup. 2023. *A Graduate Course in Applied Cryptography*. https://crypto.stanford.edu/~dabo/cryptobook/draft_0_2.pdf

[25] Tariq Bontekoe, Hassan Jameel Asghar, and Fatih Turkmen. 2025. Efficient Verifiable Differential Privacy with Input Authenticity in the Local and Shuffle Model. *Proceedings on Privacy Enhancing Technologies* 2025, 2 (April 2025), 543–563. doi:10.56553/popets-2025-0076

[26] Sean Bowe, Jack Grigg, and Daira Hopwood. 2019. Halo: Recursive Proof Composition without a Trusted Setup. *IACR Cryptology ePrint Archive* 1021 (2019). https://eprint.iacr.org/2019/1021

[27] Jeremy Bruestle, Paul Gafni, and RISC Zero Team. 2023. RISC Zero zkVM: Scalable, Transparent Arguments of RISC-V Integrity. Whitepaper. https://dev.risczero.com/proof-system-in-detail.pdf.

[28] Kevin Buck, Diane Hanf, and Daniel Harper. 2015. Cloud SLA considerations for the government consumer. (2015).

[29] Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs. Cryptology ePrint Archive, Paper 2019/142. doi:10.1145/3319535.3339820

[30] Fernando Castillo, Eduardo Brito, Sebastian Werner, Pille Pullonen-Raudvere, and Jonathan Heiss. 2025. Towards Trusted Service Monitoring: Verifiable Service Level Agreements. *arXiv preprint arXiv:2510.13370* (2025).

[31] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding frequent items in data streams. In *Automata, Languages and Programming: 29th International Colloquium, ICALP 2002 Málaga, Spain, July 8–13, 2002 Proceedings 29*. Springer, 693–703.

[32] Chen Chen, Petros Maniatis, Adrian Perrig, Amit Vasudevan, and Vyas Sekar. 2013. Towards verifiable resource accounting for outsourced computation. In *Proceedings of the 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. 167–178.

[33] Benoit Claise. 2004. *Cisco systems netflow services export version 9*. Technical Report.

[34] Coalition, Inc. 2024. Why MDR is the Next MFA for Cyber Insurance. https://www.coalitioninc.com/blog/cyber-insurance/why-mdr-is-the-next-mfa. Carriers require managed detection/response in underwriting.

[35] Lois Colby. 2026. Trust Services Criteria (TSCs) for SOC 2 Reports. https://linfordco.com/blog/trust-services-critieria-principles-soc-2/. Linford & Company LLP. CC7.3–7.4: security-event and breach response procedures under the System Operations criteria.

[36] Conviva. 2025. DPI Overview. https://docs.conviva.com/learning-center-files/content/a_common/dpi_overview.html.

[37] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[38] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX symposium on networked systems design and implementation (NSDI 17)*. 259–282.

[39] Scott A Crosby and Dan S Wallach. 2009. Efficient data structures for tamper-evident logging. In *USENIX security symposium*. 317–334.

[40] CrowdStrike. 2026. How Customized Dashboards Help Security Teams. https://www.crowdstrike.com/tech-hub/endpoint-security/customized-dashboards/. Shows CrowdStrike dashboards filtering detections by severity and time, including critical and high severity detections in a selected time frame..

[41] Databricks, Inc. 2026. Databricks – Data and AI Platform for Enterprises. https://www.databricks.com/.

[42] Trisha Datta, Binyi Chen, and Dan Boneh. 2024. VerITAS: Verifying Image Transformations at Scale. Cryptology ePrint Archive, Paper 2024/1066. https://eprint.iacr.org/2024/1066

[43] Dell Technologies. 2024. Managed Detection and Response with CrowdStrike. https://i.dell.com/sites/csdocuments/Legal_Docs/en/us/managed-detection-and-response-with-crowdstrike-sd-en.pdf.

[44] Sireesha Devalla. 2025. AI-Driven Telemetry Analytics for Predictive Reliability and Privacy in Enterprise-Scale Cloud Systems. *International Journal of Artificial Intelligence, Data Science, and Machine Learning* 6, 2 (2025), 125–134.

[45] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. Rocksdb: Evolution of development priorities in a key-value store serving large-scale applications. *ACM Transactions on Storage (TOS)* 17, 4 (2021), 1–32.

[46] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and trends® in theoretical computer science* 9, 3–4 (2014), 211–407.

[47] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. 1054–1067.

[48] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete mathematics & theoretical computer science* Proceedings (2007).

[49] Alexander Frolov, Hal Triedman, and Ian Miers. 2025. zk-Cookies: Continuous Anonymous Authentication for the Web. *Cryptology ePrint Archive* (2025).

[50] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. *IACR Cryptology ePrint Archive* 953 (2019). https://eprint.iacr.org/2019/953

[51] Thomas Gassmann, Stefanos Chaliasos, Thodoris Sotiropoulos, and Zhendong Su. 2026. Evaluating compiler optimization impacts on zkvm performance. In

Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. 697–714.

[52] Craig Gentry and Daniel Wichs. 2011. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*. 99–108.

[53] Joseph A Goguen and José Meseguer. 1982. Security policies and security models. In *1982 IEEE symposium on security and privacy*. IEEE, 11–11.

[54] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract). In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, Robert Sedgewick (Ed.). ACM, 291–304. doi:10.1145/22145.22178

[55] Google. 2020. Google ClusterData 2019 Traces. https://github.com/google/cluster-data/blob/master/ClusterData2019.md.

[56] Google. 2024. Trillian: A Transparent, Highly Scalable and Cryptographically Verifiable Data Store. https://github.com/google/trillian.

[57] Google Cloud. 2026. BigQuery: From Data Warehouse to Autonomous Data and AI Platform. https://cloud.google.com/bigquery?hl=en

[58] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. 2021. Poseidon: A new hash function for {Zero-Knowledge} proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*. 519–535.

[59] Intel Corporation. 2026. Intel Software Guard Extensions (Intel SGX). https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html.

[60] Harshavardhan Kamarthi, Harshil Shah, Henry Milner, Sayan Sinha, Yan Li, B Aditya Prakash, and Vyas Sekar. 2026. AHA: Scalable Alternative History Analysis for Operational Timeseries Applications. *arXiv preprint arXiv:2601.04432* (2026).

[61] Partha Kanuparthy and Constantine Dovrolis. 2011. ShaperProbe: end-to-end detection of ISP traffic shaping using active methods. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. 473–482.

[62] Zohar Karnin, Kevin Lang, and Edo Liberty. 2016. Optimal quantile approximation in streams. In *2016 ieee 57th annual symposium on foundations of computer science (focs)*. IEEE, 71–78.

[63] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. 2022. Nova: Recursive zero-knowledge arguments from folding schemes. In *Annual International Cryptology Conference*. Springer, 359–388.

[64] Ben Laurie, Adam Langley, and Emilia Kasper. 2013. Certificate Transparency. RFC 6962. doi:10.17487/RFC6962 https://www.rfc-editor.org/rfc/rfc6962.

[65] Xiling Li, Chenkai Weng, Yongxin Xu, Xiao Wang, and Jennie Rogers. 2023. Zksql: Verifiable and efficient query evaluation with zero-knowledge proofs. *Proceedings of the VLDB Endowment* 16, 8 (2023).

[66] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. 2019. Nitrosketch: Robust and general sketch-based monitoring in software switches. In *Proceedings of the ACM Special Interest Group on Data Communication*. 334–350.

[67] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 101–114.

[68] Haarika Manda, Varshika Srinivasavaradhan, Laasya Koduru, Kevin Zhang, Xuanhe Zhou, Udit Paul, Elizabeth Belding, Arpit Gupta, and Tejas N Narechania. 2024. The efficacy of the connect America fund in addressing US internet access inequities. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 484–505.

[69] Antonis Manousis, Zhuo Cheng, Ran Ben Basat, Zaoxing Liu, and Vyas Sekar. 2022. Enabling efficient and general subpopulation analytics in multidimensional data streams. *Proc. VLDB Endow.* 15, 11 (July 2022), 3249–3262. doi:10.14778/3551793.3551867

[70] Charles Masson, Jee E Rim, and Homin K Lee. 2019. Ddsketch: A fast and fully-mergeable quantile sketch with relative-error guarantees. *arXiv preprint arXiv:1908.10693* (2019).

[71] Niko Matsakis and Josh Stone. 2024. Rayon: Simple Work-Stealing Parallelism for Rust. https://crates.io/crates/rayon. Version 1.10.0, accessed 2026-07-01.

[72] Jämes Ménétrey, Christian Göttel, Anum Khurshid, Marcelo Pasin, Pascal Felber, Valerio Schiavoni, and Shahid Raza. 2022. Attestation mechanisms for trusted execution environments demystified. In *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 95–113.

[73] Ralph C. Merkle. 1988. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology — CRYPTO '87*, Carl Pomerance (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 369–378.

[74] Silvio Micali. 2000. Computationally sound proofs. *SIAM J. Comput.* 30, 4 (2000), 1253–1298.

[75] Arash Molavi Kakhki, Abbas Razaghpanah, Anke Li, Hyungjoon Koo, Rajesh Golani, David Choffnes, Phillipa Gill, and Alan Mislove. 2015. Identifying traffic differentiation in mobile networks. In *Proceedings of the 2015 Internet Measurement Conference*. 239–251.

[76] Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-based fault injection attacks against Intel SGX. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1466–1482.

[77] Hun Namkung, Zaoxing Liu, Daehyeok Kim, Vyas Sekar, and Peter Steenkiste. 2022. SketchLib: Enabling efficient sketch-based monitoring on programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 743–759.

[78] Arjun Narayan, Ariel Feldman, Antonis Papadimitriou, and Andreas Haeberlen. 2015. Verifiable differential privacy. In *Proceedings of the Tenth European Conference on Computer Systems*. 1–14.

[79] Natural Resources Canada. 2025. *Fuel consumption ratings*. https://open.canada.ca/data/en/dataset/98f1a129-f628-4ce4-b24d-6f16bf24dd64 Data update: July 24, 2025; model-specific fuel consumption ratings and estimated carbon dioxide emissions for new light-duty vehicles in Canada.

[80] Pavlos Nikolopoulos, Christos Pappas, Katerina Argyraki, and Adrian Perrig. 2019. Retroactive packet sampling for traffic receipts. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3, 1 (2019), 1–39.

[81] Ookla. 2025. Speedtest by Ookla – The Global Broadband Speed Test. https://www.speedtest.net/.

[82] OpenTelemetry Community. 2026. Metrics Data Model. https://opentelemetry.io/docs/specs/otel/metrics/data-model/. OpenTelemetry Specification.

[83] OpenTelemetry Community. 2026. OpenTelemetry: An Open-Source Observability Framework. https://opentelemetry.io/.

[84] PCI Security Standards Council. 2022. *PCI DSS v4.0, Requirement 10: Log and Monitor All Access*. Technical Report. PCI SSC. Mandates daily review of security events/logs as audit evidence.

[85] Mayank Rathee, Yuwen Zhang, Henry Corrigan-Gibbs, and Raluca Ada Popa. 2024. Private analytics via streaming, sketching, and silently verifiable proofs. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3072–3090.

[86] Oscar Reparaz, Josep Balasch, and Ingrid Verbauwhede. 2017. Dude, is my code constant time?. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 1697–1702.

[87] RISC Zero. 2026. Precompiles. https://dev.risczero.com/api/zkvm/precompiles.

[88] Sasha Romanosky, Lillian Ablon, Andreas Kuehn, and Therese Jones. 2019. Content analysis of cyber insurance policies: how do carriers price cyber risk? *Journal of Cybersecurity* 5, 1 (2019), tyz002. doi:10.1093/cybsec/tyz002

[89] Michael Rosenberg, Tushar Mopuri, Hossein Hafezi, Ian Miers, and Pratyush Mishra. 2024. Hekaton: Horizontally-scalable zkSNARKs via proof aggregation. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 929–940.

[90] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*. IEEE, 459–474.

[91] Vinnie Scarlata, Simon Johnson, James Beaney, and Piotr Zmijewski. 2018. *Supporting Third Party Attestation for Intel® SGX with Intel® Data Center Attestation Primitives*. Technical Report. Intel Corporation. https://cdrdv2-public.intel.com/671314/intel-sgx-support-for-third-party-attestation.pdf.

[92] Bruce Schneier and John Kelsey. 1998. Cryptographic support for secure logs on untrusted machines. In *USENIX Security Symposium*, Vol. 98. San Antonio, TX, 53–62.

[93] Vyas Sekar and Petros Maniatis. 2011. Verifiable resource accounting for cloud computing services. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. 21–26.

[94] Alan Shieh, Emin Gün Sirer, and Fred B Schneider. 2011. NetQuery: A knowledge plane for reasoning about network properties. In *Proceedings of the ACM SIGCOMM 2011 conference*. 278–289.

[95] Sigstore. 2024. Rekor: Software Supply Chain Transparency Log. https://github.com/sigstore/rekor.

[96] Snowflake, Inc. 2026. Snowflake AI Data Cloud. https://www.snowflake.com/en/.

[97] Janwillem Swalens, Lode Hoste, Emad Heydari Beni, and Lieven Trappeniers. 2024. zkStream: a Framework for Trustworthy Stream Processing. In *Proceedings of the 25th International Middleware Conference*. 252–265.

[98] Justin Thaler. 2022. *Proofs, Arguments, and Zero-Knowledge*. Now Publishers. https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK

[99] The CloudLab Team. 2025. *CloudLab: Flexible Scientific Infrastructure for Cloud Computing Research*. CloudLab. https://www.cloudlab.us/

[100] The systemd Project. 2025. *systemd: The System and Service Manager*. freedesktop.org / systemd project. https://systemd.io

[101] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient {Out-of-Order} execution. In *27th USENIX Security Symposium (USENIX Security 18)*. 991–1008.

[102] Alexander Van Renen and Viktor Leis. 2023. Cloud analytics benchmark. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1413–1425.
[103] Guanhua Wang, Sudipta Chattopadhyay, Arnab Kumar Biswas, Tulika Mitra, and Abhik Roychoudhury. 2020. KLEESpectre: Detecting information leakage through speculative cache attacks via symbolic execution. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 29, 3 (2020), 1–31.
[104] Zhemin Yang, Min Yang, Yuan Zhang, Guofei Gu, Peng Ning, and X Sean Wang. 2013. Appintent: Analyzing sensitive data transmission in android for privacy leakage detection. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 1043–1054.
[105] Hongli Zhang, Lin Ye, Jiantao Shi, Xiaojiang Du, and Mohsen Guizani. 2014. Verifying cloud service-level agreement by a third-party auditor. *Security and Communication Networks* 7, 3 (2014), 492–502.
[106] Xiaoli Zhang, Huayi Duan, Cong Wang, Qi Li, and Jianping Wu. 2019. Towards verifiable performance measurement over in-the-cloud middleboxes. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1162–1170.
[107] Yinda Zhang, Peiqing Chen, and Zaoxing Liu. 2024. {OctoSketch}: Enabling {Real-Time}, Continuous Network Monitoring over Multiple Cores. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 1621–1639.
[108] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 863–880.
[109] Yinda Zhang, Zaoxing Liu, Ruixin Wang, Tong Yang, Jizhou Li, Ruijie Miao, Peng Liu, Ruwen Zhang, and Junchen Jiang. 2021. CocoSketch: High-performance sketch-based measurement over arbitrary partial key query. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 207–222.

Appendices are supporting material that has not been peer-reviewed.

A Deferred Definitions

Definition A.1. A **zk-SNARK** (zero-knowledge Succinct Non-interactive ARgument of Knowledge) for a relation $\mathcal{R} = (x, w)$ is a triple of probabilistic polynomial time algorithms (Setup, Prove, Verify) where:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$, where pp are public parameters for the scheme.
- $\mathsf{Prove}(\mathsf{pp}, x, w) \to \pi$, where π is a proof showing that $(x, w) \in \mathcal{R}$.
- $\mathsf{Verify}(\mathsf{pp}, x, \pi) \to \{0, 1\}$, where 1 means that π is a valid proof with public values x, and 0 means π is invalid.

A zk-SNARK must be complete, knowledge-sound and zero-knowledge. Additionally, it must be non-interactive and succinct. We now define these properties, largely following the definitions from [42]. These definitions use standard cryptographic notation.

- **Completeness**: if $(x, w) \in \mathcal{R}$, then verifying a proof π based on (x, w) should succeed. In other words, for all $\lambda \in \mathbb{N}$ and $(x, w) \in \mathcal{R}$:

$$\Pr\left[\begin{array}{lcl} \mathsf{Verify}(\mathsf{pp}, x, \pi) = 1 & : & \mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda) \\ & & \pi \leftarrow \mathsf{Prove}(\mathsf{pp}, x, w) \end{array}\right] = 1$$

- **Knowledge Soundness**: if an adversary can produce a valid proof for some x, then there should be a polynomial time extractor that can compute a witness w such that $(x, w) \in \mathcal{R}$. That is, the proof system Π has knowledge error $\epsilon \in [0, 1]$ if for every probabilistic polynomial time (p.p.t.) adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ there exists a probabilistic polynomial time extractor $\mathcal{E}$ such that:

$$\Pr\left[\begin{array}{lcl} (x, w) \in \mathcal{R} & : & \mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda) \\ & & (x, \mathsf{state}) \xleftarrow{\$} \mathcal{A}_0(\mathsf{pp}) \\ & & w \xleftarrow{\$} \mathcal{E}^{\mathcal{A}_1(\mathsf{state})}(\mathsf{pp}, x) \end{array}\right] \geq$$
$$\Pr\left[\begin{array}{lcl} \mathsf{Verify}(\mathsf{pp}, x, \pi) = 1 & : & \mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda) \\ & & (x, \mathsf{state}) \xleftarrow{\$} \mathcal{A}_0(\mathsf{pp}) \\ & & \pi \xleftarrow{\$} \mathcal{A}_1(\mathsf{state}) \end{array}\right] - \epsilon$$

- **Zero-Knowledge**: We state the definition in the random oracle model where all the algorithms are oracle machines that can query an oracle $H : \mathcal{X} \to \mathcal{Y}$ for some finite sets $\mathcal{X}$ and $\mathcal{Y}$. A zk-SNARK is zero knowledge if there is a p.p.t. simulator $\Pi.\mathcal{S}$ such that for all $(x, w) \in \mathcal{R}$ and all $p.p.t.$ adversaries $\mathcal{A}$, the following function is negligible

$$\mathsf{Adv}^{zk}_{\mathcal{A},\Pi}(\lambda) := \left|\begin{array}{r} \Pr\left[\mathcal{A}^H(\mathsf{pp}, x, \mathsf{Prove}^H(\mathsf{pp}, x, w)) = 1\right] - \\ \Pr\left[\mathcal{A}^{H[h]}(\mathsf{pp}, x, \pi)) = 1\right] \end{array}\right|$$

  where $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ and $(\pi, h) \xleftarrow{\$} \Pi.\mathcal{S}(\mathsf{pp}, x)$. Here h is a partial function $h : \mathcal{X} \to \mathcal{Y}$ output by $\Pi.\mathcal{S}$, and $H[h]$ refers to the oracle $H : \mathcal{X} \to \mathcal{Y}$ modified by entries in h. That is, we allow $\Pi.\mathcal{S}$ to program the oracle H.
- **Non-interactive**: the proof is non-interactive, and a proof created by the prover can be checked by any verifier.
- **Succinct**: the proof size and verifier runtime are $o(|w|)$. The verifier can run in linear time in $|x|$.

Definition A.2. A commitment scheme Comm is a collection of algorithms (Comm, Open) where:

- $\mathsf{Comm}(x, r) \to c$ gives a commitment c to x with blind r.
- $\mathsf{Open}(c, x, r) \to \{0, 1\}$ returns a bit, indicating whether c is a valid commitment to x with blind r.

B Security Analysis

We state zk-Analytics's guarantees and their limits.

Assumptions. (A1) each data source honestly produces log entries (ts, key, $value$), commits to them per Alg. 1, and periodically publishes its hash-chain checkpoints to a public append-only transparency log TLog_s. (A2) CommitHash is collision-resistant and preimage resistant. (A3) the zk-SNARK is complete, knowledge-sound, and zero-knowledge (Def. A.1), with public parameters pp produced by an honest (or transparent) setup. (A4) Commit is binding and hiding (Def. A.2), and each epoch commitment $c^E_k = \mathsf{Commit}(\mathsf{state}_k; r_k)$ uses fresh randomness r_k. (A5) the transparency logs TLog^E and $\{\mathsf{TLog}_s\}$ are append-only, publicly readable, and non-equivocating, so every party observes a single consistent view; the set $\mathcal{S}$ of sources contributing in W is determined by $\{\mathsf{TLog}_s\}$, and the window endpoints align with published checkpoints. (A6) AggregateFunc and Merge are deterministic and output correct statistics over aggregated data.

Attested Statements. Fix an allowed query q over window W, relabel its epochs $k = 1, \ldots, n$, and let $\mathcal{S}$ be the contributing sources, and write buf_k for the epoch k buffer buf of Alg. 2. Each aggregation

proof π_k (Alg. 2), on public input $(h^E_{k-1}, \{inTip_k[s]\}_{s\in\mathcal{S}})$ and output $(h^E_k, c^E_k, \{outTip_k[s]\}_{s\in\mathcal{S}})$, attests that: (a) for each s, applying CommitHash as part of a hash chain over s's batches from $inTip_k[s]$ reproduces every claimed batch hash and yields $outTip_k[s]$; (b) $\text{state}_k = \text{AggregateFunc}(\text{buf}_k)$; (c) $c^E_k = \text{Commit}(\text{state}_k; r_k)$; and (d) $h^E_k = \text{CommitHash}(\text{TAG\_EPOCH} \,\|\, h^E_{k-1} \,\|\, c^E_k)$. The query proof π (Alg. 3), on public input $(h^E_0, \{(h^E_k, c^E_k)\}^n_{k=1})$ and output $(h^E_{\text{start}}, h^E_{\text{end}}, res)$, attests that: (1) each c^E_k opens to state_k; (2) the h^E_k form a valid chain from h^E_0; and (3) $res = q(\text{Merge}(\{\text{state}_k\}^n_{k=1}))$.

Acceptance Conditions. The verifier (Alg. 4) accepts iff: (V1) π and every π_k verify; (V2) the $\{(h^E_k, c^E_k)\}$ used by π equal those in the per-epoch outputs $\{out_k\}$; (V3) $outTip_k[s] = inTip_{k+1}[s]$ for all s and consecutive k; and (V4) $inTip_1[s]$ and $outTip_n[s]$ match s's checkpoints in TLog_s for every contributing $s \in \mathcal{S}$, with h^E_0 descending from a published checkpoint in TLog^E.

Limitations. We define correctness relative to an honest computation: the guarantee is that the per-epoch summary states and their merge are *bit-for-bit identical* to those an honest party computes over the same committed logs. Whether the resulting answer matches the true statistic over the raw entries exactly (additive statistics, e.g., count, sum, min, max) or only up to the summary's stated error bound (approximate summaries, e.g., exponential histograms, Count-Min sketches) is a property of the chosen structure, independent of zk-Analytics. This theorem also has all the limitations discussed in Section 2.1.

Theorem B.1 (End-to-end integrity). *Under (A1)–(A6), for any allowed q over W, if the verifier accepts per (V1)–(V4) then, except with probability negligible in λ, the aggregation data structure* $\text{Merge}(\{\text{state}_k\}^n_{k=1})$ *is identical to the one an honest party computes over the* unique, complete, correctly ordered *log entries that each contributing source $s \in \mathcal{S}$ committed within W (completeness holding at checkpoint granularity), and the public output $res = q(\text{Merge}(\{\text{state}_k\}^n_{k=1}))$ is its faithful evaluation. By (A6), this answer matches the true statistic over those entries exactly when exactly collecting log entries, and up to the summary's stated error bound for approximate summaries. In particular, a malicious P.P.T. provider (prover) cannot (i) alter, drop, reorder, or inject committed entries of a contributing source, (ii) substitute aggregation state inconsistent with those logs, or (iii) report $res' \neq q(\text{Merge}(\cdot))$, without breaking knowledge-soundness, binding, or collision-resistance of the underlying cryptographic primitives.*

Proof sketch. We show that an accepting transcript with an incorrect *res* would break one of the assumed primitives, an event with probability negligible in λ; correctness rests on four facts. *First*, each epoch state is honestly aggregated: by knowledge soundness (A3), from each accepting π_k we extract a witness (buf_k, r_k) satisfying (a)–(d), so its batches recompute s's chain from $inTip_k[s]$ to $outTip_k[s]$ and c^E_k binds to state_k. *Second*, the query uses exactly those states: by (V2) the (h^E_k, c^E_k) used by π are the per-epoch outputs, and extracting from π yields (state'_k, r'_k) opening the same c^E_k, which commitment binding (A4) forces to equal state_k; hence the merged data structure is byte-identical to an honest party's and $ans = q(\text{Merge}(\cdot))$ is its faithful evaluation, matching the true statistic exactly for queries over exact logs and within the summary's stated error bound otherwise (A6). *Third*, the window is complete and non-equivocating: (V3) makes the per-source boundaries contiguous across W, (V4) pins the endpoints to the sources' checkpoints in $\{\text{TLog}_s\}$ at granularity P, and by (A5) the contributing set $\mathcal{S}$ and the epoch history are fixed by the public logs, so the provider can neither omit a contributing source nor present divergent histories. *Finally*, tampering is caught: by collision-resistance (A2), any altered, missing, reordered, or injected batch changes a recomputed hash-chain link and fails an aggregation assert or an endpoint match. Combining these, an accepting transcript with incorrect *res* implies a hash collision (A2), a break of knowledge-soundness (A3) or binding (A4), each of which have probability negligible in λ, so acceptance implies *res* is correct except with negligible probability. □

Theorem B.2 (Privacy). *Under zero-knowledge (A3) and hiding (A4), the proofs $\pi, \{\pi_k\}$ reveal nothing* to verifiers *about raw log entries or intermediate aggregation state beyond the public values the protocol declares: the result res, the hiding commitments $\{c^E_k\}$, the epoch tips $\{h^E_k\}$, and the per-source boundaries $\{inTip_k[s], outTip_k[s]\}$. (§2.1).*

Limitations. Concretely, zk-Analytics provides *content* privacy for individual log values and keys; it does not hide the structural metadata these boundaries expose, namely the per-source, per-epoch batch counts, the set $\mathcal{S}$ of contributing source identifiers, and the number of epochs n. These identifiers are the system-level source ids s, not the sources' real-world identities, which can be made pseudonymous independently of zk-Analytics. Privacy holds against verifiers only: the provider already holds the raw logs and intermediate state in plaintext, so it is trusted for *confidentiality* (logs do not have to be hidden from them) but remains untrusted for *integrity* (the system must enforce honest computation from the provider).

Proof sketch. Aggregation and query run inside zk-SNARKs that take the raw logs and intermediate state as private witness data and expose only the statements as public input/output; we show a verifier's entire view is simulatable from the declared public values. *First*, the proofs leak nothing about their witnesses: through a hybrid over the n+1 proofs $\{\pi_k\}^n_{k=1}$ and π, zero-knowledge (A3) lets us replace each real proof with the simulator $\Pi.\mathcal{S}$ run on its public values alone, bounding the distinguishing advantage by a negligible value. *Second*, the commitments hide the epoch states: each $c^E_k = \text{Commit}(\text{state}_k; r_k)$ uses fresh r_k, so by hiding (A4) it is computationally indistinguishable from a commitment to a fixed dummy value, and substituting all c^E_k this way reveals nothing about state_k. *Third*, the epoch tips add nothing: each $h^E_k = \text{CommitHash}(\text{TAG\_EPOCH} \,\|\, h^E_{k-1} \,\|\, c^E_k)$ is a deterministic function of public values and the hiding c^E_k, hence leaks no more than the c^E_k already do. *Finally*, the per-source boundaries and source checkpoints are outputs of CommitHash on committed batch prefixes; by the preimage resistance of CommitHash (A2), they are infeasible to invert and disclose only the stated structural metadata, but nothing about the hashed entry values or keys. Thus the entire view can be simulated from the declared public values, with no dependence on raw logs or intermediate state. □