# TVA: A multi-party computation system for secure and expressive time series analytics*

Muhammad Faisal
Boston University

Jerry Zhang†
University of California San Diego

John Liagouris
Boston University

Vasiliki Kalavri
Boston University

Mayank Varia
Boston University

**Abstract**

We present TVA, a multi-party computation (MPC) system for secure analytics on secret-shared time series data. TVA achieves strong *security guarantees* in the semi-honest and malicious settings, and high *expressivity* by enabling complex analytics on inputs with unordered and irregular timestamps. TVA is the first system to support arbitrary composition of oblivious window operators, keyed aggregations, and multiple filter predicates, while keeping all data attributes private, including record timestamps and user-defined values in query predicates. At the core of the TVA system lie novel protocols for secure window assignment: (i) a tumbling window protocol that groups records into fixed-length time buckets and (ii) two session window protocols that identify periods of activity followed by periods of inactivity. We also contribute a new protocol for secure division with a public divisor, which may be of independent interest. We evaluate TVA on real LAN and WAN environments and show that it can efficiently compute complex window-based analytics on inputs of $2^{22}$ records with modest use of resources. When compared to the state-of-the-art, TVA achieves up to $5.8\times$ lower latency in queries with multiple filters and two orders of magnitude better performance in window aggregation.

*This article is the full, extended version of a published article at USENIX Security 2023 [36]. This material is based upon work supported by the National Science Foundation under Grants No. 1915763 and 2209194, the DARPA SIEVE program under Agreement No. HR00112020021, a Red Hat Collaboratory grant (No. 2022-01-RH02), a Hariri Institute Focused Research Project Award, and a gift from Robert Bosch GmbH.

†Work completed at Boston University.

# Contents

# 1 Introduction

Time series analysis is used by organizations to understand the behavior of non-stationary data, monitor the evolution of metrics over time, identify trends, and predict future outcomes [12, 59, 81]. Time series data are ubiquitous in today's connected world and crucial for applications in IoT, smart cities, mobile health, traffic monitoring, and stock trading [72, 76, 97]. Applications leveraging multi-origin time series data are highly valuable to communities and businesses alike. For example, a smart grid company can monitor energy consumption across clients to improve conservation and reduce peak demand. Medical investigators can assess a drug's efficacy on patient cohorts by analyzing physiological signals of many individuals over time. Similarly, a smart city infrastructure can leverage location time series to optimize public transport scheduling. While highly valuable, such analyses raise privacy concerns for data holders, who wish to protect their sensitive or proprietary information.

Cryptographically secure collaborative analytics have recently become more accessible and practical thanks to advances in secure multi-party computation (MPC) [71], fully homomorphic encryption [45], and Oblivious RAM [46, 47]. In this work, we focus on the decentralized trust setting where MPC has been successfully used for machine learning [67, 74, 94, 95], relational queries [69, 79, 89], graph analysis [7], aggregate statistics [3, 15, 56], and other operations such as sorting and PSI [9, 68]. Even though existing MPC-based solutions cover a wide range of collaborative use cases, they are not suitable for analytics on time series data.

Time series computations are commonly expressed with advanced time-oriented operators, called *windows* [4, 59, 85]. Window operators group events into meaningful intervals of time and allow computing aggregation functions on the contents of each group. For example, a *tumbling* window operator divides a time series into fixed-sized time intervals and can be used to compute analytics such as the per-hour energy consumption of a smart grid. Window intervals can also be defined by custom metrics and data-dependent values. A *session* window, for instance, identifies periods of "activity" followed by periods of "inactivity" and can be used to compute analytics such as the average glucose level in a patient cohort during their sleep or their exercise.

The challenge of providing efficient and expressive secure collaborative analytics on time series data has not been successfully addressed yet. Existing approaches either sacrifice expressivity to achieve good performance with strong security guarantees [30] or allow information leakage in return for additional functionality [17, 18]. More importantly, prior works do not support secure windowing and assume *regular*, *ordered*, and *public* timestamps [17, 18, 23, 30, 58, 84]. This assumption is problematic for two reasons. First, missing values and out-of-order records are common in practice, as time series data are typically produced by distributed and unreliable sources (e.g., sensors). Second, time series analytics are often performed offline for forecasting and pattern mining [81]. In such cases, timestamps must be protected to prevent untrusted parties from inferring the event distribution and learn sensitive information, such as rare incidents or activities of individuals.

In this paper we present TVA (*Time-Varying Analytics*), a secure and expressive time series analysis system that leverages outsourced MPC. TVA protects data by distributing trust and generalizes the functionality of prior works without compromising security or performance. To mitigate the tension between these two, TVA employs new protocols, cross-layer optimizations, and vectorized primitives that enable it to scale to large datasets with modest use of resources.

## 1.1 Limitations of existing systems

TVA addresses the following limitations of prior work:

**Lack of advanced window functionality.** To extract value from time series data, analysts must be able to query the evolution of metrics over time. Such temporal analysis is typically performed with window operators that group events into consecutive time intervals of fixed (tumbling) or custom (session) length. Waldo [30] is optimized for *snapshot* queries that operate on a single time interval and, as we show in §6.1, it cannot efficiently perform recurring computations. TimeCrypt [17] and Zeph [18] provide limited support for tumbling windows but also leak information to untrusted parties.

By contrast, TVA is the first system to efficiently support both snapshot and arbitrary recurring queries. It offers generic window operators that can efficiently compute aggregates over fixed and custom time intervals, with strong security guarantees through its black-box use of MPC primitives.

**No support for complex filters with keyed aggregations.** Besides temporal operators, time series analysis typically requires relational transformations, such as filters, sorting, grouping, and aggregations (keyed and global). These operators must be customizable (to support arbitrary predicates) and composable (to express arbitrary analysis tasks). For example, users should be able to define multiple filter conditions along with complex aggregations on time and any other base or derived attributes. Keyed aggregations allow computing functions on logical partitions of the data, e.g., the hourly energy consumption *per postcode*. Prior work either supports global aggregation only [30] (total consumption for all postcodes) or allows limited additive keyed aggregation by requiring data holders to pre-encode the attribute domain (all possible postcodes) and to periodically transmit values [18, 58].

In contrast, TVA's aggregation operators can be composed with any operator and can be also *chained*. For instance, TVA can compute the average energy consumption per postcode over 1-hour windows, which can then be used to compute the maximum average consumption across all windows.

**No support for unordered and irregular timestamps.** Time series data are commonly produced by distributed sources, such as sensors or wearable devices. As a result, events may be ingested out of order due to network delays and clock skew. Furthermore, sources may transmit data at a fixed frequency (e.g., heart rate every minute) or only when an event occurs (e.g., smoke is detected). State-of-the-art approaches assume in-order and regular timestamps [17, 18, 23, 30, 58, 84], that is, they expect data owners to provide data at fixed time intervals. As a result, data sources must pad the time series with dummy records when events are missing and pre-aggregate events locally when the event generation frequency is higher than the transmission rate. This approach simplifies the problem of providing privacy-preserving time series analytics, since regular and ordered timestamps do not reveal any information (other than the time domain) and can thus be kept public. However, it has two considerable drawbacks. First, when the time series events are irregular, padding can lead to significant communication and computation costs. Second, pre-aggregation comes at loss of data granularity and accuracy. Subsequent queries can only return exact results on time intervals that are multiples of the pre-aggregation bucket size.

On the contrary, TVA does not require the input time series to be regular and ordered by time. Both window and snapshot operators can correctly and efficiently operate on unordered and irregular timestamps, which are protected like all other data attributes. Due to its ability to operate on private timestamps, TVA can serve queries on both recent and historical data collected independently by different entities, without revealing the event distribution to computing parties.

## 1.2 Summary of contributions

**Protocols for secure window assignment (§4.1–§4.4).** We present the first oblivious algorithms for (global and keyed) tumbling and session window assignment on private time series data. We analyze the complexity of our protocols and prove their correctness and security guarantees.

Our tumbling window protocol relies on a new MPC primitive for secure division with public constants. The state-of-the-art protocols for fast division produce small errors that are acceptable in other use cases (e.g., machine learning applications [74]) but cause incorrect aggregation results in our setting, as records end up in wrong windows with high probability. To address this challenge, we propose error correction protocols for semi-honest and malicious-secure division that can be used to compute tumbling windows with a number of communication rounds independent of the time series size.

For session windows, the plaintext algorithms [4, 85] traverse the time series and update session boundaries while keeping track of the last open session. Although simple, an oblivious implementation of this technique would incur a prohibitive number of rounds that is linear to the dataset size. To address this challenge, we introduce a novel two-phase sessionization protocol that enables us to identify variable-length session windows with a logarithmic number of rounds. The first phase of the protocol marks the beginning of each session according to a windowing strategy and the second phase leverages these markers to complete the sessionization by performing an oblivious grouping.

The core idea of the second phase is to formulate sessionization as an odd-even merge circuit [10]. This task is not straightforward, as directly applying the standard oblivious grouping algorithm [61] in our setting turns out to be both incorrect and inefficient. To tackle this problem, our protocol reverses access patterns to ensure correctness and uses an iterative control flow with a butterfly-like structure [11, 48] to facilitate fast vectorized execution and message batching. Our protocol is agnostic to the windowing strategy and can accommodate both timeout-based and

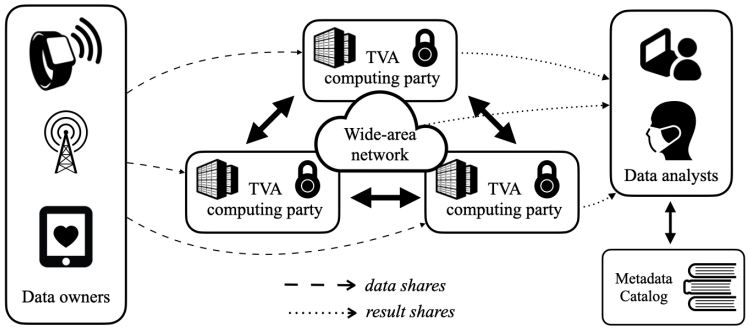


Figure 1: TVA system overview for secure time series analytics

threshold-based windows. At the same time, it is amenable to efficient composition with window aggregation, as we describe next.

**Efficient operator composition for complex time series analytics (§4.5, §4.6).** We develop a library of composable operators that can be combined with window operators to construct arbitrarily complex analytics by leveraging mixed-mode secret sharing and conversion protocols. Secure operator composition requires additional interaction among computing parties and straight-forward approaches lead to substantial overheads. To achieve optimal performance, we design efficient protocols that "fuse" expensive operators into a single oblivious control flow to avoid redundant computation and communication. One representative example is the fusion of sessionization with oblivious aggregation that reduces the overall computation and communication by $2\times$.

**Semi-honest and malicious security (§3).** TVA provides the security guarantees of MPC and can be configured to offer semi-honest [6] or malicious security [28]. To achieve this, we design oblivious operators that rely solely on the functionalities provided by the protocols $(+, *, \oplus, \wedge)$ and immediately inherit their security guarantees. This design choice provides a strong foundation to support more MPC protocols in the future by simply replacing the underlying functionalities. TVA keeps all data attributes private, including record timestamps, and can also protect user-defined values in the query itself (e.g., constants and queried time intervals) using secret sharing. This way, TVA protects both the time domain of the queried dataset and the actual filtering conditions, which is sensitive information in many use cases.

**System design and implementation (§5, §6).** We provide efficient implementations of oblivious operators and vectorized MPC primitives that amortize the communication costs of MPC. We also develop a declarative and protocol-agnostic query API that hides the complexity of composition from end users. All experiments we present in this paper use real LAN and WAN deployments. We have implemented the TVA software stack entirely from scratch, and the software is available as open source [37].

# 2 TVA in a nutshell

Figure 1 provides an overview of TVA. *Data owners* are individuals or entities who have agreed to contribute their private data towards a joint analysis (e.g., a medical study, a smart grid monitoring task, etc.), provided that the data remain hidden from untrusted parties. *Data analysts* are individuals who perform the analysis using TVA. To do so, they access a public catalog of time series metadata (cf. §3) and submit queries to the system. A *computing party* is a logical entity that may consist of multiple compute nodes (TVA servers). Computing parties are deployed in different trust domains (e.g., competing cloud providers) and can be configured to execute queries either with malicious (default) or semi-honest security, depending on the use case.

TVA operates in the outsourced setting [25], that is, it does not require data owners or analysts to participate in the computation (both can be offline during the analysis). Each data owner interacts with the computing parties independently from others, and only to distribute secret shares of their data.

We emphasize that the security of TVA stems from its data-oblivious execution (so the parties learn nothing

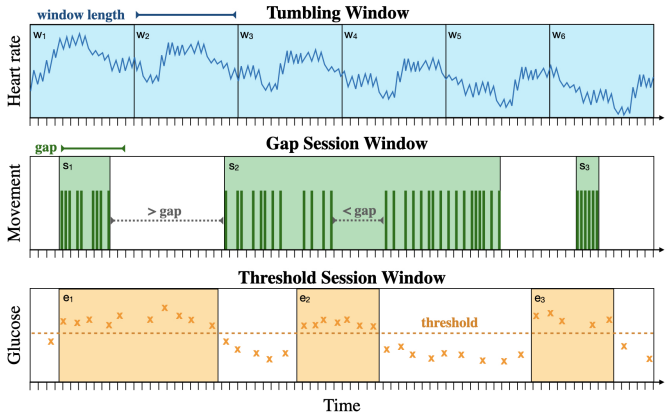


Figure 2: A visual representation of TVA's window operations on health time series data. The tumbling window splits the heart rate measurements into six fixed-length windows. The gap session window generates three activity sessions by grouping together movement signals that occur within a specified interval. The threshold window identifies three eating sessions by detecting intervals where glucose values continuously stay above a threshold.

beyond the result of the query) and access control (so even the result is only revealed to designated data analysts). In particular, TVA does not protect against inference attacks on the input based on the query results. Combining TVA with differentially private algorithms for time series data (e.g., [40,41,63,91]) remains an exciting possibility for future work.

## 2.1 Supported workloads

TVA offers a rich set of temporal and time-agnostic operators that can be composed arbitrarily to define complex time series analysis tasks.

**Temporal operators.** TVA's temporal operators can be used to express (i) *snapshot* queries that compute an aggregate over a specific time interval, and (ii) *window* queries that compute an aggregate over multiple time intervals defined by a windowing strategy. Consider a time series dataset collected from a heart rate tracker, a movement sensor, and a glucose monitor. Figure 2 illustrates TVA's windowing strategies using a digital health use case as an example.

A tumbling window operator (top) divides the time domain into non-overlapping intervals of equal length $\lambda$ so that each input record belongs to exactly one window. For example, the tumbling window can be used to compute *"the maximum heart rate per minute across individuals in a cohort"*.

A session window operator groups records into periods of activity (sessions) followed by periods of inactivity [4]. Session windows have variable and data-dependent length in time units. The *gap* window (middle) uses a timeout gap $\tau$ as the session delimiter. All records whose time difference is within an interval up to the timeout $\tau$ are grouped together into the same session. The end of a session is detected when the time difference between two consecutive records is greater than $\tau$. A gap-based window could be used to compute an aggregate over *"time periods when an individual is moving"*.

The *threshold* window operator (bottom) interprets sessions using a threshold $\theta \in \mathbb{R}$ and a function $q : S \to \mathbb{R}$, where $S$ is collection of data records. All consecutive records whose value $q(r), r \in S$, is at least equal to the threshold $\theta$ are grouped into the same session. The current session ends when a record with value below $\theta$ is detected. For example, a threshold window can identify *"eating periods when the glucose level exceeds a threshold"*. As shown in Figure 2, although every input record belongs to exactly one gap window, some records may not belong to any threshold window.

**Time-agnostic operators.** TVA provides a set of time-agnostic operators that can be combined with temporal operators. These include filtering, sorting, grouping, distinct (for duplicate elimination), and all common aggregations, i.e., `COUNT`, `SUM`, `MIN`, `MAX`, `TOP-K`, `AVG`, `STDEV`, and `PERCENTILE`. Additionally, TVA supports arbitrary User-defined

Functions (UDFs) constructed with its secure primitives. Those include logical and arithmetic operations provided by the protocols $(\wedge, \vee, \oplus, +, -, *)$, boolean addition, comparison operators $(\leq, \geq, =, \neq)$, and division with public constant.

### 2.2 Putting it all together: the TVA API

TVA exposes a high-level declarative API that lets users combine temporal operators with arbitrary filters, aggregations, and other time-agnostic operators. The API specification is given in Appendix G. Listing 1 shows an example query that computes a keyed tumbling window on the dataset of Figure 2. To maximize readability, TVA's API lets users refer to data attributes by name, as shown in the example code snippet. Once the data schema is defined, attribute names can be used as operator arguments. The query first selects records corresponding to patients weighting between 190 and 250 pounds and applies the `keyBy` operator to partition these records by age group. It then computes the maximum heart rates over 1-hour tumbling windows for each age group. The result is a new time series `hr` that can be given as input to another query.

```
// Define the time series data schema
TS ts = get_shares({"TIMESTAMP", "WEIGHT", "AGE_GROUP",
       "HEART_RATE", "MOVEMENT", "GLUCOSE", "MAX_RATE"});

// Compute the max hourly heart rate per age group over
// all participants whose weight is within a given range
TS hr = ts.filter("WEIGHT" > 190 AND "WEIGHT" < 250)
          .keyBy("AGE_GROUP")
          .tumbling_window("TIMESTAMP", 3600)
          .aggregate("HEART_RATE", "MAX_RATE", Agg::MAX);
```

Listing 1: An example tumbling window query using the TVA API

Analysts submit queries like the one above to TVA parties for execution and TVA compiles them into a secure MPC program. The query structure is public but the filter predicates and the queried time intervals are secret-shared by the TVA client application (run by the analysts).

## 3 Threat model and security guarantees

TVA protects all data, including timestamps, throughout the entire lifecycle of computations. Overall, TVA protects data privacy using non-colluding computing parties, authenticated network links, and end-to-end oblivious computation. TVA does not reveal any private inputs or intermediate data during the computation and only opens the final query result to the designated analysts. Some metadata about the time series is purposely made public and accessible to the data analysts so they can create their queries; the public metadata include the data schema, the type and number of columns, and the number of input records, as in prior works [17, 18, 30].

**Setting.** TVA assumes that data owners and analysts have previously agreed on a query to compute. The query structure is known to the computing parties, but predicates are protected with secret sharing. In particular, parties know the type of operators they compute (e.g., the window type or aggregation function) but TVA protects the actual filter values and the session window predicates (gap and threshold). Protecting these parameters ensures that the computing parties do not learn any specific time intervals that are being queried. When a tumbling window operator is used, TVA treats the window length as public. We emphasize that knowledge of the length does not reveal anything about specific time intervals, since the window operator is applied on the entire time series dataset. In the example of Listing 1, TVA parties know that the query involves a filter of the form `"WEIGHT" > X AND "WEIGHT" < Y` followed by keyed window aggregation (`MAX`) per hour, but they do not learn the filter values `190` and `250`.

**Protocols.** TVA currently provides two replicated secret sharing protocols with $N$ parties: (i) the semi-honest 3-party protocol by Araki et al. [6] ($N = 3$) and the malicious-secure Fantastic Four protocol by Dalskov et al. [28] ($N = 4$). We encode an $\ell$-bit string of secret data $s$ by splitting it into $N$ *shares* that individually have the uniform distribution over all possible $\ell$-bit strings (for privacy) and collectively suffice to specify $s$ (for correctness). Share generation is cheap and is done locally by the data owners. Each party $P_i$ receives $N - 1$ of the shares: hence, any two parties can reconstruct the secret but any single party cannot. TVA supports both boolean secret sharing $[\![x]\!]_\ell$ of length-$\ell$ bitstrings

| Symbol | Description |
|---|---|
| $[\![s]\!]_\ell$ | replicated boolean secret share of the length-$\ell$ bitstring $s$ |
| $\langle\!\langle u \rangle\!\rangle_\ell$ | replicated arithmetic secret share of $u$ in the ring mod $2^\ell$ |
| $\langle u \rangle_\ell$ | non-replicated 2-of-2 arithmetic secret share of $u$ in the ring mod $2^\ell$ |
| $\ell$ | length of a share representation in bits |
| $\ell_x$ | maximum length, in bits, for dividend $x$ in the division protocols |
| $\ell_c$ | bitlength of the public divisor $c$ (in the division protocols) plus one |
| $r$ | a data record of the form $r = (t,\ a_1, a_2, \ldots, a_k)$, where $r.t$ is the record's timestamp and $r.a_i$, $1 \le i \le k$, is a data value |
| $S$ | a time series of size $\|S\|$, i.e., a collection of $\|S\|$ data records |
| $W$ | a window in a time series $S$, i.e., a collection of data records $W \subseteq S$ |
| $\lambda$ | length of a tumbling window in time units |
| $\tau$ | timeout interval that defines a gap session window |
| $\theta$ | threshold value that defines a threshold session window |
| $s_{\uparrow k}(S)$ | sort operator on key $k$ (ascending) |
| $\gamma_k^G(S)$ | group-by operator on key $k$ with aggregation function $G$ |

Table 1: Notation used in the paper

and arithmetic secret sharing $\langle\!\langle x \rangle\!\rangle_\ell$ in the ring mod $2^\ell$, as well as primitives to convert one format to the other. By convention, we say that party $i$ holds the shares $x_i$ and $x_{i+1}$ for 3-party replicated sharing, or $x_i, x_{i+1}, x_{i+2}$ for 4-party replicated sharing.

**Threat model.** TVA supports either semi-honest or malicious security in the honest majority setting. When operating with semi-honest security, TVA can withstand adversaries who have three types of capabilities: (i) complete control over the network, (ii) the ability to compromise at most one computing party and passively eavesdrop on its internal state (e.g., memory contents, access patterns, and data sent/received) without altering its execution, and (iii) the ability to collude with one or more data owners to learn inputs into the query, or with the data analyst to learn the output of the query. When operating in the malicious setting, the adversary can additionally force the compromised computing party to actively deviate from the protocol arbitrarily. In this case, TVA provides security up to abort: the honest computing parties will stop and report an error to the analyst.

**Security guarantees.** TVA offers two types of security guarantees: (i) *privacy*, meaning that computing parties do not learn anything beyond the public metadata and the information held by any data owners or analysts they are colluding with, and (ii) *correctness*, meaning that all participants are convinced that the computation output is accurate. TVA inherits both guarantees from the underlying MPC protocols by always operating over secret-shared data using the underlying MPC primitives in a black-box manner, ensuring an oblivious control flow for each operator individually (§4.1-4.4) and jointly (§4.6), and never opening any intermediate results.

To show this claim formally, in this work we perform security analysis in the arithmetic black-box model, as shown in Appendix A. This is a hybrid model in which we presume the existence of a functionality $\mathcal{F}_{\text{abb}}$ that provides perfect correctness and privacy for the following operations: $\mathcal{F}_{\text{abb}}$.input and $\mathcal{F}_{\text{abb}}$.output to share and reconstruct secrets in either arithmetic or boolean representations, $\mathcal{F}_{\text{abb}}$.add and $\mathcal{F}_{\text{abb}}$.mult to compute arithmetic (mod $2^\ell$) or multi-bit boolean (mod 2) operations, and $\mathcal{F}_{\text{abb}}$.A2B and $\mathcal{F}_{\text{abb}}$.B2A to convert between the two representations. In the malicious setting, we also consider a shared input functionality $\mathcal{F}_{\text{abb}}$.INP in which *two* parties provide as input a common secret that they both wish to be shared, and the functionality detects and aborts if there is a discrepancy between the two. For readability, we sometimes use informal notation (e.g., $[\![c]\!]_\ell = [\![a]\!]_\ell \wedge [\![b]\!]_\ell$) rather than formal notation (e.g., $\mathcal{F}_{\text{abb}}.\text{mult}(\text{boolean}, \text{id}_a, \text{id}_b, \text{id}_c)$) in this work.

# 4 Secure time series data analysis in TVA

In this section, we describe TVA's oblivious operators for secure time series analysis. In §4.1, we describe our division protocol that is the core building block of the tumbling window operator in §4.2. In §4.3 and §4.4, we present two session window operators and, in §4.5, we discuss time-agnostic operators supported by TVA. Finally, in §4.6, we describe TVA's efficient composition techniques for creating complex time series analysis tasks. Table 1 summarizes the notation.

**Protocol 1:** DIVISION (semi-honest 3PC)

**Input :** Arithmetic sharing $\langle\langle x \rangle\rangle_\ell$ of a secret $x \in [0, 2^{\ell_x})$
Public integer $c \in [1, 2^{\ell_c - 1})$
**Output :** Quotient $\langle\langle z \rangle\rangle$ computed as follows.

1 Construct a 2-of-2 sharing $\langle y \rangle_\ell$ of $y \approx z$ as follows:
1.1 Party 1 locally computes $y_1 = \lfloor (x_1 + x_2)/c \rfloor$
1.2 Parties 2, 3 locally compute $y_2 = 2^\ell - \lfloor (2^\ell - x_3)/c \rfloor$
2 Construct 2-of-2 sharing $\langle s \rangle_{\ell_c}$ of $s$ computed as follows:
2.1 Party 1 locally computes $s_1 = (x_1 + x_2) \bmod c$
2.2 Parties 2, 3 locally compute $s_2 = (2^\ell - x_3) \bmod c$
3 Create replicated sharings $\langle\langle y \rangle\rangle_\ell, \langle\langle s \rangle\rangle_{\ell_c}$ as follows:
3.1 Use $\mathcal{F}_{\text{abb}}$.input to share $y_1$ and $y_2$ in the ring mod $2^\ell$
3.2 Use $\mathcal{F}_{\text{abb}}$.input to share $s_1$ and $s_2$ in the ring mod $2^{\ell_c}$
3.3 Locally compute $\langle\langle y \rangle\rangle_\ell = \langle\langle y_1 \rangle\rangle_\ell + \langle\langle y_2 \rangle\rangle_\ell$ and $\langle\langle s \rangle\rangle_{\ell_c} = \langle\langle s_1 \rangle\rangle_{\ell_c} - \langle\langle s_2 \rangle\rangle_{\ell_c}$ using $\mathcal{F}_{\text{abb}}$.add
4 Calculate replicated sharing $\langle\langle b \rangle\rangle_\ell$ of the bit $b = (s_1 \overset{?}{<} s_2)$:
4.1 Use $\mathcal{F}_{\text{abb}}$.A2B to convert $\langle\langle s \rangle\rangle_{\ell_c}$ into boolean shares $[\![ s ]\!]_{\ell_c}$
4.2 Set $[\![ b ]\!]_1$ equal to the most significant bit of $[\![ s ]\!]_{\ell_c}$
4.3 Use 1-bit conversion $\mathcal{F}_{\text{abb}}$.b2A on $[\![ b ]\!]_1$ to compute $\langle\langle b \rangle\rangle_\ell$
5 Locally compute $\langle\langle z \rangle\rangle_\ell = \langle\langle y \rangle\rangle_\ell - \langle\langle b \rangle\rangle_\ell$ using $\mathcal{F}_{\text{abb}}$.add

**Protocol 2:** DIVISION (malicious 4PC)

**Input :** Arithmetic sharing $\langle\langle x \rangle\rangle_\ell$ of a secret $x \in [0, 2^{\ell_x})$
Public integer $c \in [1, 2^{\ell_c - 1})$
**Output :** Quotient $\langle\langle z \rangle\rangle_\ell$ computed as follows.

1 Construct a 2-of-2 sharing $\langle y \rangle_\ell$ of $y \approx z$ as follows:
1.1 Parties 1 and 4 compute $y_1 = \lfloor (x_1 + x_2)/c \rfloor$
1.2 Parties 2 and 3 compute $y_2 = 2^\ell - \lfloor (2^\ell - x_3 - x_4)/c \rfloor$
2 Construct 2-of-2 sharing $\langle s \rangle_{\ell_c}$ of $s$ computed as follows:
2.1 Parties 1 and 4 compute $s_1 = (x_1 + x_2) \bmod c$
2.2 Parties 2 and 3 compute $s_2 = (2^\ell - x_3 - x_4) \bmod c$
3 Perform lines 3-5 of Protocol 1, replacing all instances of $\mathcal{F}_{\text{abb}}$.input with the Shared Input functionality $\mathcal{F}_{\text{abb}}$.INP

Let $r = (t, a_1, a_2, \ldots, a_k)$ be a data record with timestamp $t \geq 0$ (denoted with $r.t$) and attributes $a_i, 1 \leq i \leq k$ (denoted with $r.a_i$). The value $t$ corresponds to a real timestamp or a sequence number whereas the attributes can be arbitrary data. We define a time series $S$ as a collection of $|S|$ records indexed in ascending order of their timestamps, i.e., $\forall r_i, r_j \in S,\ r_i.t \leq r_j.t \Leftrightarrow i < j < |S| \in \mathbb{N}$. We stress that record indices are only used to simplify the presentation and TVA does *not* assume records to be physically ordered on their timestamps.

TVA window operators are applied to time series and generate new time series. In practice, windowing is performed by appending to each record a new attribute $w_{id}$ that contains a secret-shared window id computed under MPC.

## 4.1 Integer division by public divisor

TVA assigns records to tumbling windows by discretizing timestamps based on the window length (in time units). Several prior works provide efficient protocols for integer or fixed-point division, but they typically introduce a small rounding error, truncate only by powers of two, or are slower because they provide features that we do not need in our setting like perfect accuracy or private divisors (e.g., [21, 22, 27, 35, 35, 65, 74, 88]).

In this work, we contribute a new secure computation protocol for division-and-truncation, given mixed-mode MPC with replicated secret sharing, that can perform division by any public divisor *without a rounding error*. For

our application to tumbling windows, having some noticeable probability of error means that many data records are being assigned to the wrong window, which could significantly distort the subsequent time series analysis. That said, for improved efficiency we do allow for a negligible error with probability $2^{-\sigma}$, which is parameterized by a statistical security parameter $\sigma$. We provide a detailed comparison between our division protocol and prior works in Appendix D.

For simplicity, we show the TVA division algorithms in Protocols 1 and 2 in the case that the secret dividend $x$ and public divisor $c$ are unsigned (positive) integers. This suffices for our application, in which the dividends $x$ are timestamps and the window lengths $c$ are a positive number of time units. It is straightforward to extend our protocols to support signed fixed-point numbers, as long as the absolute value of the divisor is at least 1 (otherwise there is a risk of overflow).

For semi-honest 3-party MPC, our starting point is the probabilistic truncation protocol of Mohassel and Rindal [74], which can perform probabilistic truncation with only one round of communication. Their technique roughly corresponds to step 1 of Protocol 1. Starting from a dividend that is at most $\ell_x$ bits long ($x < 2^{\ell_x}$) and public divisor $c < 2^{\ell_c - 1}$, step 1 computes a *non-replicated* 2-of-2 arithmetic sharing $\langle y \rangle_\ell$, which we later convert to a replicated secret sharing in step 3. The value $y$ is close to the correct quotient $z = \lfloor \frac{x}{c} \rfloor$, but $y$ has two possible sources of error: an off-by-one rounding error occurs with probability ~1/2 (namely, when the fractional part of $\frac{2^\ell - x_3}{c}$ is greater than the fractional part of $\frac{x_1 + x_2}{c}$), and a large error occurs with negligible probability (when $x_3 < x$).

In lines 2-5 of Protocol 1, we compute the rounding error within MPC itself so we can eliminate it. In step 2, we calculate the fractional parts $s_1$ and $s_2$ that were the source of the potential off-by-one error. Then, we use mixed-mode operations to calculate the error bit $b$ based on whether $s_1 < s_2$. We calculate this inequality by computing $s = s_1 - s_2 \bmod 2^{\ell_c}$ where $\ell_c = |c| + 1$, and converting to boolean shares in order to isolate the most significant bit of $[\![s]\!]_\ell$ and set it equal to $[\![b]\!]_1$.

Protocol 1 makes only black-box use of mixed-mode MPC operations. This leads to a simple extension of our division algorithm to the malicious 4-party MPC setting, as shown in Protocol 2 where we achieve malicious security through redundancy. Specifically, we ensure that two parties perform each calculation; then, using the Shared Input (`INP`) protocol within Dalskov et al. [28], the honest parties can detect any discrepancy between them. Modularity also simplifies both the security analysis and the resulting implementation within TVA. In Appendix C, we prove that Protocols 1-2 are private and correct with probability $1 - 2^{-\sigma}$, where $\sigma = \ell - \ell_x$.

Protocols 1-2 require $\lceil \log(\ell_c) \rceil + 3$ communication rounds and $2\ell + 2\ell_c \lceil \log(\ell_c) \rceil + \ell_c + 1$ total bits of communication (plus some small additive overhead in the 4-party case to send the hashes in the protocol of Dalskov et al. [28]). Concretely, line 3 requires one communication round to distribute shares, line 4 requires $\lceil \log(\ell_c) \rceil + 1$ rounds to perform the arithmetic-to-boolean conversion using a Parallel Prefix Adder (PPA) [54] plus one round for the subsequent bit-to-arithmetic conversion, and the remaining lines do not require any communication. See Appendix D for a longer comparison of this division protocol with closely related works.

## 4.2 Tumbling window

We can now describe TVA's tumbling window operator. Let $S$ be a time series, $\lambda \in \mathbb{N}^*$ a window length (in time units), and $G$ an aggregation function on data records.

**Definition 1 (Tumbling Window Operator)** *Given a window length $\lambda$ and an aggregation function $G$,* TVA*'s tumbling window operator* $\mathrm{T}$ *defines a new time series as follows:*

$$\mathrm{T}_G(S, \lambda) = \{ (k,\ G(W_k)) \mid W_k = \{r_i\},\ r_i \in S, \\ \lambda \cdot k \le r_i.t < \lambda \cdot (k+1),\ k \in \mathbb{N} \}$$

The new time series defined by the tumbling window operator contains records of the form $r = (t,\ G(W_k))$, where $t = k$ is the timestamp of the new record, $W_k$ is the $k$-th tumbling window, i.e., the collection of records that fall into the time interval $\lambda \cdot k \le r_i.t < \lambda \cdot (k+1)$, and $G(W_k)$ is the result of the aggregation function on $W_k$.

The secure tumbling window operator works as follows. First, parties iterate over the records' shares in the time series $S$ and compute the discretized timestamps $\left\lfloor \frac{t}{\lambda} \right\rfloor$ by executing the secure division protocol (Protocols 1 and 2). All these divisions are independent of each other and are performed in bulk, i.e., they can all be completed in $\lceil \log(\ell_c) \rceil + 3$ total rounds. The output of each division amounts to the tumbling window id $w_{id}$ of the respective record and is

**Protocol 3:** SESSIONIZATION

**Input :** A time series $S = \{r_0, r_1, \ldots, r_{|S|-1}\}$
**Result :** The updated time series with all sessions marked

SESSIONSTART($S$); //First phase
Let $d = 1$; //Distance between records in $S$
**while** $d < |S|$ **do**
  //All $|S| - d$ steps are independent
  **for** $i = 0;\ i < |S| - d;\ i$++ **do**
    Parties locally compute bit $[\![b]\!]_1$ for $b = (r_{i+d}.w_{id} \overset{?}{<} 0)$
    Parties locally compute mask $[\![m]\!]_\ell = \mathsf{expand}([\![b]\!]_1, \ell)$
    Parties compute the window id of $r_{i+d}$ as follows: $[\![r_{i+d}.w_{id}]\!]_\ell = [\![m]\!]_\ell \wedge [\![r_i.w_{id}]\!]_\ell \oplus [\![\overline{m}]\!]_\ell \wedge [\![r_{i+d}.w_{id}]\!]_\ell$
  **end**
  $d \leftarrow d * 2$;
**end**

**Protocol 4:** SESSIONSTART (GAP)

**Input :** A time series $S = \{r_0, r_1, \ldots, r_{|S|-1}\}$, a timeout $\tau$
**Param. :** Function $\delta$ (Eq. 1), timestamp length $\ell$ (in bits)

$s_{\uparrow t}(S)$; //Sort records by timestamp (ASC)
Let $\varepsilon = -1$ be an invalid timestamp; //Negative number
//All $|S|$ steps are independent
**for** $i = 0;\ i < |S|;\ i$++ **do**
  Parties compute bit $[\![b]\!]_1 = ([\![\delta(i)]\!]_\ell \overset{?}{>} [\![\tau]\!]_\ell)$
  Parties locally compute mask $[\![m]\!]_\ell = \mathsf{expand}([\![b]\!]_1, \ell)$
  Parties compute the window id of $r_i$ as follows: $[\![r_i.w_{id}]\!]_\ell = [\![m]\!]_\ell \wedge [\![r_i.t]\!]_\ell \oplus [\![\overline{m}]\!]_\ell \wedge [\![\varepsilon]\!]_\ell$
**end**

appended to the record as a new attribute[1]. In the next step, parties apply the aggregation function $G$ to the collection of records in each window. To do so, they group records by their window ids using TVA's group-by operator $\gamma$ that is based on the protocol by Jónsson et al. [61] (cf. §4.5-4.6). The output of the operator is a new time series $S' = \gamma^G_{w_{id}}(S)$ with one record per window containing the result of the aggregation $G$.

## 4.3 Gap-based session window

Computing sessions under MPC is challenging, as windows can be of variable length that depends on the data. Let $S$ be a time series and $\tau \in \mathbb{N}^*$ a timeout (in time units). Let also $\delta : \mathbb{N} \to \mathbb{R}_+ \cup \{\infty\}$ be a function that computes the time difference between consecutive records in $S$ as shown below:

$$\delta(i) = \begin{cases} r_i.t - r_{i-1}.t, & 1 \le i < |S| \\ \infty, & otherwise \end{cases} \tag{1}$$

**Definition 2 (Gap Window Operator)** *Given $\delta$ and an aggregation function $G$,* TVA*'s gap window operator $\Gamma$ defines a new time series as follows:*

$$\Gamma_{G,\delta}(S,\tau) = \{\ (r_s.t,\ G(W_s)) \mid W_s = \{r_i\},\ 0 \le s \le i \le e < |S|,\ \delta(s) > \tau,\ \delta(e+1) > \tau,\ \nexists r_j \in S : \delta(j) > \tau,\ s < j \le e\ \}$$

[1]If the record timestamp is not needed in a subsequent step of the analysis, TVA can simply overwrite it with $w_{id}$ instead of appending a new attribute.

Records in the new time series have the form $r = (t,\ G(W_s))$, where $W_s$ is a gap session window, $t = r_s.t$ is the earliest timestamp in $W_s$, and $G(W_s)$ is the result of the aggregation function $G$ on $W_s$. The most expensive condition in the gap session window operator is the last one: $\nexists r_j \in S : \delta(j) > \tau,\ s < j \leq e$. This condition requires identifying pairs of records $r_s$ and $r_e$ with no in-between records ($r_e$ included) that belong to a different session. Given a time series $S$ sorted by timestamp, the straight-forward approach to evaluate this condition is to scan $S$ and keep track of the current session by applying function $\delta$ to each pair of adjacent records. This approach, however, incurs prohibitive communication overhead, as it requires communication rounds linear to the number of records in $S$. Instead, we propose an efficient two-phase SESSIONIZATION protocol (Protocol 3) that can correctly identify all session windows in $S$ with a logarithmic number of rounds. We discuss the phases of the protocol next:

**Phase 1: Mark the beginning of each session.** In this phase, parties identify the beginning of each gap session window as shown in Protocol 4. To do so, they first order records on their timestamp using TVA's oblivious sort operator (this step can be omitted if records arrive in order). Next, they compare adjacent records by applying function $\delta$ (Eq. 1) using a PPA. Given a timeout $\tau$, a new session is marked at $r_i$ *iff* $\delta(i) > \tau$; the result of this inequality is saved as a bit $b$. Next, we expand the single bit $b$ into a mask $m$ of length $\ell$, such that every bit of $m$ is equal to $b$. Parties set the window id of each record using an oblivious multiplexer of the form $m \wedge x \oplus \overline{m} \wedge y$, where $m$ is a mask, and $\overline{m}$ is its boolean complement. Hence, if a record $r_i$ marks the start of a new window, parties set $r_i.w_{id} = r_i.t$; otherwise, they use an invalid timestamp $\varepsilon$ (e.g., a negative constant like $-1$) to denote that this record is currently unassigned. All steps of the for-loop in the first phase of the protocol do not depend on each other and incur a number of rounds that is independent of the time series size. That is, executing steps 3-7 in SESSIONSTART (GAP) requires $2\lceil \log(\ell + 1) \rceil + 1$ rounds in total, where $\ell$ is the length of the timestamp representation in bits (fixed).

**Phase 2: Assign records to sessions.** In the second phase of TVA's SESSIONIZATION (steps 2-10 in Protocol 3), parties assign the remaining records to their corresponding session in $\left\lceil \log |S| \right\rceil$ rounds. This is done by replacing the negative $w_{id}$ of each unassigned record with the non-negative $w_{id}$ of the nearest preceding record. Records in $S$ are guaranteed to be in order after the first phase, therefore, the earliest record in each session always precedes the remaining unassigned records. Our protocol leverages this ordering to reduce interaction during window assignment as follows. In the first iteration, parties examine adjacent records at distance $d = 1$ starting from the first record in $S$. If $r_{i+d}.w_{id} < 0$ (unassigned), it is overwritten by $r_i.w_{id}$, otherwise it is left as is. This is done using an oblivious multiplexer and a mask $m$ that is generated locally, as in the first phase. Step 5 is local because $b$ corresponds to the sign of $r_{i+d}.w_{id}$, i.e., its most significant bit. At the end of each iteration, parties double the distance $d$ of the previous step and repeat the same process. All for-loop iterations in the second phase of the protocol are independent one another, i.e., executing steps 4-8 in SESSIONIZATION requires a single round (due to multiplexing), for any time series $S$. The overall control flow has a butterfly-like structure that has been used for multi-set operations [11] (e.g., to compute cardinality) and element compaction [48]. In TVA, we leverage this technique to perform sessionization and, as we show in §4.6, efficient composition with window aggregation for a wide range of functions. We prove the correctness and privacy of our protocol in Appendix E.

When sessionization terminates, parties can use TVA's group-by operator to apply the aggregation function $G$ to each window. As the number of records grows, TVA's operator $\Gamma$ takes $O(\log^2 |S|)$ rounds and $O(|S| \log^2 |S|)$ communication if records arrive out of order, since TVA relies on bitonic sort (§4.5). For ordered timestamps, the asymptotic costs are $O(\log |S|)$ and $O(|S| \log |S|)$ respectively. We provide concrete communication costs in Appendix F.

## 4.4 Threshold-based session window

TVA's threshold window operator works similarly to gap session window. Let $q : S \to \mathbb{R}$ be a function on data records in a time series $S$. In its simplest form, $q$ is a function that returns a record's attribute but, in general, it can be any function constructed using TVA's primitives. Let $h : \mathbb{N} \to \mathbb{R} \cup \{-\infty\}$ be the function:

$$h(i) = \begin{cases} q(r_i), & 0 \leq i < |S| \\ -\infty, & otherwise \end{cases} \tag{2}$$

**Protocol 5:** SESSIONSTART (THRESHOLD)

**Input :** A time series $S = \{r_0, r_1, \ldots, r_{|S|-1}\}$, a threshold $\theta$
**Param. :** Function $h$ (Eq. 2), timestamp length $\ell$ (in bits)

1 $s_{\uparrow t}(S)$; //Sort records by timestamp (ASC)
2 Let $\varepsilon = -1$ be an invalid timestamp; //Negative number
//All $|S|$ comparisons are independent
3 **for** $i = 0;\ i < |S|;\ i$++ **do**
4 Parties append single-bit attribute $a_b$ to $r_i$
5 Parties compute bit $[\![r_i.a_b]\!]_1 = ([\![h(i)]\!]_\ell \overset{?}{>} [\![\theta]\!]_\ell)$
6 **end**
//All $|S|-1$ steps are independent
7 **for** $i = 1;\ i < |S|;\ i$++ **do**
8 Parties compute $[\![b]\!]_1 = ([\![r_{i-1}.a_b]\!]_1 \oplus 1) \wedge ([\![r_i.a_b]\!]_1 \oplus 0)$
9 Parties locally compute mask $[\![m]\!]_\ell = \mathsf{expand}([\![b]\!]_1, \ell)$
10 Parties compute the window id of $r_i$ as follows: $[\![r_i.w_{id}]\!]_\ell = [\![m]\!]_\ell \wedge [\![r_i.t]\!]_\ell \oplus [\![\overline{m}]\!]_\ell \wedge [\![\varepsilon]\!]_\ell$
11 **end**

**Definition 3 (Threshold Window Operator)** *Given $h$ and an aggregation function $G$,* TVA*'s threshold window operator $\Theta$ defines a new time series as follows:*

$$\Theta_{G,h}(S,\theta) = \{\ (r_s.t,\ G(W_s)) \mid W_s = \{r_i\},\ 0 \le s \le i \le e < |S|,\ h(s) \ge \theta,\ h(s-1) < \theta,\ h(e) \ge \theta,\ h(e+1) < \theta,\ \nexists r_j \in S : h(j) < \theta,\ s < j < e\ \}$$

The boundaries of a threshold window $W_s$ are defined by a pair of records $r_s$ and $r_e$ that satisfy all conditions above. TVA identifies threshold windows using the same oblivious control flow from Protocol 3. The main difference with respect to gap session windows lies in the first phase of sessionization that is described in Protocol 5. In this case, parties identify the beginning of each threshold window by applying function $h$ (Eq. 2) to each record in $S$. A session starts at $r_i$ *iff* $h(i) \ge \theta$ and $h(i-1) < \theta$. To keep track of records that do not belong to any threshold window, parties store the result of the comparison $h(i) \ge \theta$ in a new single-bit attribute $a_b$ that is appended to record $r_i$. This bit is used in an oblivious multiplexer to mark the beginning of each session and is reused in the second phase of the protocol, as we describe later. Executing steps 3-11 in SESSIONSTART (THRESHOLD) requires $R_q + \lceil \log(\ell+1) \rceil + 2$, where $R_q$ is the number of rounds required to apply $q$ to a single record (independent of $|S|$). In the common case where $q(r) = r.a_i, 1 \le i \le k$, we have $R_q = 0$.

During the second phase, parties execute steps 2-10 in Protocol 3 with a minor change in step 5 that is now replaced by the operation $[\![b]\!]_1 = [\![[\![r_{i+d}.w_{id}]\!]_\ell \overset{?}{<} 0]\!]_1 \ \wedge\ [\![r_{i+d}.a_b]\!]_1$. For threshold windows, $b$ is a composite bit that denotes whether record $r_{i+d}$ belongs to a threshold window ($r_{i+d}.a_b = 1$) and is currently unassigned ($r_{i+d}.w_{id} < 0$). Computing $b$ requires a single round between parties since both $r_{i+d}.a_b$ and $r_{i+d}.w_{id}$ have been computed in the previous phase. As a result, steps 4-8 in SESSIONIZATION require two rounds in total for threshold windows (one more compared to gap session windows). The final aggregation step and the overall asymptotic costs in terms of rounds and bandwidth for threshold windows are the same with gap windows, as given in §4.3. We prove the correctness and privacy of our protocol in Appendix E.

## 4.5 Other TVA operators

In addition to windows, TVA provides a rich set of oblivious operators for secure time series analysis, namely, snapshot, filter, sorting, grouping, and distinct (for duplicate elimination). TVA's snapshot operator H has the same semantics as in prior work, that is, it applies an aggregation function $G$ to a collection of records $S$ whose timestamps fall into a user-defined time interval I. More formally:

$$\mathrm{H}_G(S, \mathrm{I}) \;=\; G(W) \mid W = \{r_i\},\; r_i \in S,\; r_i.t \in \mathrm{I} \tag{3}$$

Filters are logical predicates constructed with TVA's secure primitives $(+, -, \times, \oplus, \wedge, \vee, \neg, \leq, \geq, =, \neq)$ and may also include arithmetic operations, e.g., `Watt*Unit_Price` $\geq$ `$30`. TVA's equality, inequality, and parallel prefix adder [54] require $O(\log \ell)$ rounds and $O(\ell \log \ell)$ communication, where $\ell$ is the length of the secret-shared operands in bits.

TVA's sort operator $s_{\uparrow k}(S)$ is based on a sorting network that requires $O(\log^2 |S|)$ rounds and $O(|S| \log^2 |S|)$ communication *w.r.t* the number of records. We choose a sorting network because it is easily parallelizable and works well with duplicate values, however, there exist oblivious sorting algorithms with lower asymptotic costs (e.g., [8, 9, 24, 53]). Combining these techniques with TVA's protocols is an intriguing opportunity for future work.

The group-by operator $\gamma_k^G(S)$ uses the protocol by Jónsson et al. [61] and consists of two phases: a sorting phase where the input records are sorted on the grouping key(s), and an odd-even aggregation phase that applies the aggregation function $G$ to each group with $O(\log |S|)$ rounds and $O(|S| \log |S|)$ communication in total. Distinct is a special case of grouping.

## 4.6 Operator composition

In this section, we describe all non-trivial cases of operator composition in TVA. We stress that the details of the composition process are completely hidden from TVA users by the high-level query API.

**Composing filter and snapshot predicates.** TVA does not pose any restriction to the number of filter predicates a user can specify. Predicates can be combined via logical `AND` and `OR` operators; for example, `Weight` $< 140$ `lb OR (Weight` $> 150$ `lb AND Height` $\leq 6$ `ft)` is a valid composite filter. Each individual filter predicate $\phi$ appends a (secret-shared) single-bit attribute $a_\phi$ to each record it is applied to. Predicate composition requires `AND`ing or `OR`ing (under MPC) these attributes according to the abstract syntax tree (AST) of the composite filter. To reduce interaction between parties, TVA composes independent predicates in a binary tree of operations. This way, composing a series of $m$ result bits of the form $a_{\phi_1} \wedge a_{\phi_2} \wedge \cdots \wedge a_{\phi_m}$ can be done with $\log m$ rounds in total. In the end, each record has only one attribute $a_\phi = a_{\phi_1} \wedge a_{\phi_2} \wedge \cdots \wedge a_{\phi_m}$ that denotes whether the record "passes" the composite filter or not.

In case a filter is followed by a snapshot operator, the condition $r_i.t \in \mathrm{I}$ from Eq. 3 is treated as one more filter on timestamps (with its own $a_\phi$ attribute) and is composed with the rest of the filter predicates as described above.

**Composing session window with aggregation.** The strawman approach to compute the final aggregation on session windows is to first execute the SESSIONIZATION protocol (Protocol 3) and then use TVA's group-by operator $\gamma_{w_{id}}^G$ to apply the aggregation function $G$ to each window. Although correct, this two-step approach requires $2\times$ more operations and communication rounds compared to sessionization alone. In TVA we employ a more efficient composition technique that eliminates this overhead.

Recall that TVA's group-by operator is based on the oblivious control flow by Jónsson et al. [61, Algorithm 6]. This algorithm compares records at distance $d = |S| \;/\; 2$ and, at each step of the iteration, reduces the distance to half of that in the previous step. Sessionization, however, only works if we access the time series $S$ in the opposite way, i.e., when we start comparing adjacent records (at distance $d = 1$) and double the distance at each iteration. Interestingly, the oblivious aggregation by Jónsson et al. can be modified to work similarly, enabling us to perform the sessionization and group-by protocols within a single oblivious control flow. This approach saves $O(\log |S|)$ rounds and $O(|S| \log |S|)$ communication in total and requires minor protocol modifications. Specifically, we only need to add the following operation at the end of each for-loop in Protocol 3 (right after line 7):

$$\begin{aligned} \langle\!\langle r_i.a_g \rangle\!\rangle_1 = \langle\!\langle b \rangle\!\rangle_1 \cdot G(\langle\!\langle r_i.a_g \rangle\!\rangle_1, \langle\!\langle r_{i+d}.a_g \rangle\!\rangle_1) + \\ + (1 - \langle\!\langle b \rangle\!\rangle_1) \cdot \langle\!\langle r_i.a_g \rangle\!\rangle_1 \end{aligned} \tag{4}$$

Eq. 4 updates (in place) the value of the aggregated attribute $a_g$ of the $i$-th record. To do so, parties apply the aggregation function $G$ to the pair of records $(r_i,\, r_{i+d})$ and use arithmetic shares of the bit $b$ (already computed in Protocol 3) to perform the oblivious multiplexing of the form $b \cdot x + (1 - b) \cdot y$. Arithmetic shares of $b$ are constructed on the fly using the single-bit conversion protocol from Mohassel and Rindal [74] (semi-honest security) or Dalskov et al. [28] (malicious security). This conversion requires one round between parties.

The composition technique we described works with any function $G$ that is *commutative* and *associative*. This holds for all common aggregations, including `SUM`, `COUNT`, `MIN`, `MAX`[2], `AVG`, and `STDEV` (for the last two we must maintain the nominators and denominators separately).

**Composing `keyBy` with session window.** When a `keyBy` operator precedes a session window, the key $k$ used in `keyBy` must be taken into account in the first phase of SESSIONIZATION. In this case, records are ordered on the pair $(k, t)$, instead of their timestamp $t$ only, and the start of each session is marked when the default condition holds *or* the $i$-th record in the time series $S$ has a different key value from the previous one, i.e., $r_i.k \neq r_{i-1}.k, 0 \leq i < |S|$ (true for $i = 0$). This requires evaluating an extra equality predicate in step 4 of Protocol 4 and step 8 of Protocol 5.

**Composing multiple operators with aggregation.** The final aggregation is always computed with TVA's group-by operator but the actual grouping keys depend on the operators in the query. Recall that all operators besides `keyBy` append a secret-shared attribute to the records they are applied to: `filter` and `snapshot` append an attribute $a_\phi$ whereas `window` operators append an attribute $w_{id}$. Whenever such an attribute exists in the input of the aggregation, TVA uses it along with the key $k$ of the `keyBy` operator (if any) to construct a composite key for grouping. The more complex case is when the query contains a `filter`, a `keyBy`, and a `window` operator followed by an aggregation $G$. In this case, TVA applies the operator $\gamma^G_{a_\phi,k,w_{id}}(S)$ that returns one aggregated value for each group of records with the same composite key $(a_\phi, k, w_{id})$.

# 5 Implementation

We have developed TVA from scratch in C++. We use MPI [2] for inter-party communication and libsodium [1] for random number generation. TVA supports shares of size $2^\ell$, where $\ell$ can be configured by the users. The default length for attributes is $\ell = 32$ and for timestamps is $\ell = 64$.

TVA's architecture consists of three layers: (i) a runtime that is responsible for resource management and communication, (ii) a protocol layer that contains vectorized implementations of semi-honest and malicious-secure primitives $(+, *, -, \oplus, \wedge, \neg, /)$, and (iii) a library of oblivious temporal and time-agnostic operators. The TVA runtime is carefully implemented to load balance computation in the asymmetric malicious protocol and to trade off computation for communication depending on the available resources. For example, the `INP` protocol (cf. §4.1) requires a party to send a vector of shares, another party to send the vector's hash, and a third party to verify the hash. Hashing is used to reduce bandwidth consumption at the cost of increasing local computation. In low-latency environments, however, it is sometimes better if both parties send shares instead of hashing.

Computing parties can be deployed on premises or across multiple clouds. Analysts submit queries through a client application that exposes TVA's high-level API and runs a lightweight module for generating and shipping shares.

# 6 Evaluation

Our experimental evaluation is structured into three parts:

**Comparison with state-of-the-art.** In §6.1, we present a performance comparison with Waldo [30], the only publicly available time series database with strong security guarantees. For multi-predicate queries, TVA provides up to $5.8\times$ lower latency compared to Waldo in the malicious setting and $2\times$ lower latency in the semi-honest setting. For window queries, TVA is up to two orders of magnitude faster than Waldo, which becomes competitive only when the ratio of the window length over the whole time domain is relatively small.

**Performance in real-world applications.** In §6.2, we use three real use cases to evaluate TVA's performance in LAN and WAN with varying input sizes. We use queries that combine tumbling and session windows with filters and keyed aggregations. Our results demonstrate that TVA offers excellent performance for both continuous and historical analysis: it can successfully compute online queries with rigid time constraints and evaluate complex analytics on millions of input rows with modest use of resources.

[2] For `MIN` and `MAX`, parties execute the oblivious aggregation in Eq. 4 on boolean shares, since TVA comparison primitives operate on boolean shares.

| | Query latency (s) | | | |
|---|---|---|---|---|
| | Equality | | Range | |
| | Malicious | Semi-honest | Malicious | Semi-honest |
| **Waldo** | 11.94 | 1.7 | 11.82 | 1.65 |
| **TVA** | 2.056 | 0.876 | 3.833 | 1.583 |

Table 2: Performance evaluation of WaldoTable and TVA queries with 8 predicates on a time series dataset with $2^{20}$ records.

**Microbenchmarks and cloud costs.** In Appendix F, we dive deeper into TVA's performance and evaluate the scalability of window assignment protocols, the overhead of malicious security, the benefits of parallelization, and its cost. We show that TVA's primitives scale effectively as the input size grows and that primitives under the malicious protocol are only $2\times$ slower compared to the semi-honest. At the same time, increasing the number of compute threads reduces latency by up to $7\times$. Finally, our results verify TVA's ability to amortize I/O with message batching. Comparing the performance achieved in LAN with that of a WAN deployment with $200 - 250\times$ higher RTT, latency is only $4.6\times$ higher on a time series with $2^{22}$ records.

## 6.1 Comparison with Waldo

### 6.1.1 Differences between Waldo and TVA

We highlight a few important differences between the two systems that need to be considered when evaluating the performance results. First, `WaldoTable` requires data owners to pre-encode attributes into a $N \times 2^{\ell}$ table of one-hot vectors, where $N$ is the number of input records. In TVA, data owners only perform secret sharing (arithmetic or boolean, depending on the target query). Second, `WaldoTree` queries operate on public timestamps and return partial aggregates to the client, who is responsible for computing the global aggregation in the clear. On the other hand, TVA's parties compute on private timestamps and return the final aggregation result to the client. Neither the preprocessing (index creation) nor the post-processing time of Waldo are included in the results of §6.1.2. Finally, both systems operate in the honest-majority setting: for semi-honest security, they rely on the same protocol [6] but, for malicious security, TVA uses a more efficient 4-party protocol [28] that requires three honest parties as opposed to Waldo's 3-party protocol that requires two honest parties.

### 6.1.2 Performance results

TVA can comfortably operate on machines with modest resources and supports large feature sizes (as we show later), however, for the sake of the comparison in this section we use the experimental setting of the Waldo paper [30, §VII]. In particular, we deploy parties on `r5n.16xlarge` instances (64 vCPUs and 512GB RAM), we use 8-bit attributes for both systems, and we set the network RTT to $20ms$. We note that we were unable to reproduce some of the published Waldo performance results. In our comparison below, we use the result reported in the Waldo paper when it is better than the one we measured in our own experiments.

Since Waldo is primarily optimized for snapshot queries, we devise the following two experimental scenarios to perform a meaningful comparison. First, we execute a Waldo-style multi-predicate snapshot query in TVA and compare its performance with the respective `WaldoTable` query. Second, we express a TVA-style tumbling window query in Waldo by executing multiple concurrent snapshot queries on consecutive time intervals on `WaldoTree`. All experiments of this section use sorted time series, as Waldo does not support out-of-order records.

**Comparison with WaldoTable.** For this experiment we set the number of input records to $2^{20}$ and run composite filters with equality and range predicates. Each filter contains 8 predicates of the same type and has the form $p_1 \wedge p_2 \wedge \cdots \wedge p_8$. Table 2 shows the results. We see that TVA's performance exceeds that of the specialized `WaldoTable` data structure. This is not surprising, as `WaldoTable`'s computation and communication costs are linear in the input size and the round trips are linear in the number of predicates [30] (§VII-C). TVA can perform equality (resp. inequality) with three (resp. four) rounds and requires another three for the composition of the final result. Further performance benefits come from TVA's ability to effectively parallelize the computation and amortize network I/O with its vectorized

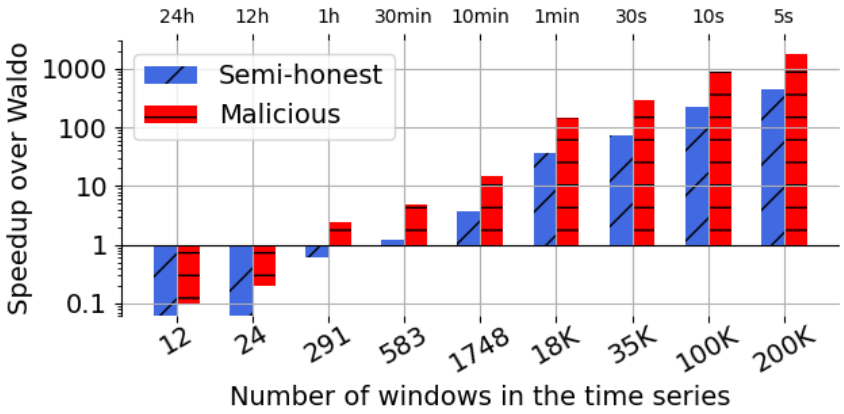


Figure 3: Speedup of TVA's tumbling window aggregation over WaldoTree on a time series dataset with $2^{20}$ records. The bottom x-axis shows the number of windows in the time series, which is equal to the number of searches Waldo performs for the respective query. Waldo achieves low latency for coarse-grained windows but its performance degrades significantly for fine-grained windows.

primitives. In the semi-honest case, TVA provides $1.94\times$ and $1.04\times$ lower latency than Waldo, for equality and range queries respectively. When operating with malicious security, the performance difference is more significant. TVA computes the equality query $5.8\times$ and the range query $3.08\times$ faster than Waldo.

**Comparison with WaldoTree.** In time series analytics, the window length can vary significantly depending on the use case. For example, monitoring applications compute fine-grained windows in the order of seconds, forecasting applications operate on intervals of minutes or hours, while anomaly detection involves comparing results over windows of different lengths [44]. To cover these diverse requirements, in this experiment, we use a time series with $2^{20}$ measurements recorded every second (~12 days of data in total). TVA needs 20 bits to encode the whole time domain, that is, the security parameter is $\sigma = 44$. We run tumbling window queries that compute a global aggregate per window and measure latency for both systems.

Figure 3 shows TVA's speedup over Waldo for window lengths ranging from $5s$ to $24h$ (top x-axis). The bottom axis shows the number of searches that Waldo needs to perform on the aggregation tree to compute the corresponding window. TVA's query latency is $6.8s$ (semi-honest) and $12.5s$ (malicious) irrespective of the window length. Recall that TVA's tumbling window cost does not depend on the window length $\lambda$ (§4.2) but Waldo needs to perform $\lceil \frac{T_{max}-T_{min}}{\lambda} \rceil$ searches, where $T_{min}$ and $T_{max}$ are the minimum and maximum record timestamps in the dataset. We see that TVA is orders of magnitude faster than Waldo for small windows, as Waldo's performance depends on the number of searches it performs in the underlying aggregation tree. When the ratio of the window length over the time domain becomes small, Waldo achieves better latency, as it performs fewer searches and the benefit of pre-aggregation pays off. Finally, we emphasize that the TVA's speedup over Waldo is significantly higher if we include initialization time. While TVA's parties simply receive secret shares, Waldo servers need to construct the tree index. In our experiments, this initialization step took over $1000s$. If this overhead is considered, Waldo's latency cannot match that of TVA, even for coarse windows.

## 6.2 Performance on real-world applications

**Evaluation setup.** In the remainder of the experimental evaluation we use two cloud deployments with smaller machines than before: (i) LAN uses one EC2 `r5.8xlarge` instance per party in the `us-east-2` region of AWS, and (ii) WAN distributes parties across different geographical regions. For the semi-honest protocol, parties run in `us-east-2` (Ohio), `us-east-1` (Virginia), and `us-west-1` (California). For the malicious protocol, we use the same three regions plus `us-west-2` for the fourth party (Oregon). All VMs have 16 physical cores, 256GB RAM, and run Ubuntu `20.04.4`, `C++14`, `g++ 9.4.0`, and `MPICH 3.3.2`. The RTT between west and east regions is 40-50ms and the bandwidth is limited at 5 Gbps. In all experiments we use the default configuration with 32-bit attributes and 64-bit timestamps.

| Number of data sources (WAN) | | | |
|---|---|---|---|
| | ***Energy*** | ***mHealth*** | ***Scheduling*** |
| **Semi-honest** | 17400 | 174700 | 52400 |
| **Malicious** | 8700 | 87300 | 26210 |

Table 3: Performance evaluation of TVA in the online setting. The table shows the number of data owners TVA can sustain without violating the corresponding application's time constraints.

Reported measurements are averaged over at least three runs and plotted in log-scale, unless otherwise specified. In all experiments of this section, timestamps require up to 22 bits (which are sufficient to encode $10s$ measurements over the course of one year). That is, TVA's security parameter is $\sigma = 42$ or greater.

### 6.2.1 Application scenarios

In this set of experiments, we use three complex window queries that process time series from different application domains. We provide the queries written in TVA's API in Appendix H. We use these real-world applications to evaluate TVA's ability to support efficient time series queries in both *online* and *historical* analysis scenarios. In the first case, we demonstrate that TVA can comfortably compute query results under rigid time constraints. In the second scenario, we show that TVA can also support large-scale offline analytics on millions of input records.

**Monitoring energy consumption.** Privacy-preserving time series analysis can enable various smart grid applications, such as monitoring grid conditions to improve energy conservation and reduce peak demand [58, 86]. For this use case, we use a time series dataset that consists of energy consumption measurements across clients connected in a smart grid. The query computes the total energy consumption of the grid over tumbling windows.

**Mobile health analytics.** We developed the second use-case together with collaborators from our institution's medical school. The input time series corresponds to measurements collected by wearable devices in a cohort of patients with diabetes. The measurements include glucose values and insulin dose events, among other attributes, and are used by the investigators to assess the effects of insulin doses across patients in the cohort. The analysis requires counting the number of insulin doses during each patient's eating period, which is identified by a threshold in the glucose value. We express this query using a threshold session window operator.

**Job scheduling optimization.** The third application considers a cloud provider who wishes to optimize resource provisioning without compromising their clients' privacy. To this end, the provider needs to process telemetry time series data from multiple clients without gaining access to raw execution logs that may contain proprietary information. We use a time series that follows the schema of real-world monitoring traces collected from Google clusters [87]. The provider is interested in analyzing the length of job phases (sessions) assigned to the various machine types and use this information to decide when to transfer long-running client workloads to larger machines or co-locate client workloads with short job sessions. We implement this application by first applying a filter to retain `schedule` events and then using a keyed gap session window to identify job sessions per machine type. The respective query performs a final aggregation to compute the session length and returns the maximum session per machine type.

### 6.2.2 Online analysis scenario with time constraints

For a system to guarantee *online* operation, it needs to produce query results at a rate higher than that of data ingestion. For the energy use case, we consider sensors that generate measurements every $10s$ and a monitoring dashboard that needs to be refreshed every $5min$. These settings are more challenging than what typical smart grid applications require [58]. For the mobile health use case, we set the glucose measurement interval to $5min$ according to a recent medical study [43] and require computing the eating period in under one hour. Finally, for the scheduling application, we assume a Google cluster workload with an arrival rate of 10,000 tasks per minute, [87] and set the time constraint to $10min$, which is lower than the lifetime of $99\%$ of Microsoft Azure's VMs [26].

Table 3 shows the number of data sources that TVA can sustain without violating its online processing guarantees in the WAN setting. In all cases, the results demonstrate that TVA exceeds the requirements of real-world applications. With malicious security, TVA can update the energy grid dashboard in real time for over $8K$ sensors, it can support a

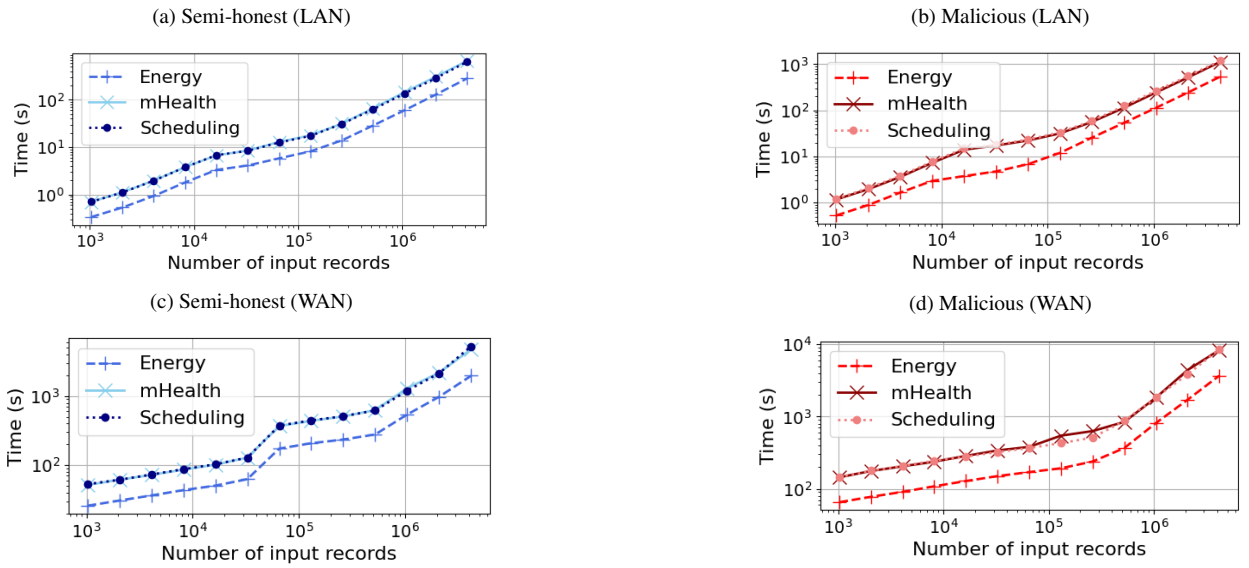


Figure 4: TVA query latency in real-world applications with out-of-order events, as the time series size increases from $2^{10}$ to $2^{22}$.

cohort of over $87K$ patients for glucose monitoring, and it can keep up with $2.6\times$ higher task arrival rate than that of a production cluster.

### 6.2.3 Historical analysis scenario

For the historical analytics scenario, we measure query latency as the time series size increases. Figure 4 shows the results. We emphasize that, both in this and the previous experiment, timestamps are not considered to be ordered and the reported times include sorting under MPC. TVA comfortably scales to millions of records and computes all queries within a few minutes. For $2^{20}$ records, the *Energy* query takes 1min in LAN and 9min in WAN, while the *mHealth* and *Scheduling* queries have similar performance and take $\sim$ 2.4min in LAN and 20min in WAN. The *mHealth* and *Scheduling* queries are both based on sessionization but the latter is more complex, including `filter` and `keyBy` operators. The overhead of these additional operators is negligible thanks to TVA's efficient composition protocols. The cost of using a malicious protocol is small, resulting in a $2\times$ slowdown.

# 7 Related work

**Privacy-preserving time series analysis.** Although there is a large amount of work in privacy-preserving time series analysis, none of the existing approaches supports secure windowing. Despite their particular differences with respect to threat models and security guarantees, the vast majority of efforts focus on global aggregations or snapshot queries with (optional) filters and aggregation [23, 29, 30, 52, 55, 58, 60, 80, 82–84, 92]. TimeCrypt [17] and Zeph [18] have built-in support for tumbling windows, but they reveal information to the untrusted server and can only operate on public timestamps. Zeph also supports limited (additive) per-group aggregation within time windows but requires data holders to pre-encode the grouping key domain (e.g., all possible zipcodes).

TVA extends the functionality of prior works without compromising security or performance. TVA introduces expressive recurring window operators (§4.2–4.4) and allows for efficient composition of window and snapshot operators with custom grouping, filters, and aggregation (§4.6). In addition, TVA hides queried time intervals, as in state-of-the-art approaches, but also supports irregular timestamps and out-of-order records. This is in contrast to many prior techniques that either assume ordered timestamps (e.g., [30]) or operate in the "synchronous" setting where the aggregator receives data every $k$ time units (e.g., [23, 80, 84]). Recently, Zheng et al. [96] proposed a technique to compute similarity queries on time series snapshots. We leave such operations as future work.

**FSS-based approaches.** Function secret sharing [13, 14] is a cryptographic technique that allows creating shares of a function as opposed to creating shares of a value in MPC. Approaches using FSS, including distributed point functions, incur lower communication compared to MPC-based techniques. Waldo [30], DuraSift [38], and Vizard [19] use FSS to securely evaluate range and equality predicates by splitting shares of the predicate to untrusted non-colluding parties. Splinter [90] uses FSS to evaluate private SQL queries on public data, whereas Dittmer et al. [31] employ FSS for efficient streaming PSI. Expressing TVA's complex window operations using FSS is still an open problem and an interesting direction for future work.

**ORAM-based approaches.** Oblivious RAM [46, 47] allows for hiding access patterns in arbitrary programs but its generality comes at a high cost. Distributed ORAM constructions by Bunn et al. [16] and Doerner et al. [32] achieve better performance by leveraging FSS. TVA hides access patterns via oblivious operators that we specifically design for time series analytics.

**Relational MPC frameworks.** Systems like Conclave [89], Senate [79], and Secrecy [69] support a broad class of relational operators, including joins, filters, and group-by with aggregation. These systems can be used to evaluate snapshot operators but they do not support recurring windows. One workaround, and only for tumbling windows, is to use multiple snapshot queries; however, this requires knowledge of the time domain and has substantial performance overheads, even for FSS-based approaches.

**Encrypted DBs.** Encrypted databases based on structural encryption [20, 42, 57, 62, 77, 78, 93] avoid the non-collusion assumption of MPC, but they reveal access and search pattern information to the database server that can be used in reconstruction attacks [50, 51, 64, 73, 75]. In addition, none of these systems supports window-based time series analytics.

**Privacy-preserving aggregate statistics.** Works in this class focus on common aggregations [33], custom analytics [15, 56], or general-purpose statistics [3, 25]. None of these works study recurring window aggregation on time series data.

# 8 Conclusion

We presented TVA, a multi-party computation system for efficient and expressive time series analytics with strong security guarantees. TVA employs new MPC protocols for secure window assignment and a modular design that enables arbitrary composition of temporal and time-agnostic operators. TVA's protocols are generic and operate correctly and efficiently on time series with unordered and irregular timestamps. Our evaluation on real-world applications demonstrates that TVA is suitable for both online and historical analysis.

# Acknowledgments

The authors thank the anonymous reviewers and shepherd for their valuable feedback. We also thank Dr. Nicole Spartano and Dr. Lisa Quintiliani for helping us develop the mobile health application, and Ian Saucy for his help with the WAN deployments.

# References

[1] Libsodium library. `https://libsodium.gitbook.io`. [Online; accessed February 2023].

[2] Message Passing Interface (MPI). `https://www.mcs.anl.gov/research/projects/mpi/standard.html`. [Online; accessed February 2023].

[3] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. In *SCN*, volume 13409 of *LNCS*, pages 516–539. Springer, 2022.

[4] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, et al. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8(12), 2015.

[5] Amazon AWS. AWS EC2 On Demand Pricing. https://aws.amazon.com/ec2/pricing/on-demand/. [Online; accessed February 2023].

[6] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *CCS*, pages 805–817. ACM, 2016.

[7] Toshinori Araki, Jun Furukawa, Kazuma Ohara, Benny Pinkas, Hanan Rosemarin, and Hikaru Tsuchida. Secure graph analysis at scale. In *CCS*, pages 610–629. ACM, 2021.

[8] Gilad Asharov, T-H. Hubert Chan, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Bucket oblivious sort: An extremely simple oblivious sort, 2021.

[9] Gilad Asharov, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Ariel Nof, Benny Pinkas, Katsumi Takahashi, and Junichi Tomida. Efficient secure three-party sorting with applications to data analysis and heavy hitters. In *CCS*, pages 125–138. ACM, 2022.

[10] Kenneth E. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computing Conference*, volume 32 of *AFIPS Conference Proceedings*, pages 307–314. Thomson Book Company, Washington D.C., 1968.

[11] Marina Blanton and Everaldo Aguiar. Private and oblivious set and multiset operations. In *AsiaCCS*, pages 40–41. ACM, 2012.

[12] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[13] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *EUROCRYPT*, volume 12697 of *LNCS*, pages 871–900. Springer, 2021.

[14] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In *TCC (1)*, volume 11891 of *LNCS*, pages 341–371. Springer, 2019.

[15] Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin. Private matching for compute. IACR Cryptology ePrint Archive, Report 2020/599, 2020.

[16] Paul Bunn, Jonathan Katz, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient 3-party distributed ORAM. In *SCN*, volume 12238 of *LNCS*, pages 215–232. Springer, 2020.

[17] Lukas Burkhalter, Anwar Hithnawi, Alexander Viand, Hossein Shafagh, and Sylvia Ratnasamy. TimeCrypt: Encrypted data stream processing at scale with cryptographic access control. In *NSDI*, pages 835–850. USENIX Association, 2020.

[18] Lukas Burkhalter, Nicolas Küchler, Alexander Viand, Hossein Shafagh, and Anwar Hithnawi. Zeph: Cryptographic enforcement of end-to-end data privacy. In *OSDI*, pages 387–404. USENIX Association, 2021.

[19] Chengjun Cai, Yichen Zang, Cong Wang, Xiaohua Jia, and Qian Wang. Vizard: A metadata-hiding data analytic system with end-to-end policy controls. In *CCS*, pages 441–454. ACM, 2022.

[20] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS*. The Internet Society, 2014.

[21] Octavian Catrina and Sebastiaan de Hoogh. Improved primitives for secure multiparty integer computation. In *SCN*, volume 6280 of *LNCS*, pages 182–199. Springer, 2010.

[22] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In *Financial Cryptography*, volume 6052 of *LNCS*, pages 35–50. Springer, 2010.

[23] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Privacy-preserving stream aggregation with fault tolerance. In *Financial Cryptography*, volume 7397 of *LNCS*, pages 200–214. Springer, 2012.

[24] Koji Chida, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Naoto Kiribuchi, and Benny Pinkas. An efficient secure three-party sorting protocol with an honest majority. IACR Cryptology ePrint Archive, Report 2019/695, 2019.

[25] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *NSDI*, pages 259–282. USENIX Association, 2017.

[26] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *SOSP*, pages 153–167. ACM, 2017.

[27] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. *Proc. Priv. Enhancing Technol.*, 2020(4):355–375, 2020.

[28] Anders PK Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four: Honest-majority four-party secure computation with malicious security. In *USENIX Security Symposium*, pages 2183–2200, 2021.

[29] George Danezis, Cédric Fournet, Markulf Kohlweiss, and Santiago Zanella Béguelin. Smart meter aggregation via secret-sharing. In *SEGS@CCS*, pages 75–80. ACM, 2013.

[30] Emma Dauterman, Mayank Rathee, Raluca Ada Popa, and Ion Stoica. Waldo: A private time-series database from function secret sharing. In *IEEE Symposium on Security and Privacy*, pages 2450–2468. IEEE, 2022.

[31] Samuel Dittmer, Yuval Ishai, Steve Lu, Rafail Ostrovsky, Mohamed Elsabagh, Nikolaos Kiourtis, Brian Schulte, and Angelos Stavrou. Streaming and unbalanced PSI from function secret sharing. In *SCN*, volume 13409 of *LNCS*, pages 564–587. Springer, 2022.

[32] Jack Doerner and Abhi Shelat. Scaling ORAM for secure computation. In *CCS*, pages 523–535. ACM, 2017.

[33] F. Emekci, D. Agrawal, A. E. Abbadi, and A. Gulbeden. Privacy preserving query processing using third parties. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 27–27, 2006.

[34] Daniel Escudero. An introduction to secret-sharing-based secure multiparty computation. IACR Cryptology ePrint Archive, Report 2022/062, 2022.

[35] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In *CRYPTO*, volume 12171 of *LNCS*, pages 823–852. Springer, 2020.

[36] Muhammad Faisal, Jerry Zhang, John Liagouris, Vasiliki Kalavri, and Mayank Varia. TVA: A multi-party computation system for secure and expressive time series analytics. In *USENIX Security Symposium*. USENIX Association, 2023.

[37] Muhammad Faisal, Jerry Zhang, John Liagouris, Vasiliki Kalavri, and Mayank Varia. TVA GitHub repository. `https://github.com/CASP-Systems-BU/tva`, 2023.

[38] Brett Hemenway Falk, Steve Lu, and Rafail Ostrovsky. DURASIFT: A robust, decentralized, encrypted database supporting private searches with complex policy controls. In *WPES@CCS*, pages 26–36. ACM, 2019.

[39] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. 3-party distributed ORAM from oblivious set membership. In *SCN*, volume 13409 of *LNCS*, pages 437–461. Springer, 2022.

[40] Liyue Fan and Li Xiong. Real-time aggregate monitoring with differential privacy. In *CIKM*, pages 2169–2173. ACM, 2012.

[41] Ferdinando Fioretto and Pascal Van Hentenryck. Optstream: Releasing time series privately. *J. Artif. Intell. Res.*, 65:423–456, 2019.

[42] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K. Cunningham. Sok: Cryptographically protected database search. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 172–191. IEEE Computer Society, 2017.

[43] Alfonso Galderisi, Luca Zammataro, Eleonora Losiouk, Giordano Lanzola, et al. Continuous glucose monitoring linked to an artificial intelligence risk index: early footprints of intraventricular hemorrhage in preterm neonates. *Diabetes technology & therapeutics*, 21(3):146–153, 2019.

[44] Peng Gao, Xusheng Xiao, Ding Li, Zhichun Li, Kangkook Jee, Zhenyu Wu, Chung Hwan Kim, Sanjeev R. Kulkarni, and Prateek Mittal. SAQL: A stream-based query system for real-time abnormal system behavior detection. In *USENIX Security Symposium*, pages 639–656. USENIX Association, 2018.

[45] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178. ACM, 2009.

[46] Oded Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *STOC*, pages 182–194. ACM, 1987.

[47] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, May 1996.

[48] Michael T. Goodrich. Data-oblivious external-memory algorithms for the compaction, selection, and sorting of outsourced data. In *SPAA*, pages 379–388. ACM, 2011.

[49] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. MPC-friendly symmetric key primitives. In *CCS*, pages 430–443. ACM, 2016.

[50] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In *CCS*, pages 315–331. ACM, 2018.

[51] Paul Grubbs, Richard McPherson, Muhammad Naveed, Thomas Ristenpart, and Vitaly Shmatikov. Breaking web applications built on top of encrypted data. CCS '16, page 1353–1364, 2016.

[52] Subir Halder and Mauro Conti. CrypSH: A novel IoT data protection scheme based on BGN cryptosystem. *IEEE Transactions on Cloud Computing*, 2021.

[53] Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Practically efficient multi-party sorting protocols from comparison sort algorithms. In *ICISC*, volume 7839 of *LNCS*, pages 202–216. Springer, 2012.

[54] D. Harris. A taxonomy of parallel prefix networks. In *The 37th Asilomar Conference on Signals, Systems & Computers*, volume 2, pages 2213–2217, 2003.

[55] Matús Harvan, Samuel Kimoto, Thomas Locher, Yvonne-Anne Pignolet, and Johannes Schneider. Processing encrypted and compressed time series data. In *ICDCS*, pages 1053–1062. IEEE Computer Society, 2017.

[56] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Mariana Raykova, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. On deploying secure computing commercially: Private intersection-sum protocols and their business applications. IACR Cryptology ePrint Archive, Report 2019/723, 2019.

[57] Yuval Ishai, Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. Private large-scale databases with distributed searchable symmetric encryption. In *Proceedings of the RSA Conference on Topics in Cryptology - CT-RSA 2016 - Volume 9610*, page 90–107, Berlin, Heidelberg, 2016. Springer-Verlag.

[58] Marek Jawurek, Florian Kerschbaum, and George Danezis. Privacy technologies for smart grids - a survey of options. Technical Report MSR-TR-2012-119, November 2012.

[59] Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. Time series management systems: A survey. *IEEE Trans. Knowl. Data Eng.*, 29(11):2581–2600, 2017.

[60] Marc Joye and Benoît Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In *Financial Cryptography*, volume 7859 of *LNCS*, pages 111–125. Springer, 2013.

[61] Kristján Valur Jónsson, Gunnar Kreitz, and Misbah Uddin. Secure multi-party sorting and applications. In *ACNS*, June 2011.

[62] Seny Kamara and Tarik Moataz. SQL on structurally-encrypted databases. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 149–180, Cham, 2018. Springer International Publishing.

[63] Manos Katsomallos, Katerina Tzompanaki, and Dimitris Kotzinos. Landmark privacy: Configurable differential privacy protection for time series. In *CODASPY*, pages 179–190. ACM, 2022.

[64] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1329–1340, 2016.

[65] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *CCS*, pages 1575–1590. ACM, 2020.

[66] Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Peter Scholl, Eduardo Soria-Vazquez, and Srinivas Vivek. Faster secure multi-party computation of AES and DES using lookup tables. In *ACNS*, volume 10355 of *LNCS*, pages 229–249. Springer, 2017.

[67] Brian Knott, Shobha Venkataraman, Awni Y. Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. CrypTen: Secure multi-party computation meets machine learning. In *NeurIPS*, pages 4961–4973, 2021.

[68] Simeon Krastnikov, Florian Kerschbaum, and Douglas Stebila. Efficient oblivious database joins. *Proc. VLDB Endow.*, 13(11):2132–2145, 2020.

[69] John Liagouris, Vasiliki Kalavri, Muhammad Faisal, and Mayank Varia. Secrecy: Secure collaborative analytics in untrusted clouds. In *NSDI*, pages 1031–1056. USENIX Association, 2023.

[70] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017.

[71] Yehuda Lindell. Secure multiparty computation. *Commun. ACM*, 64(1):86–96, December 2020.

[72] Yasuko Matsubara, Yasushi Sakurai, Willem G. van Panhuis, and Christos Faloutsos. FUNNEL: automatic mining of spatially coevolving epidemics. In *KDD*, pages 105–114. ACM, 2014.

[73] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. Oblix: An efficient oblivious search index. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 279–296, 2018.

[74] Payman Mohassel and Peter Rindal. ABY$^3$: A mixed protocol framework for machine learning. In *CCS*, pages 35–52. ACM, 2018.

[75] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In *ACM Conference on Computer and Communications Security*, pages 644–655. ACM, 2015.

[76] Spiros Papadimitriou and Philip S. Yu. Optimal multi-scale patterns in time series streams. In *SIGMOD Conference*, pages 647–658. ACM, 2006.

[77] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind seer: A scalable private dbms. In *2014 IEEE Symposium on Security and Privacy*, pages 359–374, 2014.

[78] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. Arx: An encrypted database using semantically secure encryption. *Proc. VLDB Endow.*, 12(11):1664–1678, July 2019.

[79] Rishabh Poddar, Sukrit Kalra, Avishay Yanai, Ryan Deng, Raluca Ada Popa, and Joseph M. Hellerstein. Senate: A maliciously-secure MPC platform for collaborative analytics. In *USENIX Security Symposium*, pages 2129–2146. USENIX Association, 2021.

[80] Vibhor Rastogi and Suman Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD Conference*, pages 735–746. ACM, 2010.

[81] Yasushi Sakurai, Yasuko Matsubara, and Christos Faloutsos. Mining and forecasting of big time-series data. In *SIGMOD Conference*, pages 919–922. ACM, 2015.

[82] Hossein Shafagh, Anwar Hithnawi, Lukas Burkhalter, Pascal Fischli, and Simon Duquennoy. Secure sharing of partially homomorphic encrypted iot data. In *SenSys*, pages 29:1–29:14. ACM, 2017.

[83] Hossein Shafagh, Anwar Hithnawi, Andreas Droescher, Simon Duquennoy, and Wen Hu. Talos: Encrypted query processing for the internet of things. In *SenSys*, pages 197–210. ACM, 2015.

[84] Elaine Shi, T.-H. Hubert Chan, Eleanor Gilbert Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *NDSS*. The Internet Society, 2011.

[85] Jonas Traub, Philipp M Grulich, Alejandro Rodríguez Cuéllar, Sebastian Breß, Asterios Katsifodimos, Tilmann Rabl, and Volker Markl. Efficient window aggregation with general stream slicing. In *EDBT*, volume 19, pages 97–108, 2019.

[86] Maria Lorena Tuballa and Michael Lochinvar Abundo. A review of the development of smart grid technologies. *Renewable and Sustainable Energy Reviews*, 59:710–725, 2016.

[87] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *EuroSys*, pages 18:1–18:17. ACM, 2015.

[88] Thijs Veugen and Mark Abspoel. Secure integer division with a private divisor. *Proc. Priv. Enhancing Technol.*, 2021(4):339–349, 2021.

[89] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. Conclave: secure multi-party computation on big data. In *EuroSys*, pages 3:1–3:18. ACM, 2019.

[90] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *NSDI*, pages 299–313. USENIX Association, 2017.

[91] Hao Wang and Zhengquan Xu. CTS-DP: publishing correlated time-series data via differential privacy. *Knowl. Based Syst.*, 122:167–179, 2017.

[92] Wanli Xue, Chengwen Luo, Guohao Lan, Rajib Kumar Rana, Wen Hu, and Aruna Seneviratne. Kryptein: a compressive-sensing-based encryption scheme for the internet of things. In *IPSN*, pages 169–180. ACM, 2017.

[93] Zheguang Zhao, Seny Kamara, Tarik Moataz, and Zdonik Stan. Encrypted databases: From theory to systems. In *Proceedings of the 11th Annual Conference on Innovative Data Systems Research*, 2021.

[94] Wenting Zheng, Ryan Deng, Weikeng Chen, Raluca Ada Popa, Aurojit Panda, and Ion Stoica. Cerebro: A platform for multi-party cryptographic collaborative learning. In *USENIX Security Symposium*, pages 2723–2740. USENIX Association, 2021.

[95] Wenting Zheng, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Helen: Maliciously secure coopetitive learning for linear models. In *IEEE Symposium on Security and Privacy*, pages 724–738. IEEE, 2019.

[96] Yandong Zheng, Rongxing Lu, Yunguo Guan, Jun Shao, and Hui Zhu. Efficient and privacy-preserving similarity range query over encrypted time series data. *IEEE Transactions on Dependable and Secure Computing*, 19(4):2501–2516, 2022.

[97] Yunyue Zhu and Dennis E. Shasha. StatStream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, pages 358–369. Morgan Kaufmann, 2002.

**Functionality $\mathcal{F}_{\text{abb}}$**

**input:** Invoked upon receiving $(\text{input}, P_i, \mathsf{type}, \mathsf{id}, \ell_x, x)$ from party $P_i$ and $(\text{input}, P_i, \mathsf{type}, \mathsf{id}, \ell_x)$ from all other parties. Verifies that $\mathsf{id}$ is a fresh identifier and $\mathsf{type} \in \{\mathsf{arithmetic}, \mathsf{boolean}\}$. Stores $(\mathsf{type}, \mathsf{id}, \ell_x, x)$ into memory.

**shared input:** This is an optional method. It is invoked upon receiving $(\text{INP}, P_i, P_j, \mathsf{type}, \mathsf{id}, \ell_x, x)$ from party $P_i$, $(\text{INP}, P_i, P_j, \mathsf{type}, \mathsf{id}, \ell_x, x')$ from party $P_j$, and $(\text{input}, P_i, P_j, \mathsf{type}, \mathsf{id}, \ell_x)$ from all other parties. Verifies that $x = x'$, $\mathsf{id}$ is a fresh identifier, and $\mathsf{type} \in \{\mathsf{arithmetic}, \mathsf{boolean}\}$. Stores $(\mathsf{type}, \mathsf{id}, \ell_x, x)$ into memory.

**add:** Invoked upon receiving $(\text{add}, \mathsf{type}, \mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_{\mathsf{new}})$ from all parties. Retrieves from memory the values $x_1, x_2$ corresponding to identifiers $\mathsf{id}_1, \mathsf{id}_2$. Verifies that both of them are of the stated $\mathsf{type}$ and have the same length $\ell_x$, and that $\mathsf{id}_{\mathsf{new}}$ is a fresh identifier. If $\mathsf{type} = \mathsf{arithmetic}$, then set $x = x_1 + x_2 \bmod 2^{\ell_x}$. If $\mathsf{type} = \mathsf{boolean}$, then set $x = x_1 \oplus x_2$. Stores $(\mathsf{type}, \mathsf{id}_{\mathsf{new}}, \ell_x, x)$ into memory.

**mult:** Invoked upon receiving $(\text{mult}, \mathsf{type}, \mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_{\mathsf{new}})$ from all parties. Retrieves from memory the values $x_1, x_2$ corresponding to identifiers $\mathsf{id}_1, \mathsf{id}_2$. Verifies that both of them are of the stated $\mathsf{type}$ and have the same length $\ell_x$, and that $\mathsf{id}_{\mathsf{new}}$ is a fresh identifier. If $\mathsf{type} = \mathsf{arithmetic}$, then set $x = x_1 \cdot x_2 \bmod 2^{\ell_x}$. If $\mathsf{type} = \mathsf{boolean}$, then set $x = x_1 \wedge x_2$. Stores $(\mathsf{type}, \mathsf{id}_{\mathsf{new}}, \ell_x, x)$ into memory.

**A2B:** Invoked upon receiving $(\text{A2B}, \mathsf{id}, \mathsf{id}_{\mathsf{new}})$ from all parties. Retrieves from memory the tuple $(\mathsf{type}, \mathsf{id}, \ell_x, x)$ with identifier $\mathsf{id}$. Verifies that $\mathsf{type} = \mathsf{arithmetic}$ and $\mathsf{id}_{\mathsf{new}}$ is a fresh identifier. Stores $(\mathsf{boolean}, \mathsf{id}_{\mathsf{new}}, \ell_x, x)$ into memory.

**B2A:** Invoked upon receiving $(\text{B2A}, \mathsf{id}, \mathsf{id}_{\mathsf{new}})$ from all parties. Retrieves from memory the tuple $(\mathsf{type}, \mathsf{id}, \ell_x, x)$ corresponding to identifier $\mathsf{id}$. Verifies that $\mathsf{type} = \mathsf{boolean}$ and $\mathsf{id}_{\mathsf{new}}$ is a fresh identifier. Stores $(\mathsf{arithmetic}, \mathsf{id}_{\mathsf{new}}, \ell_x, x)$ into memory.

**b2A:** Invoked upon receiving $(\text{B2A}, \mathsf{id}, \mathsf{id}_{\mathsf{new}}, \tilde{\ell}_x)$ from all parties. Retrieves from memory the tuple $(\mathsf{type}, \mathsf{id}, \ell_x, x)$ corresponding to identifier $\mathsf{id}$. Verifies that $\mathsf{type} = \mathsf{boolean}$, $\mathsf{id}_{\mathsf{new}}$ is a fresh identifier, and $\ell_x = 1$. Stores $(\mathsf{arithmetic}, \mathsf{id}_{\mathsf{new}}, \tilde{\ell}_x, x)$ into memory.

**output:** Invoked upon receiving $(\text{output}, P_i, \mathsf{type}, \mathsf{id})$ from all parties. Retrieves from memory the value $x$ corresponding to identifier $\mathsf{id}$, and sends it to party $P_i$.

Figure 5: Arithmetic black-box functionality $\mathcal{F}_{\text{abb}}$ that supports mixed-mode operations and conversions. Based upon Escudero et al. [35, Figure 1], with a few modifications described in Appendix A.

# A Arithmetic black-box model

In this section and in Figure 5, we provide a rigorous specification of the arithmetic black-box (ABB) functionality used in this work. The works of Araki et al. [6], Dalskov et al. [28], and Mohassel and Rindal [74] provide instantiations for all of these functionalities, and prove simulation-based security.

The goal of the ABB model is to consider a reactive (i.e., stateful) functionality that corresponds to an ideal specification of the MPC operations that we require. This specification provides two benefits. First, we can abstract away the underlying implementations of these MPC operations when specifying TVA's protocols (as shown formally in Protocols 1 and 2, and implicitly in all other protocols). Second, we can write generic proofs (in Appendices C and E) that all of the protocols within TVA are secure when provided with *any* instantiation of the ABB functionality. Several prior works use the ABB model to prove the security of higher-level protocols that make black-box use of MPC primitive operations (e.g., [34, 35, 39, 49, 66, 88]).

In this work, we require a functionality that supports mixed-mode operations: both arithmetic and boolean computation as well as conversions between them. For this reason, our starting point is the functionality provided by Escudero et al. [35, Figure 1]. Our functionality $\mathcal{F}_{\text{abb}}$ in Figure 5 is similar to Escudero et al. [35, Figure 1]. In particular, we emphasize that the "add" and "mult" functionalities in Figure 5 support both arithmetic and boolean operations—that is, they also support the operations of boolean XOR and boolean AND. Additionally, we follow the convention that the method name itself is also the first input to the functionality; as a consequence, we use the notations $\mathcal{F}_{\text{abb}}.\text{input}(-)$

and $\mathcal{F}_{\text{abb}}(\text{input}, -)$ interchangeably.

We do make a few modifications to our functionality $\mathcal{F}_{\text{abb}}$ as compared to the ABB model of Escudero et al. First, our functionality considers a single addition operation at a time rather than providing support for more complicated linear combinations directly; this is purely a modeling decision and has no impact on the capabilities of the functionality. Second, we support addition and multiplication over arithmetic rings of different sizes; support for multiple moduli provides a small efficiency benefit in Protocols 1-2 but is not required and may be omitted. Third, we add support for the shared input (INP) and bit-to-arithmetic (b2A) functionalities. Fourth, we allow for the output to be revealed only to a single entity (e.g., the data analyst), rather than all protocol participants. Fifth, we add an arithmetic to (multi-bit) boolean operation A2B, which is instantiated in the 3-party variant of TVA using a parallel prefix adder as described in Mohassel and Rindal [74, §5.3], and we extend their technique to the 4-party setting in Appendix B.

It is straightforward to see that the works of Araki et al. [6] and Dalskov et al. [28] provide most of the protocols that collectively instantiate our $\mathcal{F}_{\text{abb}}$. Concretely, they provide specific ideal functionalities for each method within Figure 5 (e.g., multiplication is described in [6, Functionality 3.3] and [28, Protocol 4]), and then provide simulation-secure instantiations of each method. Furthermore, revealing the output to one party does not reveal any information about other secrets, due to the use of fresh randomness to re-randomize shares after every interactive operation. There are two caveats, however. First, only Dalskov et al. [28] provides an instantiation of the shared input method INP. As a result, we only use INP within the 4-party variant of TVA (e.g., we use it in Protocol 2 but not Protocol 1). Second, we adopt the arithmetic-to-boolean conversion protocols from Mohassel and Rindal [74, §5.3] (in the 3-party setting) and Appendix B (in the 4-party setting).

# B 4-party arithmetic to boolean conversion

In Protocol 6, we provide a new mixed-mode protocol for arithmetic to boolean conversion. Its main advantage is to reduce the number of communication rounds required: our protocol uses 1 boolean adder rather than 3 of them. Additionally, our protocol is amenable to use within our 4-party secure division protocol, in which case the "shares merging" step below can be removed since the `INP` sharing is already performed within Protocol 2.

**Protocol 6:** A2B CONVERSION (Malicious 4PC)

**Input :** An arithmetic secret-shared integer $\langle\langle x \rangle\rangle_\ell$
**Output:** A boolean secret-shared integer $[\![x]\!]_\ell$

`//Shares Merging`

1 Parties 1, 4 locally compute $x'_0 = x_1 + x_2$
2 Party 2, 3 locally compute $x'_1 = x_3 + x_4$

`//Shares redistribution`

3 Let $[\![a]\!]_\ell = \texttt{INP}_B(x'_0, 1, 4)$
4 Let $[\![b]\!]_\ell = \texttt{INP}_B(x'_1, 2, 3)$
5 Output $[\![x]\!]_\ell = [\![a]\!]_\ell + [\![b]\!]_\ell$

# C Security analysis of TVA division protocols

In this section, we provide a security analysis of Protocols 1 and 2 in order to prove the following theorem.

**Theorem 1** *Let $x \in [0, 2^{\ell_x})$ and $c \in [1, 2^{\ell_c - 1})$ with $\ell_c \leq \ell_x < \ell$. Then, Protocols 1 and 2 securely compute a sharing of the exact integer division $z = \lfloor x/c \rfloor$ with probability $1 - \frac{1}{2^\sigma}$, where $\sigma = \ell - \ell_x$. Protocol 1 achieves semi-honest security, and Protocol 2 achieves malicious security up to abort.*

For ease of presentation, we split the proof of this theorem into three lemmas. We begin with a lemma showing that line 1 of Protocols 1 and 2 constructs a value $y$ that is close to the correct quotient, up to a rounding error of either 0 or 1. Line 1 is a slight modification to the division-and-truncation protocol of Mohassel and Rindal [74], and our

proof is similar to Mohassel and Rindal [74, Thm. 1]. We provide a full proof below to show how their technique adapts to our setting, in which we use a non-replicated 2-of-2 sharing $\langle y \rangle_\ell$.

**Lemma 1** *Let $c \geq 1$ and $0 \leq x < 2^{\ell_x}$, and let $z = \left\lfloor \frac{x}{c} \right\rfloor$ be the exact integer division of $x$ by $c$ with truncation. Then, line 1 of Protocols 1 and 2 computes $\langle y \rangle_\ell$ such that $y = z$ or $y = z + 1$ with probability $1 - 2^{\ell_x - \ell}$.*

**Proof** *We begin by defining a non-replicated 2-of-2 sharing of $x$ denoted by $x'_1$ and $x'_2$. First set $x'_1 = x_1 + x_2 \bmod 2^\ell$. Then, for the analysis of Protocol 1 we set $x'_2 = x_3$, and in the analysis of Protocol 2 we set $x'_2 = x_3 + x_4 \bmod 2^\ell$. (The analyses of the two protocols are identical from here onward.)*

*In other words, $x = x'_1 + x'_2 \bmod 2^\ell$, where the original secret sharing guarantees that each of $x'_1$ and $x'_2$ are distributed uniformly at random within $[0, 2^\ell)$ subject to the fact that they sum to $x \bmod 2^\ell$. Since $x$ itself is less than $2^{\ell_x}$, with overwhelming probability (at least $1 - 2^{\ell_x - \ell}$) it is the case that $x'_1 > x$ and $x'_2 > x$; in fact, each inequality implies the other.*

*As a result, we can write an integer equation (without modular arithmetic) that $x = x'_1 + x'_2 - 2^\ell$. We can rewrite this equation as $x = x'_1 - x''_2$, where we set $x''_2 = 2^\ell - x'_2$ and observe that it is also in the range $[0, 2^\ell)$. Hence,*

$$z = \lfloor x/c \rfloor = \lfloor (x'_1 - x''_2)/c \rfloor \approx \lfloor x'_1/c \rfloor - \lfloor x''_2/c \rfloor. \tag{5}$$

*We set $y$ equal to the right hand side of Eq.* (5)*, and observe that it is close to $z$. We claim that the approximation in the final step comes from the fact that splitting the truncation into the difference of two individually-truncated values can introduce an off-by-one error, namely in the case that the fractional part of $x''_2/c$ is greater than the fractional part of $x'_1/c$. This claim is easiest to see with an example: given the lengths $\ell_x = \ell_c = 4$ and $\ell = \ell_x + \sigma$, consider what happens when $x = 2$ and $c = 5$, with $x'_1 = 6$ and $x''_2 = 4$. In this case, we find that $z = \lfloor (x'_1 - x''_2)/c \rfloor = \lfloor \frac{6}{5} - \frac{4}{5} \rfloor = 0$ as expected, but $y = \lfloor x'_1/c \rfloor - \lfloor x''_2/c \rfloor = \lfloor \frac{6}{5} \rfloor - \lfloor \frac{4}{5} \rfloor = 1 - 0 = 1$. The issue here is that the fractional part of $\frac{4}{5}$ is greater than the fractional part of $\frac{6}{5}$, and therefore is large enough to subtract one from the integer part of the result when it is applied before the floor operation is performed. This idea generalizes to any possible sharing: by computing the integer quotient and remainder, we can write the shares as $x'_1 = q_1 c + r_1$ and $x''_2 = q_2 c + r_2$ for integers $q_1, q_2, r_1, r_2$ with $r_1, r_2 \in [0, c)$. The right-hand side of Eq.* (5) *is $q_1 + q_2$, but the left-hand side equals $q_1 + q_2 + \lfloor \frac{r_1 - r_2}{c} \rfloor$, where $r_1 - r_2 \in (-c, c)$. So their difference (and the cause of the approximation error) is the floor function that equals either $0$ or $-1$ since $c$ is positive.*

*We also rely upon the fact that $c \geq 1$ for Eq. 5 to hold over a modular arithmetic space. Dividing by less than 1 could result in an overflow error, but as long as $|c| \geq 1$ then we know that $x'_1/c$ requires at most $\ell$ bits to represent because the same is true of $x'_1$. Also, dividing by a negative number would flip the sign of all equations, including the direction of the off-by-one error.*

*Finally, observe that*

$$y_1 = \lfloor x'_1/c \rfloor \text{ and } y_2 = -\lfloor x''_2/c \rfloor \bmod 2^\ell = 2^\ell - \lfloor x''_2/c \rfloor$$

*in order to produce a secret sharing $y = y_1 + y_2 \bmod 2^\ell$ such that $y_2$ is in the usual range $[0, 2^\ell)$. Hence, we achieve the desired result as long as $x'_1$ and $x'_2$ are both greater than $x$, which occurs with probability at least $1 - 2^{\ell_x - \ell}$.*

With this lemma, we are now prepared to prove Theorem 1. The rounding error identified in Lemma 1 can be corrected within a secure computation. The main idea is to determine which of the fractions $x'_1/c$ or $x''_2/c$ is larger, which is exactly what the remaining lines of Protocols 1 and 2 do. To show formally that these are secure computation protocols, we must show that each protocol satisfies correctness and privacy as described in Section 3. Therefore, Theorem 1 follows immediately from the following two lemmas.

**Lemma 2 (Correctness)** *The output of Protocols 1 and 2 is equal to $z = \lfloor x/c \rfloor$ with probability at least $1 - 2^{\ell_x - \ell}$.*

**Lemma 3 (Privacy)** *Protocols 1 and 2 achieve privacy against a semi-honest and malicious adversary, respectively.*

We provide proofs below for these two lemmas, which collectively suffice to prove Theorem 1 as well.

**Proof (Lemma 2)** *To show correctness, recall from Lemma 1 that $y$ is nearly equal to $z$, up to a possible off-by-one error. More precisely, recall that Eq.* (5) *is not exact in the case that $x_2''/c$ has a larger fractional part than $x_1'/c$. The remaining lines of Protocols 1 and 2 compute this off-by-one error and add it to $y$ in order to recover $z$ exactly. That said: if $x_1' < x$ or $x_2' < x$, then we still run into a catastrophic accuracy error, for the same reason as in Lemma 1.*

*In more detail, line 2 computes $s_1$ and $s_2$ as the fractional parts of $x_1'/c$ and $x_2''/c$, respectively. Then line 4 determines whether $s_1 < s_2$ as follows: the difference $s = s_1 - s_2$ is negative if and only if $s_1 < s_2$, and through mixed-mode conversions the protocol isolates this most significant bit of $s$ and sets it equal to $b$. One can think of $b$ akin to a 'sign' bit, since it is equal to 1 only if $s_2 > s_1$, which induces a wrap-around mod $2^{\ell_c}$. Observe also that in line 3, the shares $s_1$ and $s_2$ are placed in the ring mod $2^{\ell_c}$, where $\ell_c$ is specifically chosen so that $s_1$ and $s_2$ contain one more bit than $c$ itself. This provides the extra space to hold the 'sign' bit.*

**Proof (Lemma 3)** *At a high level, in Protocols 1 and 2, privacy is obtained through the black-box use of the underlying building blocks of the mixed-mode MPC protocol in the arithmetic black-box model $\mathcal{F}_{abb}$. Lines 1-2 are local computations, line 3 uses $\mathcal{F}_{abb}$.input to distribute new secret shares and inherits privacy from the underlying secret-sharing protocol, lines 3 and 5 inherit security from the underlying addition protocol $\mathcal{F}_{abb}$.add, and line 4 inherits security from the underlying conversion protocols $\mathcal{F}_{abb}$.A2B and $\mathcal{F}_{abb}$.b2A. Moreover, the protocol never opens any secret, and the control flow is data-independent.*

*As a result, the privacy of Protocol 1 follows directly from the privacy of the underlying MPC protocol of Araki et al. [6]. To show this claim formally, we must design a simulator $\mathcal{S}$ that produces a transcript of the protocol execution that is indistinguishable from the flow of Protocol 1, given only the inputs and outputs of the protocol participants that the adversary has (statically) corrupted. If the corrupted parties know all of the shares of either the input $x$ and/or the output $z$ (say, if the adversary corrupts the data owners and/or the data analyst), then simulation is trivial since division by a public divisor $c$ is an invertible function. Otherwise, the simulator $\mathcal{S}$ can execute Protocol 1 on a random input $x$, and the the distinguisher won't be able to tell the difference due to the perfect privacy of all primitives in $\mathcal{F}_{abb}$; in this case even the distinguisher's knowledge of the true output does not help, since the view of Protocol 1 is independent of it.*

*The proof of security in the 4-party case is similar, with the additional property that Protocol 2 achieves malicious security through redundancy: two parties perform each local computation in lines 1-2, and in line 3 we use the Shared Input protocol $\mathcal{F}_{abb}$.INP of Dalskov et al. [28] so that the honest parties can detect any malicious party's attempt to share an incorrect value. Finally, in lines 3-5 we rely on the malicious-secure instantiations of the functionalities $\mathcal{F}_{abb}$.add, $\mathcal{F}_{abb}$.A2B, and $\mathcal{F}_{abb}$.b2A instantiated by Dalskov et al. [28]. Hence, a malicious party is detected if it deviates from the protocol in any of the lines of the protocol. Therefore, Protocol 2 provides malicious security up to abort, where the corresponding simulator $\mathcal{S}$ can use the shared input functionality $\mathcal{F}_{abb}$.INP to extract the inputs provided by any adversarial data owners and then operates similarly as in the semi-honest setting.*

In Appendix D, we compare this division protocol with prior work. Also to improve efficiency, in Appendix B we contribute a simple arithmetic-to-boolean sharing protocol in the 4-party setting that is compatible with our division operator; it only requires a single boolean adder rather than 3 of them.

# D Comparison with other division protocols

In this section, we compare our Protocols 1-2 for division-and-truncation with some of the most related prior works. Secure division protocols have been a topic of study within MPC for more than a decade, and this section prioritizes comparison with recent, related protocols over a comprehensive review of all prior works. We distinguish between comparisons to protocols with and without rounding error.

First, there are several prior works that provide efficient protocols for integer or fixed-point division with probabilistic truncation, meaning that they perform randomized rounding of the result with a possible error of $\pm\delta$ where $\delta$ ranges from 1 to 3 depending on the protocol (and in some cases, with this error being biased toward the "right" answer). Catrina and Saxena [22] and Catrina and de Hoogh [21] introduced this technique for computation in a field modulo a prime, then Mohassel and Rindal [74] and Dalskov et al. [27] extend the technique to support rings modulo $2^\ell$ (where multiplication by the inverse is not always possible), and Escudero et al. [35] provide a method that uses mixed-mode MPC to perform efficient division by powers of two.

TVA operates in the ring modulo $2^\ell$ (with a small efficiency improvement if arithmetic in the smaller ring mod $2^{\ell_c+1}$ is also supported) and uses mixed-mode operations. In comparison to the protocols above, TVA makes a small improvement by using non-replicated 2-of-2 secret sharing within line 1 of Protocols 1-2 to ensure that the error is at most one bit (i.e., 0 or 1), before switching back to replicated secret sharing with 1 round of communication. Additionally, the remaining lines of our protocols use even more communication and computation in order to adjust the rounding error. We also remark that we have only presented Protocols 1-2 in the scenario where the dividend $x$ is an integer because it suffices for our application, but extending it to work with fixed-point arithmetic is straightforward using the same ideas as in prior work.

Second, there exist protocols that support division-and-truncation without rounding error. The most directly related work to ours is the recent paper of Veugen and Abspoel [88]. Their full construction supports secure integer divisions with a *private* divisor $c$, which TVA does not support. When comparing their protocol to ours in the setting of a public divisor, our protocol has improved storage and computation efficiency: for a given statistical security parameter $\sigma$ our construction requires a field of size $\ell = \ell_x + \sigma$ whereas their construction requires $\ell = \ell_x + 2(\sigma + \ell_c)$, and their functionality requires more correlated randomness in preprocessing in order to perform $O(\ell)$ multiplications per online division operation. Additionally, there are a few works [35, 65] that perform exact division in the restricted setting where the public divisor $c$ is a power of two, in which case they can use right-shifting of individual shares to speed up the computation. Our computational cost is similar to these works, with the added benefit that TVA supports public division by any divisor $c$.

# E Security analysis of TVA gap- and threshold-based sessionization protocols

In this section, we show that our oblivious sessionization protocol in Protocol 3 is a secure computation of gap- or threshold-based session windows. Specifically, we prove the following theorem.

**Theorem 2** *Protocol 3 securely identifies all session windows in a time series $S$ according to Definition 2 (for gap-based session windows) and Definition 3 (for threshold-based session windows), when initiated with the appropriate* SESSIONSTART.

For ease of presentation, we split this security proof into four claims that collectively suffice to prove Theorem 2: termination, completeness, correctness, and privacy.

**Lemma 4 (Termination)** *Protocol 3 will terminate.*

**Proof** *This is straightforward to see by inspection of Protocol 3 and its subroutines in Protocols 4-5 that initialize the gap- and threshold-based session window schemes. The outer* `while` *loop in Protocol 3 starts at $d = 1$ and doubles after every iteration, meaning that it will reach the stopping criterion of $d > |S|/2$ in $\log(|S|)$ steps. Additionally, the inner* `for` *loop in Protocol 3 and the* `for` *loops within each* SESSIONSTART *protocol both take a fixed number of steps $|S|$, all of which can be run in parallel. Ergo, the algorithm clearly terminates.*

**Lemma 5 (Completeness)** *After Protocol 3 is executed, every record will be assigned a window. That is, every record that is initially assigned an invalid window id $\varepsilon = -1$ will eventually be modified to be a valid window id.*

**Proof** *We prove this claim by induction on the number of iterations of the outer while loop. Specifically, we claim that for any choice of the outer-loop variable $d$, it is always the case that the first $d$ elements have correct window ids before the loop begins and the first $2d$ elements have the correct window ids after this iteration of the loop ends.*

*The base case of this induction is easy to verify. At the start of the protocol (and before the first iteration of the loop), $d$ is set to $1$ and the* SESSIONSTART *protocol is executed. It is easy to verify by inspection that for both gap- and threshold-based* SESSIONSTART *protocols (shown in Protocols 4-5), the first record of the dataset is initially assigned using the $\delta$ (Eq. 1) or $h$ function (Eq. 2) to the correct window id and that the subsequent loop within Protocol 3 does not alter the window id of the first record.*

*For the induction step, we consider the inner for loop of Protocol 3 for an arbitrary choice of the variable $d$. By induction, we presume that the first $d$ records already have valid window ids. We then make two observations about*

*steps 5-7 of Protocol 3. First, this iteration of the loop makes no changes to the first $d$ window ids, so they continue to remain valid. Second, the protocol then checks the timestamps of records $d+1$ to $2d$ and, if they are set to $\varepsilon$, replaces them with the timestamp from $d$ records earlier. As a result, at the end of the iteration the first $2d$ records are guaranteed to have valid window ids.*

*For the final iteration of the loop, $d \geq |S|/2$. So by the end of the loop, all $|S|$ records have valid window ids, thereby completing the proof.*

**Lemma 6 (Correctness)** *After Protocol 3 is executed with semi-honest adversaries, the window $r_j.w_{id}$ that is assigned to each record $r_j$ will be correct.*

**Proof** *We have already shown in Lemma 5 that every record is eventually assigned to a window. Furthermore, we also showed that this assignment procedure is monotonic: any record that is initially assigned to a window id of $\varepsilon = -1$ during* SESSIONSTART *may have its window id changed later in the procedure, but as soon as a record id $r_j.w_{id}$ is changed to become a valid window id (i.e., not $\varepsilon$), then it will never change again in subsequent iterations of the outer while loop.*

*Therefore it suffices to show the following claim: for every record $r_j$ whose window id $r_j.w_{id}$ is set to $\varepsilon$ during* SESSIONSTART*, its window id is subsequently set to that of the "closest" record before this one that has a valid window id. In other words, Protocol 3 eventually sets $r_j.w_{id} \leftarrow r_i.w_{id}$, where $i$ is the record with the property that: $i < j$, $r_i$ had a valid window id after* SESSIONSTART*, and every record $i^* \in (i, j)$ had an invalid window id $\varepsilon$ after* SESSIONSTART.

*In fact, we will prove a stronger statement by induction on the distance between $i$ and $j$. Specifically, we claim that after the outer while loop completes the iteration with loop parameter $d$, the first $2d-1$ entries to the right of any valid timestamp $r_i.w_{id}$ that were initially $\varepsilon$ will now be set to $r_i.w_{id}$.*

*In the base case where $d = 1$, let $j = i + 1$ and consider the case that $r_i.w_{id}$ is set to a valid, non-negative number but $r_j.w_{id} = \varepsilon$. Now, we examine the execution flow of Protocol 3 during the first iteration of the outer while loop and the $i^{th}$ iteration of the inner for loop:*

- *In step 5, the value $b$ will be set to true, because $r_j.w_{id}$ is less than 0 at the moment.*
- *The oblivious multiplexer within step 7 will change the window id for record $j$ to $r_j.w_{id} \leftarrow r_i.w_{id}$.*

*For the induction step, suppose $2^k \leq j - i < 2^{k+1}$. That is:*

- *At the start of the protocol there were at least $2^k - 1$ records in between $i$ and $j$ that are set by* SESSIONSTART *to the timestamp $\varepsilon$, and*
- *By induction, at the end of iteration $d = 2^{k-1}$ of the outer while loop, it must be the case that the $2^k - 1$ records to the right of $i$ have now had their window ids set to $r_i.w_{id}$.*

*Then, during the $d = 2^k$ iteration of the outer while loop, record $r_j.w_{id}$ will be overwritten with the value of $r_{(j-2^k)}.w_{id}$, which is guaranteed to be equal to $r_i.w_{id}$ by induction. This completes the proof.*

**Lemma 7 (Privacy)** *Protocol 3, when instantiated with either the gap or threshold* SESSIONSTART *protocol, satisfies privacy against one semi-honest computing party, potentially in collusion with one or more data owners or the data analyst. Moreover, for the 4-party instantiation of Protocol 3, oblivious sessionization also maintains correctness against one malicious computing party that is potentially in collusion with one or more data owners or the data analyst.*

At a high level, our sessionization protocol is *oblivious* (i.e., hides all data and has a data-independent control flow) for three reasons:

1. Protocol 3 uses the operations of the underlying mixed-mode MPC protocol in a black-box manner using $\mathcal{F}_{\text{abb}}$.
2. The control flow of Protocols 3, 4, and 5 is data-independent.
3. These operations retain security under composition.


Figure 6: Scalability of the secure window assignment primitives, as the time series size increases from $2^{10}$ to $2^{22}$ (LAN).

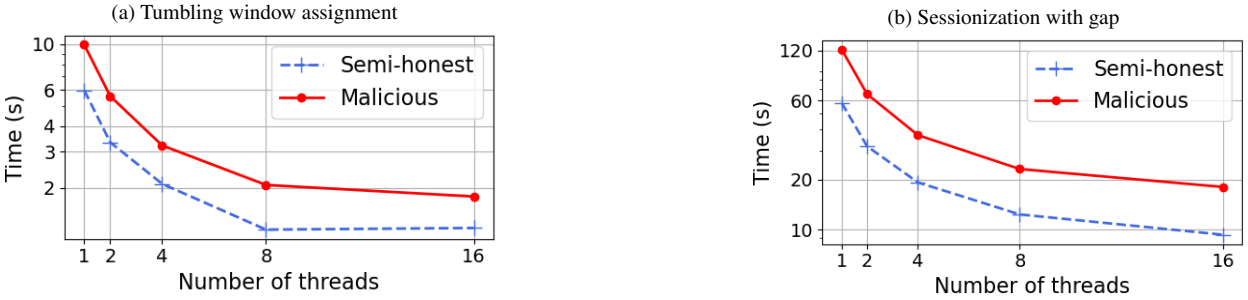


Figure 7: Benefits of parallelism for the windowing primitives on a time series with $2^{20}$ records (LAN).

We make these points more formal in the proof that follows.

**Proof** *The privacy of Protocols 3-5 follows from their black-box use of operations in $\mathcal{F}_{abb}$ that securely compute over boolean secret shares and the fact that they never run $\mathcal{F}_{abb}$.output to open any secrets. Concretely, the arithmetic black-box model supports secure computation of the boolean XOR and AND operations directly, and it is well known how to use them to build the equality, greater-than, and bit expansion operations. Moreover, simulation-based security requires that any instantiation of the arithmetic black-box model provides security both for each individual operator and their sequential composition [70]. Finally, the control flow of Protocols 3, 4, and 5 is data-independent: the for and while loops run for a fixed, public number of iterations and each individual step (e.g., boolean logic, sorting, greater-than comparison, etc) has a data-independent instantiation within* TVA. *Hence, all operations in Protocols 3-5 exclusively perform computations through $\mathcal{F}_{abb}$.*

*As a consequence, it is straightforward to design a simulator $\mathcal{S}$ that emulates the execution of Protocol 3 given only the inputs and outputs of the protocol participants that the adversary has statically corrupted. If the adversary corrupted a data analyst who knows the result of sessionization, then the simulator $\mathcal{S}$ can: use the input $\mathcal{F}_{abb}$.input or shared input $\mathcal{F}_{abb}$.INP protocols to extract the input of any corrupted data owners, calculate inputs for the honest data owners that will lead to the required output, and execute Protocol 3 honestly on that choice of inputs. Otherwise, if the output is unknown, then the simulator $\mathcal{S}$ can execute an instance of Protocol 3 on randomly-chosen inputs; even a distinguisher who knows the true output cannot distinguish this view from a real execution in the $\mathcal{F}_{abb}$-hybrid model.*

# F TVA microbenchmarks

**Scalability of window assignment.** We perform an experiment to evaluate the performance of the window assignment protocols as the input size grows. We set the number of compute threads to 8 and vary the input records from $2^{10}$ to $2^{22}$. Figure 6 shows the results. The tumbling window assignment relies on the division protocol and only takes 6.1s for $2^{22}$ records in the semi-honest setting. Sessionization is also efficient and can identify sessions within 1min for the same semi-honest setting and input. Latency in the malicious setting is only $\sim 2\times$ higher for both primitives. We do not present the latency of threshold-based sessionization as it is almost identical to that of the gap protocol.


Figure 8: Latency of the threshold sessionization primitive in LAN and WAN deployments.

| | Tumbling Window | | | | Session Window | | | |
|---|---|---|---|---|---|---|---|---|
| | in-order records | | out-of-order records | | in-order records | | out-of-order records | |
| | Bandwidth (GB) | Cost ($) | Bandwidth (GB) | Cost ($) | Bandwidth (GB) | Cost ($) | Bandwidth (GB) | Cost ($) |
| **Semi-honest** | 0.21 | 0.035 | 5.94 | 0.51 | 1.52 | 0.22 | 7.25 | 0.69 |
| **Malicious** | 0.50 | 0.071 | 17.68 | 1.74 | 4.53 | 0.68 | 21.71 | 2.35 |

Table 4: Bandwidth and cloud costs of tumbling and session window operators on ordered and unordered time series with $2^{20}$ records

**Benefits of parallelism.** We now evaluate TVA's ability to leverage parallelism. We set the input size to $2^{20}$ records and measure the latency of window assignment for tumbling and session windows. Figure 7 shows the results as we increase the number of threads per computing party from 1 to 16. We see that TVA can effectively utilize the available cores and share work among multiple threads. With 16 threads per party, TVA computes sessions in 18s, which is 6.7× faster compared to the single-threaded execution.

**Amortizing I/O.** We now investigate TVA's ability to amortize I/O in a high-latency network. Figure 8 shows the latency of the threshold sessionization primitive in LAN and WAN for both MPC protocols. As the input size increases, the performance gap narrows significantly: at $2^{22}$ records, TVA identifies sessions with only 4.6× higher latency in WAN compared to LAN, despite the former's 40-50ms RTT. This result demonstrates that TVA's batching and vectorization are effective in amortizing the communication costs of MPC.

**Cloud costs.** We now calculate cloud costs for TVA's window and multi-predicate queries in our WAN setting on AWS. The `r5.8xlarge` instances we use for the TVA servers cost $2.016/h, while cross-region bandwidth costs $0.02/GB [5]. Table 4 shows the transmitted data per party and the total monetary costs (including compute instance charges) for TVA's window operators using a time series with $2^{20}$ records and 32-bit share representation. When records are out of order, bandwidth and costs are higher due to sorting. We only show results for gap-based session windows, as threshold windows have similar costs. The overall cost of the equality and range queries we used in Section 6.1 is much lower: $0.009 and $0.028 in the semi-honest and malicious setting, respectively.

# G The TVA API

Listing 2 shows the overall structure of TVA queries:

```
ts_data                    // handle to secret-shared data
  [.filter(...)]             // optional: apply conditions
  [.keyBy(...)]              // optional: partition by key
  .window(...) | .snapshot(...)  // required: temporal op
  .aggregate(...)                // required: aggregation
```

Listing 2: TVA query structure

Each query includes exactly one temporal operator (`window` or `snapshot`) followed by an aggregation. Operators are applied on a secret-shared time series `TS` and output a new secret-shared time series. The only exception is

the `aggregate` operator that outputs one or more aggregated values, as we explain below. TVA's API exposes the following methods:

- `get_shares(schema)` → `TS` retrieves the secret-shared records from data sources and defines the time series schema, i.e., attribute names. Each attribute has a share type (boolean or arithmetic) and a predefined representation of length $\ell$.
- `filter(predicate)` → `TS` is used to select a subset of records that satisfy some conditions. The conditions are expressed as logical predicates on data attributes and are combined with `AND` and `OR` operators.
- `keyBy(keys)` → `TS` groups records on one or more data attributes (`keys`) and is used to define per-group operations: when a `keyBy` exists in a query, the time series is logically divided into disjoint parts and the subsequent operators are applied to each part independently.
- `snapshot(interval)` → $\texttt{TS}_s$ selects a subset of records that correspond to the given time `interval`. The method returns a new time series $\texttt{TS}_s$ on which a subsequent aggregation can be applied.
- `window(strategy)` → $\texttt{TS}_w$ assigns each record in the input time series to windows according to the given `strategy`. The method returns a new time series $\texttt{TS}_w$ on which a window aggregation can be applied. The formal semantics of the new time series depend on the window and are given in §4. Our API also provides three shortcut methods for each of the supported strategies, namely `tumbling_window()`, `threshold_window()`, and `gap_window()`.
- `aggregate(IN, OUT, agg_fun)` → `List[V]` applies the aggregation function `agg_fun` on the attribute `IN` and writes the result in the attribute `OUT`. This function needs to be applied as a separate (required) step following a `snapshot` or `window` operator. In the case of a snapshot query, the aggregation is applied to one or more groups of records and returns a single aggregated value (global aggregation) or a collection of aggregated values (one per group), if combined with `keyBy`. In the case of a window query, the aggregation is applied to each identified window and returns a collection of aggregate values (one per window).

# H Queries for real world applications

```
// Data schema
TS ts = get_shares({"TIMESTAMP",  "ENERGY_CONSUMPTION",
                    "TOTAL_CONSUMPTION"});

// Window aggregation
TS res = ts.tumbling_window("TIMESTAMP", 3600)
           .aggregate("ENERGY_CONSUMPTION",
            "TOTAL_CONSUMPTION",  Agg::SUM);
```

Listing 3: The energy monitoring query in the TVA API

```
// Data schema
TS ts = get_shares({"TIMESTAMP", "PATIENT_ID", "GLUCOSE",
                    "INSULIN", "TOTAL_INSULIN_EVENTS"});

// Window aggregation
TS res = ts.keyBy("PATIENT_ID")
           .threshold_window("TIMESTAMP", "GLUCOSE", 5)
           .aggregate("INSULIN", "TOTAL_INSULIN_EVENTS",
            Agg::COUNT);
```

Listing 4: The mobile health query in the TVA API

```
// Data schema
TS ts = get_shares({"TIMESTAMP",  "MACHINE_TYPE",
              "EVENT_TYPE", "TOTAL_TASKS_PER_SESSION"});

// Window aggregation
```

```
TS res = ts.filter("EVENT_TYPE" == 0)
           .keyBy("MACHINE_TYPE")
           .gap_window("TIMESTAMP",  10)
           .aggregate("*", "TOTAL_TASKS_PER_SESSION",
            Agg::COUNT);
```

Listing 5: The job scheduling query in the TVA API