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

Paper Summary

1 Summary

Note: Implementation available

1. Novel protocols for Secure Window Assignment
   - Tumbling Window: Groups records into fixed length time buckets

     Secure division with public divisor: Motivation - State of art protocols for fast division produce small errors acceptable in ML use cases but cause incorrect aggregation results as they end up in wrong windows with high probability. Therefore, TVA proposes error correction protocols for semi-honest and malicious secure division to compute tumbling windows.

     Use case: Per-hour energy consumption of smart grid
   - Gap Session Window

     Use case 1: Identifies periods of "activity" followed by periods of "inactivity" such as average glucose level in a patient cohort during their sleep or exercise

     Use case 2: Time periods when individual is moving
   - Threshold Session Window

     Use case: Eating periods when glucose level exceeds a threshold
2. Limitations of Prior works
   - Waldo optimized for snapshot queries on single time interval (2 DCF keys to check if $a < x < b$), cannot efficiently perform recurring computations

     TVA: Performant secure window assignment protocols for fixed and custom length time intervals
   - Prior works only support global aggregations (total consumption for all postcodes) or limited additive keyed aggregation (requires data holders to pre-encode attribute domain i.e. all possible postcodes)

     TVA: Supports keyed aggregations (hourly energy consumption per postcode)
   - Prior works assume regular, ordered, public timestamps
3. Supported Workloads
   - Temporal operators

     Snapshot queries that compute an aggregate over specific time interval

     Window queries that compute an aggregate over multiple time intervals : Tumbling Window operator, Gap Session Window operator, Threshold Session Window operator
   - Time agnostic operators

     Filtering:

     Sorting:

     Grouping:

     Distinct (special case of Grouping):

     Aggregations: COUNT, SUM, MIN, MAX, TOP-K, AVG, STDEV, PERCENTILE

     Operations: Logical (AND, OR, XOR), Arithmetic ($+$, $-$, $*$), Comparison ($\geq$, $\leq$, $=$, $\neq$)

2 Setting

• Public Data schema, type and no. of columns, no. of input records.
• Supported i) Semi-honest 3 party, 1 corruption, replicated secret sharing (Araki) (ii) Malicious 4 party, 1 corruption, replicated secret sharing (Fantastic Four by Dalskov)
• Query structure known to computing parties

3 Prime Techniques

3.1 Tumbling Window Operator

Example query: COUNT over time intervals of length λ across individuals in a cohort
Consider n rows, t represents timestamp attribute

1. Secure Division: $w_{id} = \lfloor \frac{t}{\lambda} \rfloor$ where t is secret shared and λ is public:
    - For 1 record
      Communication: $2\ell_t + 2 \log \lambda (\log \log \lambda) + \log \lambda + 1$ bits (ℓ_t represents no. of bits timestamp t is secret shared over)
      Rounds: $\log \log \lambda + 3$ rounds
    - For n records
      Communication: $n(2\ell_t + 2 \log \lambda (\log \log \lambda) + \log \lambda + 1)$ bits (ℓ_t represents no. of bits timestamp t is secret shared over)
      Rounds: $\log \log \lambda + 3$ rounds
2. Sort records by w_{id}
   Communication: $O(n \log^2 n)$ gates i.e. $n \log^2 n c_{comp}$ bits where $c_{comp} = c \log g$
   Rounds: $O(\log^2 n) rounds$
3. Groupwise COUNT aggregation
   Communication: $n \log n c_{eq}$ bits where $c_{eq} = c \log g$
   Rounds: $O(\log n)$ rounds

References

[1] Muhammad Faisal, Boston University; Jerry Zhang *TVA: A multi-party computation system for secure and expressive time series analytics*, USENIX (2023)

[2] Gilad Asharov, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Ariel Nof, Benny Pinkas, Katsumi Takahashi, Junichi Tomida *Efficient Secure Three-Party Sorting with Applications to Data Analysis and Heavy Hitters*, CCS (2022)

[3] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, Mayank Rathee *Function Secret Sharing for Mixed-Mode and Fixed-Point Secure Computation*, Eurocrypt (2021)

Waldo: A Private Time-Series Database from Function Secret Sharing
Paper Summary

1 Summary

Note: Implementation available

1. Multi-predicate functionality
   - Additive aggregates based on multiple predicates: Sum, Count, Mean, Variance, SD
   - Arbitrary aggregates over time interval: Max, Min, Top-K
   - Technique: FSS + Replicated Secret Sharing
   - Setting: Data contents, query filter values, search access patterns protected
   - Baseline: ORAM solutions
2. 3 party malicious security with honest majority
3. Constraint: Requires small feature size for multi-predicate evaluation
4. Constraint: Does not support retrieval of individual records, sorts, group by, joins

2 Techniques to calculate No. of Records Matching Single Predicate with Semi Honest Security

Applying FSS to **private queries** over **public database** is well studied. Waldo tries to apply FSS to **private queries** over **private database**. Using the **FSS for secure computation** idea for this would mean that each entry x_i should be masked with independently sampled r_i, implying that each entry is mapped to a function $f(x - r_i)$. Therefore, size of function shares would match size of database. Therefore, this is not explored.

(Inspired by Dory) Waldo builds a table of size n (no. of rows) x k (no. of candidate values for feature), for each feature. So for f features, we have f such tables. This is a shared one hot index. For each record, corresponding row in table set to 1 at location corresponding to record value and 0 elsewhere. For each entry $d_{i,j}$ of this table, server evaluates FSS key on current value j, multiplies by table entry $d_{i,j}$.

Waldo observation 1: Feature size needs to be small. In general, only values compared in predicates need to use small feature size, not values being aggregated. Techniques to encode values in large domain using small feature size:
• Bucketing intervals. Affects precision. Preserves ordering for range predicates, efficient to compute aggregate stats. Explored in prior works.
• Hash maps/Bloom filters for ID fields
• Conjunction of predicates e.g. representing time as timestamp or conjunction of year, month, day, hour, minute

Private updates not supported. **Appends are straightforward.**

3 Techniques to calculate No. and Sum of Records Matching Multiple Predicates with Semi Honest Security

Form a filter i.e. vector of size n s.t. value at index i = 1 if record i $\in [n]$ matches predicate, 0 otherwise. For multiple predicates, filters need to be combined i.e. multiplied for logical AND. Therefore, Replicated Secret sharing.

4 Techniques to calculate Complex Aggregates (Min/Max/Top-K) over Time Range with Semi Honest Security

Salient features:
• No server-server interaction
• Time interval being queried not revealed
• Valuable where query predicates are predefined

Solution: Shared aggregate tree. Each leaf node contains (private) record value, each internal node contains (private) aggregate of its 2 children. Each leaf node has public timestamp. n leaf nodes ordered by time so that each internal node has a public time interval. Hence, client computes aggregate over some time interval by retrieving at most $2\log n + 1$ nodes. Query result = Aggregate of these intermediate aggregates.

Client cannot directly request covering set from server, as this set reveals to server time period being queried.

Strawman solution: Client sends $2\log n + 1$ logical DPF keys to servers.
Better solution: Client generates DCF keys for leaf level of tree, sends keys to server. Each server evaluates its DCF keys on timestamp for each leaf separately for both $a < b$ and $x < b$. A leaf node is considered activated if its DCF evaluation is a secret share of 1 i.e. node is within range of client query. Activation result is obliviously copied to parent node. Client finally receives log n values for each of the single sided ranges which it uses to recover covering set for intersection of the 2 ranges.

Multiple Interval Containment FSS technique from Eurocrypt paper is incorporated in Waldo.

5 Communication and Round Complexities

Let N be number of records, m be number of features.

For each feature $i \in [m]$, we store a table of size $N \times 2^{l_i}$ where $[2^{l_i}]$ represents all possible candidate values for feature i.

Cost: $mN2^l$ bits, 1 round

Given a single predicate query, suppose we want to compute no. of records matching the predicate. For every entry $d_{i,j}$ in the shared one hot index table D for record $i \in [N]$ and feature value $j \in [2^l]$, the server s evaluates its FSS key K on the current value j, multiplies the evaluation by the table entry $d_{i,j}$ and then computes

EvalPred(s, K, D) ← $\sum_{j=0}^{2^l-1}(Eval(K,j).\sum_{i=0}^{N} d_{i,j})$

For q such single predicate evaluations, Cost: $q(\lambda\ell + 3\log n)$ bits, 1 round

In case of a multi-predicate query, the following is computed

FilterPred(s, K, D) ← $\sum_{i=0}^{2^l-1}(Eval(K,i).d_{0,i}), ..., \sum_{i=0}^{2^l-1}(Eval(K,i).d_{N-1,i})$

For q multi-predicate queries with p predicates per query:

Cost: $qp(\lambda\ell + 3n\log n)$ bits, 1 round (Check?)

Prio: Private, Robust, and Scalable Computation of Aggregate Statistics

Paper Summary

1 Summary

1. Operations covered
   - X(A) where X = SUM, MEAN, Variance, STDDEV, Most popular (Approx), Heavy Hitters (Approx), Boolean OR and AND, MIN and MAX (Approx), Frequency COUNT, Intersection and Union of sets, Quality of arbitrary regression model (R^2), Least-squares regression, Stochastic gradient descent
2. Setting
   - Consider that each client has a private data point
   - Clients send an encrypted share of their data to each aggregator
3. Security Setting
   - Exact Correctness: If all servers are honest, servers learn f(.)
   - Privacy: If at least one server is honest, servers learn only f(.)
   - Robustness: Robust only against adversarial clients, not adversarial servers.
4. Small no. of servers, Very large no. of clients

2 Strawman Approach

Problem: Consider that every device i holds a value x_i (0 or 1). We want to compute $f(x_1, ..., x_N) = x_1 + ... + x_N$ without learning any user's private value x_i. Suppose at least one server is honest and we have 3 servers.

Approach

1. Every device i distributes 3-party secret shares of its value x_i to the 3 servers.
2. At the end, each server holds a share of the total sum i.e. $f(x_1, ..., x_N)$

Issues with Approach: A single malicious client can undetectably corrupt the sum. Typical defenses (NIZKs) are costly.

3 Technical Contributions

1. Secret Shared Non-Interactive Proofs (SNIPs): Client proves that its encoded submission is well-formed i.e. proves that its shares sum up to 0 or 1.
   - Servers hold shares of client's private value x
   - Servers hold an arbitrary public predicate Valid(.) expressed as an arithmetic circuit.
   - Want to test if Valid(x) holds, without leaking x.
   - Cost: 0 public key operations for both Client and Server, $\theta(M)$ Client to Server Communication, $O(1)$ Server to Server Communication where M = No. of Multiplication gates in Valid(.) circuit. For specific Valid() circuits, it's possible to eliminate the $\theta(M)$ cost for Client to Server communication. (From presentation)
2. Aggregatable Encodings: Can compute sums privately $\implies$ Can compute f(.) privately (for many f's of interest)

4 Cost

Refer Table 2 (Taken from paper)

	Exps	**Muls**	**Proof Len**
Client	0	$M \log M$	M
	Exps/Pairs	**Muls**	**Data Transfer**
Server	0	$M \log M$	1

Table 1: Asymptotic cost for PRIO. Client holds a vector $x \in \mathcal{F}^M$, each server i holds a share $[x]_i$, and the client convinces the servers that each component of x is a 0/1 value in $\mathcal{F}$

5 Overview of Technique

PRIO computes $f(x_1, ..., x_n)$ privately as follows:

1. Each client encodes it data value x using AFE Encode routine for the aggregation function f.
2. Every client splits its encoding s shares and sends one share to each of s servers.
3. Client uses a SNIP proof to convince the servers that its encoding satisfies the AFE Valid predicate.
4. Upon receiving a client's submission, the servers verify the SNIP to ensure that the encoding is well formed.
5. If the servers conclude that the encoding is valid, every server adds the first k' components of the encoding share to its local running accumulator (k' is a parameter of the AFE scheme).
6. Finally, after collecting valid submissions from many clients, every server publishes its local accumulator, enabling anyone to run the AFE Decode routine to compute the final statistic in the clear.

Computationally Secure Aggregation and PIR in the Shuffle Model

Paper Summary

1 Summary

Problem Statement: (Secure Aggregation in the Shuffle Model). Let $\mathbb{F}$ be a finite field, and let $n, c \in N$. A (single server) secure aggregation protocol over c clients in the shuffle model is a tuple of algorithms ShAgg = (Share,Mix,Aggregate):

1. Share($x \in \mathbb{F}^n$) $\rightarrow^{\$}$ S: A randomized algorithm executed by the client that takes in an input $x \in \mathbb{F}^n$, and outputs a set of shares S.
2. Mix($S_1, \ldots, S_c$) $\rightarrow^{\$}$ $\hat{S}$: A randomized algorithm executed by the shuffler that takes in the sets of shares from c clients $S_1, \ldots, S_c$, and output a set of shuffled shares $\hat{S}$.
3. Aggregate($\hat{S}$) $\rightarrow a$: A deterministic algorithm executed by the server that takes in the shuffled elements and outputs an aggregated answer a.

2 Strawman Approach

Problem: Consider that every device i holds a value x_i (0 or 1). We want to compute $f(x_1, ..., x_N) = x_1 + ... + x_N$ without learning any user's private value x_i. Suppose at least one server is honest and we have 3 servers.
Approach

1. Every device i distributes 3-party secret shares of its value x_i to the 3 servers.
2. At the end, each server holds a share of the total sum i.e. $f(x_1, ..., x_N)$

Issues with Approach: A single malicious client can undetectably corrupt the sum. Typical defenses (NIZKs) are costly.

3 Technical Contributions

1. Secret Shared Non-Interactive Proofs (SNIPs): Client proves that its encoded submission is well-formed i.e. proves that its shares sum up to 0 or 1.
    - Servers hold shares of client's private value x
    - Servers hold an arbitrary public predicate Valid(.) expressed as an arithmetic circuit.
    - Want to test if Valid(x) holds, without leaking x.
    - Cost: 0 public key operations for both Client and Server, $\theta(M)$ Client to Server Communication, $O(1)$ Server to Server Communication where M = No. of Multiplication gates in Valid(.) circuit. For specific Valid() circuits, it's possible to eliminate the $\theta(M)$ cost for Client to Server communication. (From presentation)
2. Aggregatable Encodings: Can compute sums privately $\implies$ Can compute f(.) privately (for many f's of interest)

4 Cost

Refer Table 2 (Taken from paper)

	Exps	**Muls**	**Proof Len**
Client	0	$M \log M$	M
	Exps/Pairs	**Muls**	**Data Transfer**
Server	0	$M \log M$	1

Table 2: Asymptotic cost for PRIO. Client holds a vector $x \in \mathcal{F}^M$, each server i holds a share $[x]_i$, and the client convinces the servers that each component of x is a 0/1 value in $\mathcal{F}$

5 Overview of Technique

PRIO computes $f(x_1, ..., x_n)$ privately as follows:

1. Each client encodes it data value x using AFE Encode routine for the aggregation function f.
2. Every client splits its encoding s shares and sends one share to each of s servers.
3. Client uses a SNIP proof to convince the servers that its encoding satisfies the AFE Valid predicate.
4. Upon receiving a client's submission, the servers verify the SNIP to ensure that the encoding is well formed.
5. If the servers conclude that the encoding is valid, every server adds the first k' components of the encoding share to its local running accumulator (k' is a parameter of the AFE scheme).
6. Finally, after collecting valid submissions from many clients, every server publishes its local accumulator, enabling anyone to run the AFE Decode routine to compute the final statistic in the clear.

Constant-Round Private Decision Tree Evaluation for Secret Shared Data

Paper Summary

1 Summary

1.1 Setting

1. 3 servers
2. Service provider (tree owner) has the trained DT model. It generates and distributes 3P shares of the tree to the 3 servers.
3. Client (can be many clients for many evaluations) with an attribute vector as input. It generates and distributes 3P shares to 3 servers.
4. (2, 3) RSS, Semi honest security, honest majority (only 1 out of 3 corruptions permitted)

1.2 Results

1. 5 round online phase

1.3 Decision Tree Representation

Consider a complete binary tree with height h, size of an attribute vector m.

Internal Node:

1. (i, j) where i represents level number (ranges from 0 to h-1) and j represents position within a level (ranges from 0 to $2^i - 1$.
2. cond = 0 (equality check) or 1 (less than)
3. attribute index (replicated secret shared over m-bit ring)
4. threshold (replicated secret shared over k-bit ring)

Leaf Node:

1. Leaf no. i (ranges from 0 to $2^h - 1$)
2. value v (replicated secret shared over k bit ring)

1.4 Protocol Overview

1. Feature Selection (Offline: $10mk(2^h - 1)$ bits, 2 rounds, Online: $(4mk + 3\log m)(2^h - 1)$ bits, 1 round)
   - i/p: Share of index (idx), Shared attribute m-length vector (x)
   - o/p: Share of attribute corresponding to index (1 k-bit ring element)
   - Approach:
     - Oblivious circular shift of x by randomness η, call it y.
     - $s = idx + \eta$ and reveal s
     - Output y_s
2. Comparison (Offline: $(2(\lambda + 7)(k + 1) - 5)(2^h - 1)$ bits, Online: $(4k + 13)(2^h - 1)$ bits, 2 rounds)
   - Share conversion (for selected attribute and threshold): (3, 2) RSS to (2, 2) additive shares
   - Compute in parallel: a = LT(Selected attribute, Threshold), b = EQ(Selected attribute, Threshold) in parallel (1 round).
   - Compute $b \oplus$ SC-AND$(cond, a \oplus b$. SC-AND to evaluate AND gate and share conversion at the same time.
   - Convert above back to (3, 2) RSS.

3. Path Evaluation (Offline: $4(2^h - 1)$ bits, 2 rounds, Online: $5k.2^h + 4(2^h - 1)$, 2 rounds)
    - i/p: Shares of $2^h - 1$ leaf values, shares of comparison results at every inner node
    - o/p: Shares of value associated with corresponding leaf node indicated by comparison results
    - Idea: Oblivious tree shuffle - Shuffle a shared tree obliviously while sustaining tree structure. Then, reveal the shuffled node values, detect the right trace path while leaking no info on i/p tree.
    - Suppose c is the complete binary tree until depth $h - 1$, which consists of just 0s and 1s representing results of comparisons on the internal nodes.
    - Let r be a complete binary tree until depth $h-1$ with 0s and 1s randomly sampled. Consider that permutation π_r is calculated in a special way from r.
    - Let v represent the vector of leaf values.
    - $(c, v).r = (c \circ r, v.\pi_r)$ and $c \circ r = (c.\pi_r) \oplus r = (c \oplus r.\pi_r^{-1}).\pi_r$

1.5 Limitations

1. Semi-honest security
2. Operation at inner node limited to LT and EQ.
3. We deal with complete binary tree.