Conclave: secure multi-party computation on big data

Nikolaj Volgushev Malte Schwarzkopf † Ben Getchell Andrei Lapets
Mayank Varia Azer Bestavros

Boston University † MIT CSAIL

Abstract

Secure Multi-Party Computation (MPC) allows mutually distrusting parties to run joint computations without revealing private data. Current MPC algorithms, however, scale poorly with data size, making MPC on "big data" prohibitively slow and inhibiting use cases.

Many relational analytics queries, however, can maintain MPC's end-to-end security guarantee without using cryptographic MPC techniques for all operations. Conclave is a query compiler that accelerates such queries by transforming each query into a combination of data-parallel, local cleartext processing and small, isolated MPC steps. When parties trust others with specific subsets of the data, Conclave applies new hybrid MPC-cleartext protocols to run additional steps outside of MPC and improve scalability.

Our Conclave prototype generates and executes code for scalable cleartext processing in Spark, and for secure MPC using the Sharemind and Obliv-C frameworks. Conclave scales to data sets between three and six orders of magnitude larger than state-of-the-art MPC frameworks support on their own. Conclave's optimizations enable it to outperform SMCQL, the most similar existing system, by 6–50×.

1 Introduction

Many businesses run analytics over "big data" to draw insights or to satisfy regulatory requirements. Such computations cannot currently combine proprietary, private data sets from multiple sources, whose sharing is restricted by law and by justifiable privacy concerns. However, running joint analytics over large private data sets is valuable: for example, drug companies, medical researchers, and hospitals can benefit from jointly measuring the incidence of illnesses without revealing private patient data [3; 44; 63]; banks and financial regulators can assess systemic risk without revealing their private portfolios [8; 47]; and antitrust regulators can measure monopolies using companies' revenue data.

Secure Multi-Party Computation (MPC) is a class of cryptographic techniques that allow for secure computation over sensitive data sets. In MPC, a set of participants compute jointly over the data they hold individually, while federating trust among the computing parties. As long as some parties — typically a majority, or any one party — honestly follow the protocol, no party learns anything beyond the final output of the computation. In particular, MPC protects input and intermediate data and meta-data statistics — such as value frequency distributions — about them.

Unfortunately, implementing a performant secure MPC currently requires domain-specific expertise that makes it impractical for most data analysts. Existing algorithmic techniques and software frameworks for secure MPC scale poorly with data size, even for small numbers of parties (§2). These limitations have led researchers to build oblivious query processors that generate and execute secure query plans [3; 63]. These query processors improve accessibility of MPC by allowing data analysts to write convenient relational queries, and performance by securely doing part of the computation outside of MPC. For example, SMCQL [3] performs local preprocessing and "slices" the overall MPC into several smaller MPCs to reduce input data size, while Opaque [63] relies on Intel SGX hardware to compute securely "in-the-clear".

This paper presents Conclave, an MPC-enabled query compiler that makes MPC on "big data" accessible and efficient. Data analysts write relational queries as if they had access to all parties' data in the clear, and Conclave turns the queries into a combination of efficient, local processing steps and secure MPC steps. As a result, Conclave delivers results with near-interactive response times — *i.e.*, a few minutes — for input data several orders of magnitude larger than existing systems can support.

Three key ideas help Conclave scale. First, Conclave analyses the queries to apply transformations that reduce runtime without compromising security guarantees, even though they burden individual parties with extra local work. Second, Conclave uses optional, coarse-grained annotations on input relations to apply new, "hybrid" cleartext–MPC protocols that gain further speedups especially for operations that are notoriously slow under MPC, such as joins and grouped aggregations. Third, Conclave generates code that combines scalable, insecure data-processing systems (*e.g.*, Spark [61]) with secure, but slow, cross-party MPC systems (*e.g.*, Sharemind [11] or Obliv-C [62]).

Conclave's optimizations are largely complementary to those introduced by SMCQL [3], the most similar related system. Conclave introduces new hybrid protocols that improve the efficiency of joins and aggregations over private key columns, as well as additional query transformations that speed up the MPC steps. Moreover, Conclave generates code for both garbled-circuit and secret-sharing MPC frameworks, as well as for data-parallel local processing systems.

Secret-sharing MPC backends are better suited to the arithmetic operations prevalent in relational queries, and thus outperform SMCQL's garbled-circuit backend; data-parallel local processing allows Conclave to support larger inputs.

In summary, this paper makes four key contributions:

1. analyses that derive which parts of a relational query must be executed under MPC using only coarse-grained annotations (§4, §5.1);
2. transformations that move expensive oblivious operations outside the MPC while preserving security guarantees (§5.2, §5.4);
3. new "hybrid" MPC–cleartext protocols that improve performance of MPC joins and aggregations using existing partial trust between parties (§5.3); and
4. our prototype implementation of Conclave, which applies these ideas to generate efficient code for execution using sequential Python, Spark [61], Obliv-C [62] and Sharemind [11] (§6).

We measured the performance of our prototype using microbenchmarks and end-to-end relational analytics queries (§7). Even with minimal annotations on input relations and basic optimizations, Conclave scales queries to input data orders of magnitude larger than existing MPC frameworks support on their own. Our new "hybrid" MPC–cleartext protocols speed up join and aggregation operators by 7× or more compared to execution in Sharemind [11], a fast, commercial MPC framework. Compared to SMCQL, Conclave executes a medical research query 6–50× faster.

Nevertheless, our prototype has some limitations. Like similar systems, it defends only against honest-but-curious attackers, though Conclave's approach is compatible with stronger threat models. While prototype reimplements some of SMCQL's techniques for fair comparison, it does not support all; an ideal system would combine the two systems' techniques. Finally, our prototype supports only the Obliv-C and Sharemind MPC frameworks, which limit MPC steps to two or three parties, but adding support for further frameworks requires only modest effort.

2 Motivation and background

MPC allows joint execution of agreed-upon computations across several parties' private data without a trusted party. MPC has served purposes from detecting VAT tax fraud by analyzing business transactions [9], to setting sugar beet prices via auction [12], relating graduation rates to employment [10], and evaluating the gender pay gap across businesses [7]. However, these applications all compute only over a few hundreds or thousands of records, while many useful computations on large data that would benefit from MPC are currently infeasible.

2.1 Use cases for MPC on "big data"

We sketch two example applications of MPC that would be worthwhile if MPC could run efficiently on large data.

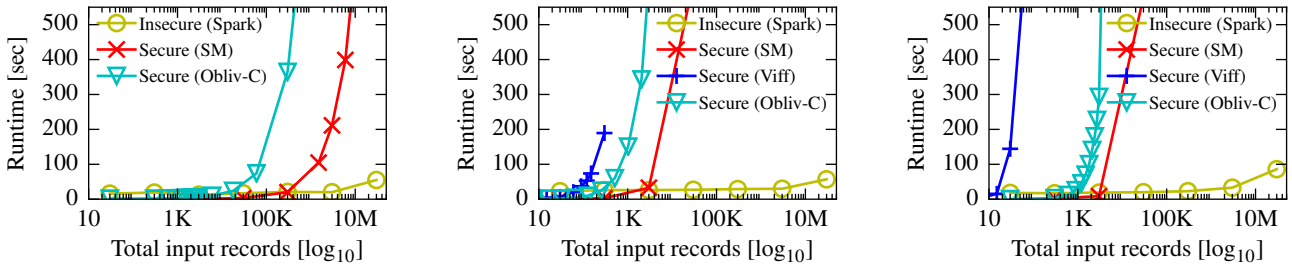
Credit card regulation. A government regulator (in the U.S., the OCC [56]) who oversees consumer credit reporting agencies (in the U.S., *e.g.*, TransUnion) may wish to estimate the average credit score by geographic area (*e.g.*, ZIP code). The government regulator holds the social security numbers (SSNs) and census ZIP code of potential card holders; credit reporting agencies, by contrast, have the SSNs of card holders, their credit lines, and their credit ratings. By law, the government regulator cannot share the residence information. Likewise, credit reporting agencies cannot share raw portfolios for fear of leaking information to competitors through carelessness or compromise, so MPC is needed. The input to this query is large: there are over 450M SSNs [57] and 167M credit cards [54] currently issued in the U.S.

Market concentration. Competition law requires governments to regulate markets to prevent oligopolies or monopolies. Regulators often use the Herfindahl-Hirschman Index (HHI) — the sum of squared market shares of companies active in a marketplace — to decide whether scrutiny is warranted [55, §5.3]. Public revenue data is coarse-grained, and the market shares of privately-held companies are difficult to obtain. For example, airport transfers in New York City constitute a marketplace, for which an effective HHI considers *only* the revenue derived from airport transfers in the market shares. Airport transfers made up 3.5% of 175M annual NYC yellow cab trips in 2014; many trips were serviced by other vehicle-for-hire (VFH) companies [45]. While the HHI computation inputs are small (a single number per company), computing them requires filtering and aggregating over millions of trip records that companies keep private.

2.2 Security guarantees

MPC guarantees the *privacy* of a computation's input and intermediate data. Specifically, MPC reveals no more about each party's input data than can be inferred from the final, publicly revealed output of the computation. MPC also guarantees *correctness* of the output revealed to each party, and can provide *integrity* properties [41].

MPC provides these guarantees under a specific model of its adversary (*i.e.*, dishonest participant). Commonly, the adversary is presumed to be *honest-but-curious*, respecting the protocol but trying to learn other participants' data, or *malicious*, actively deviating from the protocol's rules to expose private data or compromise the integrity of outputs. Generic techniques for information-theoretically secure MPC require an honest majority [6; 49] whereas techniques that leverage computational assumptions only require a single honest party (an "anytrust" model) [26; 59]. Relaxing security guarantees typically makes MPC faster: honest (super-)majority techniques perform better than anytrust ones [2; 11; 23], and an honest-but-curious adversary can be withstood with at least 7× lower overhead than a malicious adversary [1].



(a) `PROJECT`. **(b)** Aggregation (`SUM`). **(c)** `JOIN`.

Figure 1. Existing MPC frameworks only scale to small data sets for common relational operators, *e.g.*, aggregations and joins. By contrast, Spark runs these operators on tens of millions of records in seconds (note the log-scale x-axis).

2.3 MPC techniques and scalability

Since the 1980s [6; 26], two dominant MPC approaches have emerged: garbled circuits [59] and secret sharing [52]. While cryptographers have made substantial effort to scale them to many *parties* [14; 16–18], scalability to *large data* remains a challenge. In this work, we focus on scaling to large data sizes, and assume a constant, small number of parties.

With **garbled circuits**, one party encrypts each input bit and intermediate state to create a "wire label". Then, it converts the computation into a circuit of binary gates, each expressed as a "garbled truth table" comprising a few ciphertexts. A single evaluator party receives the circuit and the encrypted wire labels for all input bits. At each gate, the evaluator combines the inputs to produce an encrypted output using the garbled truth table. Garbled circuits require no communication between parties during evaluation, but their state is far larger than the input data (*e.g.*, 80–128× for typical security parameters), which makes processing of large data impractical.

Secret sharing, by contrast, splits each sensitive input (*i.e.*, each integer, rather than each bit) into several "secret shares", which in combination yield the original data. A common encoding chooses secret shares such that they cancel out into the encoded cleartext value when added together. Each computing party works on one secret share; additions happen locally and without communication, but multiplications require interaction before or during the evaluation [4–6]. Secret sharing only multiplies state size by the number of shares, but the communication (*i.e.*, network I/O) during computation limits scalability. Batching communication helps reduce overheads, but a multiplication requires sending at least one bit between parties [2].

Practical scalability. Figure 1 compares insecure plaintext execution of three relational operators to execution in MPC frameworks using secret-sharing (Viff [58] and Sharemind [11]) and garbled-circuits (Obliv-C [62]). Each experiment inputs random integers and runs a single operator. The MPC frameworks run with two (Obliv-C) or three (Sharemind, Viff) parties, who in aggregate contribute the record count on the log-scale x-axis; insecure computation runs a single Spark job on the combined inputs. A projection (Figure 1a) requires only a single pass over the input and no communication even under MPC; hence, it should scale well. In practice, though, Obliv-C's circuit state growth limits its performance (it runs out of memory at 300k records), and Sharemind takes over 10 minutes beyond 3M input records (≈37 MB) due to overheads of secret-sharing and in its storage layer implementation. MPC scales far worse for aggregations (Figure 1b) and joins (Figure 1c). These operators require communication in secret sharing, and Obliv-C's state grows fast (*e.g.*, the join ran out of memory at 30k records). Thus, MPC only scales to a few thousand input records.

These results are consistent with prior studies: Sharemind takes 200s to sort 16,000 elements [36], and DJoin takes an hour to join 15,000 records [44]. Current MPC systems therefore seem unlikely to scale to even moderate-sized data sets. In particular, the poor performance of joins and aggregations is concerning: more than $^3/_5$ of practical analytics queries use joins, and over $^1/_3$ contain aggregations [34, §2]. Short of new cryptographic techniques, the best way to run MPC on large data may therefore be to *avoid* using its cryptographic techniques unless absolutely necessary.

3 Conclave overview

Conclave builds on the insight that the end-to-end security guarantees of MPC often hold even if parts of a query run outside MPC. This allows Conclave to use cheaper algorithms, local computation, and data-parallel processing systems for parts of the query — all crucial to scaling MPC to large data.

Conclave's guiding principle is *to do as little as possible and as much as necessary under MPC*: in other words, Conclave minimizes the computation under MPC until no further reduction is possible. Intuitively, any operation computed using only a party's local inputs and publicly available data can run outside MPC, as can any operation that applies only reversible operations to reach a final, revealed result.

3.1 Threat model

Like many practical MPC systems, Conclave assumes an *honest-but-curious* adversary that can observe all network communication during execution of the protocol and also

statically (*i.e.*, before the protocol begins) choose parties that it can corrupt and whose data in their local file system and memory it can inspect. Because the adversary monitors the computation's control flow and memory access patterns, the MPC must use *oblivious* operations that avoid data-dependent control flow and hide access patterns. However, all parties — including corrupted ones — faithfully execute the MPC protocol and submit valid input data.

3.2 Security guarantees and assumptions

Conclave inherits the security guarantees (and assumptions) of its MPC backend. Additionally, Conclave provides the option to trade some security for better performance.

By default, Conclave provides the full guarantees of MPC; in particular, Conclave hides all private data processed in the MPC step, along with all meta-data such as the size of intermediate relations. Consistent with MPC literature, Conclave treats the cardinalities of all *input relations* as public.

Because Conclave applies optimizations that move operations like aggregations and filters out of MPC, the inputs to the MPC operation after Conclave's query rewriting may not be the same as the inputs to the original (un-optimized) query. Depending on the query, this change might either improve or harm security. To assess the impact, Conclave determines if the cardinalities of input relations are data-independent (as is the case in the market concentration query). If so, the transformation is non-leaky; if not, then Conclave provides all parties with the option to decline the performance optimizations and revert to the original query execution.

Optionally, Conclave includes several *hybrid operators* that further speed up aggregations and joins (§5.3). These additional optimizations rely on an understanding of *de facto* trust relationships between parties to introduce deliberate "leakage" that may be acceptable in some situations. Importantly, Conclave applies value-leaking optimizations *only* if users supply explicit input annotations that permit deriving an authorization (§4.3).

For instance, the performance of joins and aggregations can improve significantly if Conclave can leak their key columns (but no other columns) to a *semi-trusted party* (STP), who can then catalyze the computation by performing otherwise expensive operations outside of MPC. In this case, the STP learns the authorized column and all parties learn the sizes of all inputs and outputs to the join or aggregation relation; otherwise, MPC's traditional security guarantee continues to hold. A more formal description of Conclave's security guarantees can be found in §7.1.

4 Specifying Conclave queries

Conclave is a *query compiler* that transforms a relational query into a data processing *workflow*. It is similar to plaintext-only big data query compilers (*e.g.*, Hive [53], Pig [46], or Scope [15]) and "workflow managers", like Apache Oozie [32] or Musketeer [25]. Like these systems, Conclave transforms the query into a directed acyclic graph (DAG) of relational operators, and executes this DAG on several *backend* systems. Unlike prior query compilers, however, Conclave rewrites the query to improve performance while taking care to preserve MPC's security guarantees.

```
1  import conclave as cc
2  pA, pB, pC = cc.Party("mpc.ftc.gov"), \
3       cc.Party("mpc.a.com"), cc.Party("mpc.b.cash")
4  demo_schema = [Column("ssn", cc.INT, trust=[]),
5                 Column("zip", cc.INT, trust=[])]
6  demographics = cc.newTable(demo_schema, at=pA)
7  # banks trust the regulator to compute on SSNs
8  bank_schema = [Column("ssn", cc.INT, trust=[pA]),
9                 Column("score", cc.INT, trust=[])]
10 scores1 = cc.newTable(bank_schema, at=pB)
11 scores2 = cc.newTable(bank_schema, at=pC)
12 scores = cc.concat([scores1, scores2])
13 # query to compute average credit score by ZIP
14 joined = demographics.join(scores, left=["ssn"],
15                            right=["ssn"])
16 by_zip = joined.aggregate("count", cc.COUNT,
17                                group=["zip"])
18 total_sc = joined.aggregate("total", cc.SUM,
19                              group=["zip"])
20 avg_scores = \
21   total_sc.join(by_zip, left=["zip"], right=["zip"])
22           .divide("avg_score", "total", by="count")
23 # regulator gets the average credit score by ZIP
24 avg_scores.writeToCSV(to=[pA])
```

Listing 1. Credit card regulation query in Conclave's LINQ-style frontend with input relations locations (lines 6, 10–11), and an optional trust annotation (line 8).

4.1 Assumptions

Conclave assumes that analysts are comfortable writing relational queries using SQL or LINQ [43], that parties agree via out-of-band mechanisms on the query to run, and that all parties faithfully execute the protocol. Parties locally store input data with the schema expected by the query. Each party runs *(i)* a local Conclave agent, which communicates with the other parties and manages local and MPC jobs, *(ii)* at least one local data processing system (*e.g.*, Hadoop MapReduce, Spark, or Naiad), and *(iii)* a local MPC endpoint (*e.g.*, an Obliv-C or Sharemind node), all on private infrastructure.

4.2 Query specification

Conclave queries can be written in any way that compiles to a directed acyclic graph (DAG) of operators. Listings 1 and 2 show the credit ratings and market concentration queries from §2.1 in a DryadLINQ-like language [60].

Even though the input data to a Conclave query is distributed across multiple parties, Conclave largely abstracts this fact away from analysts. Concretely, the analysts specify the query's core as though it was a relational query on a single database stored at a trusted party (lines 14–22 of Listing 1 and lines 14–25 of Listing 2).

```
1  import conclave as cc
2  pA, pB, pC = cc.Party("mpc.a.com"), \
3        cc.Party("mpc.b.com"), cc.Party("mpc.c.org")
4  # 3 parties each contribute inputs with same schema
5  schema = [Column("companyID", cc.INT, trust=[]),
6            # ...
7            Column("price", cc.INT, trust=[])]
8  inputA = cc.newTable(schema, at=pA)
9  inputB = cc.newTable(schema, at=pB)
10 inputC = cc.newTable(schema, at=pC)
11 # create multi-party input relation
12 taxi_data = cc.concat([inputA, inputB, inputC])
13 # relational query
14 rev = taxi_data.project(["companyID", "price"])
15        .aggregate("local_rev", cc.SUM,
16                   group=["companyID"], over="price")
17        .project([0, "local_rev"])
18 market_size = rev.aggregate("total_rev", cc.SUM,
19                             over="local_rev")
20 share = rev.join(market_size, left=["companyID"],
21                  right=["companyID"])
22             .divide("m_share", "local_rev",
23                     by="total_rev")
24 hhi = share.multiply(share, "ms_squared", "m_share")
25           .aggregate("hhi", cc.SUM, on="ms_squared")
26 # finally, party A gets the resulting HHI value
27 hhi.writeToCSV(to=[pA])
```

Listing 2. Market concentration query in Conclave's LINQ-style frontend. Note the owner annotations on the input tables (lines 8–10) and the final result (line 27).

The only difference from a relational query is that Conclave also requires identifying at which input relations are located (lines 6–11 and 5–10). Each input relation has a specified "owner", *viz.*, the party storing it. This information helps Conclave *(i)* locate the data, and *(ii)* detect where operations combine data across parties. By combining per-party input relations using a duplicate-preserving set union operator (`concat`, lines 12 and 12), analysts can create compound relations across parties and use them in the query. In addition to inputs, analysts also annotate each output relation with one or more recipient parties. These parties end up storing the cleartext result of executing the query (lines 24 and 27).

Conclave's high-level, declarative query specification contrasts with existing MPC frameworks, which usually provide Turing complete DSLs (*e.g.*, SecreC [33] or Obliv-C [62]). While they are expressive, such interfaces are often unfamiliar to data analysts and require fine-grained security annotations of intermediate variables.

4.3 Optional trust annotations

In addition to mandatory input relation location annotations, Conclave also supports optional light-weight *trust annotations* that help it apply further optimizations. These annotations specify parties who are authorized to learn specific values in specific input schema columns in the clear to compute more efficiently on them.

The intuition behind trust annotations is that the sensitivity of data within a relation often varies by column. Consider a relation that holds information about a company's branches: it may have public `address` and `zip` columns (as this information is readily available from public sources), but privately-owned columns `isFranchise` or `workforceUnionized`. Other columns may be private, but the owning party might be happy to reveal them to a specific *semi-trusted party* (STP), such as a government regulator. For example, in the credit card regulation query (Listing 1), the government regulator already holds demographic information organized by SSN, and the credit card companies may be willing to reveal the SSNs of their customers to the regulator (though not to the other parties, *i.e.*, their competitors). Hence, the parties agree to make the regulator a semi-trusted party for the `ssn` column of the credit card companies' customer relations (see Listing 1, line 8). Such selective revealing of columns where permissible can help Conclave avoid, or shrink, expensive MPC steps and substantially improves performance.

A trust annotation associates a column definition with a *trust set* of one or more parties. Any party in the trust set can be a semi-trusted party for computing on the annotated column. This party may obtain the cleartext data for this column and combine it — locally and in the clear — with public columns, other columns with an overlapping trust set, and columns it privately owns. A party storing an input relation is in the trust set for all its columns; as are recipients for an output relation. Finally, a public column is one that has all parties in its trust set.

5 Query compilation

The annotations on input and output relations enable Conclave to determine which parts of the query DAG must run under MPC. Conclave automates this reasoning to free the data analysts from manual labor and to avoid subtle mistakes. Conclave applies a combination of static analysis, query rewriting transformations, and partitioning heuristics. Its goal is to execute as many operators as possible outside of MPC, and to reduce data volume processed under MPC where possible, while maintaining security guarantees.

Conclave compiles a query in six stages (partly illustrated in Figure 2); all parties run these deterministically.

1. Conclave starts with a query plan consisting of a single, large MPC. First, it propagates input relation locations to intermediate relations to determine where data crosses party boundaries (§5.1).
2. Using this information, Conclave then rewrites the query into an equivalent query with fewer operators under MPC. This results in a DAG with a clique of inner operators under MPC, and with efficient cleartext operators at the roots and leaves (§5.2).
3. Conclave then propagates the trust annotations from input relations through the DAG, and combines them

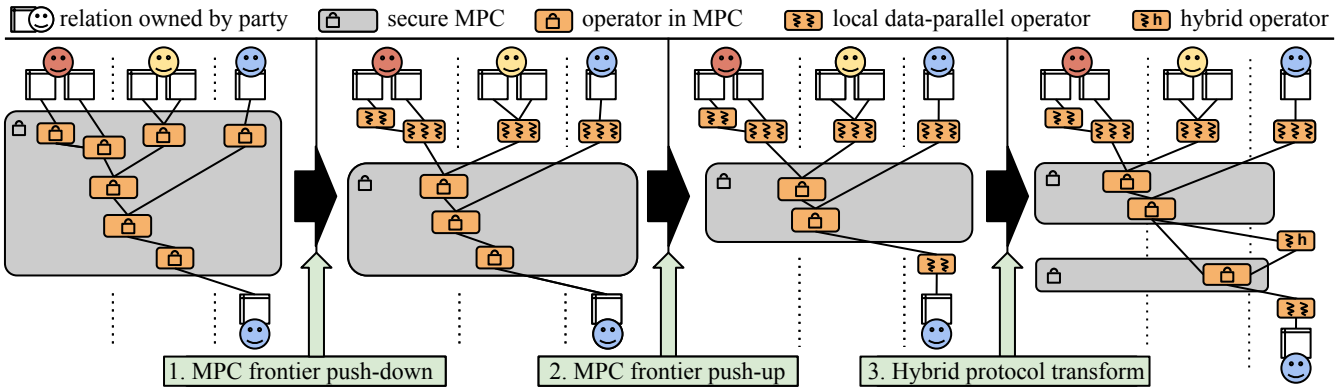


Figure 2. Conclave minimizes the work under MPC by: *(i)* pushing the MPC frontier down and locally preprocessing where possible; *(ii)* pushing the MPC frontier up from the outputs, processing reversible operators in the clear at the receiving party; and *(iii)* inserting special "hybrid" operators that implement efficient hybrid MPC-cleartext protocols.

according to inference rules in order to determine when parts of operators can run outside MPC.

4. Subsequently, and propagated trust annotations permitting, Conclave splits the monolithic inner MPC into several smaller MPCs and local steps by adding hybrid MPC-cleartext operators in place of operators that can run partially outside MPC (§5.3).
5. Conclave further minimizes the use of expensive oblivious sub-protocols, such as sorts, by moving these operations into local processing or replacing them with cheaper equivalents if possible (§5.4).
6. Finally, Conclave partitions the query by splitting the DAG at each transition between local and MPC operations, generates code for the resulting sub-DAGs, and executes them on the respective backends.

Note that all transformations that Conclave applies improve end-to-end query runtime, but they do *not* strictly reduce the overall work. In fact, they may create *additional* processing work that parties must do locally, or they may reduce the efficiency of local cleartext processing in exchange for doing less work under MPC. For example, a join between an intermediate relation and a public relation (*e.g.*, a relation mapping ZIP codes to geolocations) might process fewer records if it runs late in the query (*e.g.*, after filters and aggregations). Conventional query optimizers would apply filters before joins, but this may force the join into MPC, which is much slower than a local joins of the input data against the public relation. Hence, doing the join several times (once per party) and against more rows actually speeds up the query.

5.1 Propagating annotations

The input and output relation annotations give Conclave information about the roots and leaves of the DAG. Conclave propagates this information through the DAG to infer the execution constraints on its operators.

In a **first pass**, Conclave propagates input locations down the DAG in a topological order traversal, and propagates output locations back up the graph in a reverse topological order traversal. At each intermediate operator, the propagation derives the *owner* of its output relation. A party "owns" a relation if it can derive it locally given only its own data; input relations are owned by the party that stores them. The output relation of any operator that combines data across parties, has no individual owner: no single party can compute it. Operators producing output relations with no owner *must* run under MPC.

Conclave propagates relation ownership along edges using inference rules based on operators' input count. The output of a unary (*i.e.*, single-input) operator inherits the ownership of its input relation directly. The owner of the output relation of a multi-input operator depends on ownership of its input relations. If all input relations have the same owner, the ownership propagates to the output relation; if they have different owners, the output relation has no unique owner. This process captures the fact that any operator which combines data held by different parties produces joint data that must be protected.

In a **second pass**, Conclave propagates trust annotations from input relations in topological order, and combines them using similar rules. Specifically, unary operators propagate their input's trust annotation to all outputs, while multi-input operators intersect their input's trust annotations to determine their outputs. This captures the fact that Conclave can only use hybrid operators to run an MPC with the aid of a semi-trusted party if *all* parties supplying inputs to the MPC trust that semi-trusted party.

5.2 Finding the MPC frontier

Conclave starts planning the query with the entire DAG in a single, large MPC. It then pulls operators that can run on local cleartext data out of MPC, and splits other operators into local pre-processing operators and a smaller MPC step. These transformations push the *MPC frontier* — *viz.*, the

boundary between MPC operators and local cleartext operators — deeper into the DAG, where a clique of operators remains under MPC.

MPC frontier push-down. Conclave pushes the MPC frontier down the DAG as far as possible while preserving correctness and security guarantees, starting from the input relations. After the ownership propagation pass, each relation is either *(i)* a *singleton relation* with a unique owner; or *(ii)* a *partitioned relation* without an owner. In a partitioned relation, multiple parties hold a subset (*i.e.*, partition) of the relation. Conclave traverses the DAG from each singleton input relation and pulls operators out of MPC until it encounters an operator with a partitioned output, which it must process under MPC.

Queries often combine inputs from multiple parties into a single, partitioned relation via a `concat` operator. This creates a "virtual" input relation that contains data from all parties (*e.g.*, `scores` on line 12 of Listing 1, and `taxi_data` on line 12 of Listing 2). While convenient, this forces Conclave to enter MPC early, since the output of a `concat` operator is a partitioned relation. To avoid this, Conclave pushes `concat` operators down past any operators that are distributive over input partitions: *i.e.*, for operator `op` and relations R_{pA} to R_{pN} owned by pA to pN,

$$\mathsf{op}(R_{pA} \mid \ldots \mid R_{pN}) \equiv \mathsf{op}(R_{pA}) \mid \ldots \mid \mathsf{op}(R_{pN}).$$

For example, the projection over `taxi_data` in the market concentration query (Listing 2, line 13) is distributive, as applying it locally to `inputA`, `inputB`, and `inputC` produces the same result and leaks no more information than running it under MPC. Consequently, Conclave can push the MPC frontier further down.

Other operators, however, do not trivially distribute over the inputs of a `concat`. For example, splitting an aggregation that groups a partitioned relation by key and applying it to the original singleton relations requires another, secondary aggregation to produce identical results. Conclave inserts such a secondary aggregation when possible.[1] While the secondary aggregation must remain under MPC, Conclave can pull the local pre-aggregations out of MPC and significantly reduce the MPC's input data size. Moreover, Conclave can push the operator clique of `concat` and the secondary aggregation (which now forms the MPC frontier) past other distributive operators. In the market concentration query, Conclave pushes the MPC frontier to right above the `market_size` relation, at which point the data amount to only few integers per party.

MPC frontier push-up. In certain cases, Conclave can also push the MPC boundary up from the output relations (*i.e.*, the DAG's leaves). Some relational operators are *reversible*, *i.e.*, given their output, it is possible to reconstruct the input without additional information. For example, multiplication

[1]This transformation leaks the count of distinct group-by column values each party contributes; §7.1 discusses its security implications.

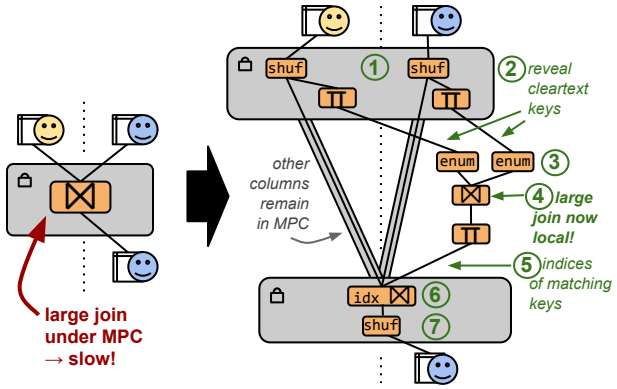


Figure 3. The hybrid join transformation turns an MPC join into an oblivious shuffle followed by a cleartext enumeration, a join, and a projection outside MPC (at the STP), and finally performs an inexpensive oblivious indexing protocol to reconstruct the join result under MPC.

of column values by a fixed non-zero scalar has this property. For a reversible operator, the output of the operation fundamentally *leaks* its input, and hence it need not run under MPC. Instead, Conclave reveals the reversible operator's input relation to the final recipients for local cleartext multiplication, even if the input has a different owner.

Conclave's MPC push-up pass starts at output relations and lifts the MPC frontier through reversible operators. Key examples are arithmetic operations and re-ordering projections; and while aggregations are generally not reversible, special cases are. If `count` occurs as a leaf operator, it inherently reveals the group-by key frequencies, and Conclave can rewrite it to an MPC projection and a cleartext count. The projection removes all columns apart from the group-by columns (under MPC), and the recipients count the keys in the clear. As projections are more scalable under MPC than aggregations (§2.3), this improves performance.

5.3 Hybrid operators

In this rewrite pass, Conclave splits work-intensive operators, *viz.*, joins and aggregations, into *hybrid operators*. Hybrid operators outsource expensive portions of an operator to a semi-trusted party (STP) by revealing some input columns to the STP. Hybrid operator execution thus involves local computation at the STP and MPC steps across all parties. Conclave *only* applies this transformation if the query's trust annotations relax input columns' privacy constraints.

In the context of hybrid operators, a party is either the STP, or a regular, untrusted party. Conclave exposes the plaintext *values* of some columns to the STP, and the size of the result (*i.e.*, row count) to all parties; otherwise, it maintains full MPC guarantees for these columns towards the untrusted parties, and towards all parties for all other columns.

Conclave currently supports three hybrid operators: a *hybrid join*, a *public join*, and a *hybrid aggregation*.

Hybrid join. Conclave can transform a regular MPC join into a hybrid join if the key columns of *both* sides of the join have intersecting trust sets, *i.e.*, they share an STP. This STP learns the key columns on *both* sides of the join and computes, in the clear, which keys match. The STP hence determines which rows in the secret-shared relations are in the join result without learning any other column values.

The protocol proceeds as follows (Figure 3):

1. The parties obliviously shuffle the input relations under MPC.
2. The parties project away all columns other than the join key columns, and reveal the resulting key-column-only relations to the STP.
3. For each key-column relation, the STP enumerates the rows. The enumeration assigns a unique identifier to each input row, which Conclave later uses to link the joined results back to rows in the secret-shared, shuffled input relations still protected by MPC.
4. The STP performs a cleartext join on the enumerated key-column relations. This produces a set of rows that each contain the join key and two unique row identifiers for the left and right rows joined.
5. The STP projects away the join key, and secret-shares the resulting *index relation* to the untrusted parties.
6. Back under MPC, the parties perform an oblivious indexing protocol akin to the one by Laud [40] using the secret-shared index relation rows to construct the result of the join out of the shuffled input rows.
7. The parties obliviously reshuffle the result.

Step 2 of the protocol reveals the values of the key columns to the STP, but this exposure is authorized by the input relations' trust annotations. The oblivious shuffles in Step 1 and Step 7 prevent the STP from linking the information learned about the shuffled relations back to the input relations and therefore other parts of the query. The oblivious indexing protocol in Step 6 requires $\mathcal{O}((n+m)\log(n+m))$ non-linear operations, where n is the input size and m the result size, unlike the standard MPC join protocol, which requires $\mathcal{O}(n^2)$ non-linear operations (but assumes no STP).

Public join. Conclave can use the public join operator if the join key columns of *both* sides of an MPC join are public. This is the case if both columns' trust annotations include all parties, and hence *any* party may learn the key column values (though not other column values). The public join proceeds exactly as the hybrid join but without the oblivious indexing and shuffle steps: the parties send the join key columns to an STP (which can be any random party in this case); the STP enumerates and joins the rows; and the STP finally sends the joined index relation to all parties, who compute the joined result. Even though some MPC frameworks have built-in cleartext processing capabilities, Conclave uses this approach since it allows the use of a data-parallel framework (*e.g.*, Spark) for local work.

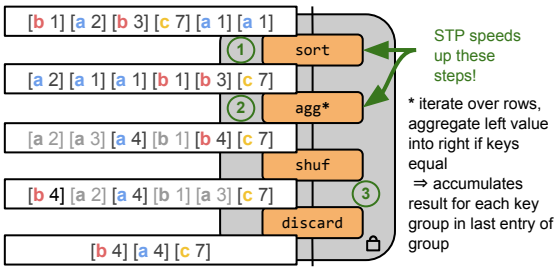


Figure 4. A hybrid aggregation outsources expensive steps of an MPC aggregation [36] to the STP. In the original protocol, the parties arrange the rows into key-groups by sorting on the key, obliviously accumulate the aggregate for each key-group into its last entry, and discard the other entries. In the hybrid version, the STP can perform the sort in the clear and assist the other parties in the accumulation step.

Hybrid aggregation. Conclave can transform an MPC aggregation into a hybrid aggregation if the trust set on the group-by column contains an STP, *i.e.*, there is an STP authorized to learn the group-by column values.

Conclave's hybrid aggregation protocol adapts the sorting-based MPC protocol by Jónsson *et al.* [36], whose core proceeds as follows (Figure 4):

1. The parties obliviously sort the input on the group-by column, thus arranging it into key groups.
2. The parties scan over the result, and at each entry compute, under MPC, an *equality flag*, *i.e.*, whether the entry's key equals the previous entry's. If it does, they aggregate the previous value into the current, which can be expressed via a circuit of additions and multiplications (and thus done obliviously). This accumulates the aggregate for each key group into the group's last entry. At each entry, the parties also store the secret equality flag for the last comparison; the flag is set unless the entry is the last one in the group.
3. The parties shuffle the result and reveal the equality flags; they discard all entries with the flag set.

Conclave transforms the above protocol into a hybrid aggregation by outsourcing Steps 1 and 2 to the STP. In particular, the parties obliviously shuffle the input and reveal the group-by column values of the result to the STP. The STP performs the sort in the clear and sends to the parties the ordering of the sorted rows (this does not leak any additional information to the parties, since any linking to input values was broken by the oblivious shuffle). The STP furthermore computes the equality flags of Step 2 and secret-shares those with the other parties. The parties use the equality flags and ordering information to complete the protocol.

The hybrid aggregation improves asymptotically over the regular MPC protocol: the oblivious sorting step of the original protocol is based on a sorting network and requires $\mathcal{O}(n\log^2 n)$ oblivious comparisons; in contrast, the hybrid

aggregation performs the sort in the clear and only needs an oblivious shuffle which can be realized with $\mathcal{O}(n \log n)$ multiplications (which are furthermore more efficient than comparisons under secret-sharing based MPC). However, the hybrid aggregation leaks the size of the result to all untrusted parties; we describe such leakage further in §7.1.

5.4 Reducing oblivious operations

Conclave's relational operator implementations rely on MPC "sub-protocols" such as oblivious sorts and shuffles (*e.g.*, the aggregation in Figure 4 uses oblivious sorts).

These operations, especially sorts, are expensive under MPC, since they must remain control-flow agnostic. Conclave can minimize their use by moving them up through the DAG into cleartext processing. Sort operations can move across any order-preserving operator (such as a filter). If a sort reaches a relation in which the sort order column is public, Conclave can sort the rows in the clear. Likewise, if a sort reaches a relation owned by a single party, the party can perform the sort in the clear. While `concat` operations are not order-preserving, Conclave can still push the sort through the `concat` by inserting after it a merge operation. The merge takes several sorted relations and obliviously merges them, which is cheaper than obliviously sorting the entire data. As Conclave pushes sort operations up through the DAG, it can also eliminate redundant sorts.

These transformations are another example of a key idea in Conclave: doing more work locally (*e.g.*, redundantly sorting duplicate rows later eliminated) can yield lower execution times than doing reduced work under MPC.

6 Implementation

We implemented Conclave as a query compiler architected similarly to "big data" workflow managers like Musketeer [25]. Our prototype consists of 5,000 lines of Python, and currently integrates sequential Python and data-parallel Spark [61] as cleartext backends, and Sharemind [11] and Obliv-C [62] as MPC backends. As Conclave's interfaces are generic, adding other backends requires modest implementation effort.

Our prototype supports table schema definitions, relational operators (join, aggregate, project, filter) as well as enumeration, arithmetic on columns and scalars. This captures many practical queries: Conclave's current query support is, for example, sufficient to express 88% of 8M sensitive real-world queries at Uber [35].

We implemented the same standard MPC algorithms for joins (a Cartesian product-based approach) and aggregations [36] in both Sharemind and Obliv-C. Our prototype does not yet implement interactive authorization of push-down operations (§3.2); it is easy to add.

7 Evaluation

We evaluate Conclave using our motivating queries (§2) as well as microbenchmarks. We seek to answer:

1. What security guarantees can Conclave make? (§7.1)
2. How does the runtime of realistic queries running on Conclave scale as data size grows? (§7.2, §7.4)
3. What impact on performance do Conclave's trust annotations and hybrid operators have? (§7.3)
4. How does Conclave compare to SMCQL [3], a state-of-the-art query processor for MPC? (§7.5)

Setup. Unless otherwise specified, we run our experiments with three parties; each party runs a four-node cluster that consists of three Spark VMs and one Sharemind VM. The Spark VMs have 2 vCPUs (2.4 GHz) and 4 GB RAM, and run Ubuntu 14.04 with Spark 2.2 and Hadoop 2.6. The Sharemind VM has 4 vCPUs (2.4 GHz) and 8 GB RAM, and runs Debian Squeeze and Sharemind 2016.12.

Metrics. All our graphs increase the data size on the x-axis by five to eight orders of magnitude, and plot query runtime on the y-axis. Less is better in all graphs, and we use a $\log_{10}$-scale x-axis to be able to show the scalability limits of different systems on the same graph.

7.1 Security guarantees

In this section, we formally state the assumptions that Conclave requires and the security implications of the transformations it performs. Hence, this section formalizes §3.2.

Adversary corruptions. A participant in the computation may either be a regular party or an STP; we assume that only a single STP may be involved in a query. Conclave operates under the assumption that an adversary may either corrupt the STP *or* a permissible subset of regular parties (as determined by the concrete MPC backend); however, the adversary cannot corrupt both regular parties and the STP.

Guarantees. We consider the security guarantees provided by Conclave in three scenarios: *(i)* it does not optimize queries at all, *(ii)* it performs only query transformations, and *(iii)* it applies query transformations and hybrid operators.

Without any transformations or optimizations, Conclave inherits the security guarantees provided by its MPC backend. Define the *view* of the adversary as the set of messages it observes over the network together with the state of all parties it corrupts. Secure multi-party computation guarantees that its view is *simulatable* simply given the corrupted parties' inputs, the overall output, and the size of uncorrupted parties' inputs. Formally, a secure MPC protocol π for function f guarantees that for all adversaries $\mathcal{A}$ corrupting a permissible set of parties $\mathcal{C}$, there exists a simulator $\mathcal{S}$ such that for any set of private inputs x_i of lengths $\ell_i = |x_i|$ on which π yields output out^π at the receiving party, the following two ensembles are computationally indistinguishable:

$$\langle \text{view}^\pi_{\mathcal{A}}, \vec{x}_{\mathcal{C}}, \vec{\ell}, \text{out}^\pi \rangle \equiv \langle \mathcal{S}(\vec{x}_{\mathcal{C}}, \vec{\ell}), \vec{x}_{\mathcal{C}}, \vec{\ell}, f(\vec{x}) \rangle. \quad (1)$$

If $\mathcal{C}$ includes the receiving party, then $\mathcal{S}$ is also given $f(\vec{x})$.

Conclave's push-down and push-up query transformations (§5.2) leave intact MPC's correctness and security guarantee. By construction, Conclave's push-up is simulatable

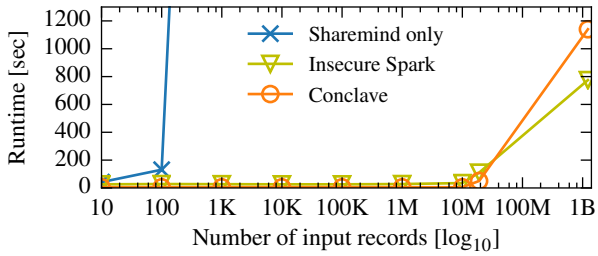
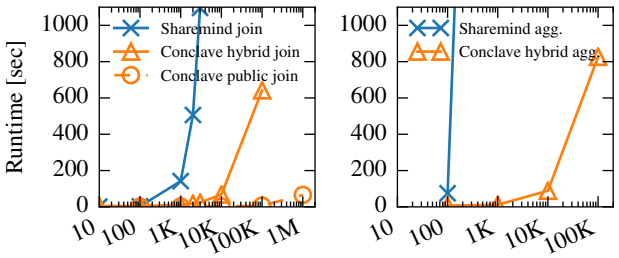


Figure 5. Conclave runs the market concentration query in <20 minutes for 1B input records; under Sharemind MPC, the query cannot scale to > 100 input records.



(a) Hybrid join. **(b)** Hybrid aggregation.

Figure 6. Conclave's hybrid operators help scale joins and aggregations to large data by combining MPC and cleartext compute; Sharemind alone only handles 2,000 (join) and 100 (agg.) records within tens of minutes.

from the output, and thus it has no impact on security. However, push-downs do have security implications because they change the lengths of inputs to the MPC backend, and therefore they influence the type of information that is simulatable in (1); *e.g.*, splitting an aggregation into a local step and an MPC step results in the parties learning the count of distinct group-by column values contributed by each party, as opposed to the total number of records per party. For this reason, Conclave requires the consent of parties to make any push-down transformations that result in data-dependent cardinalities at the input to secure MPC.

Conclave's hybrid operators (§5.3) introduce two new types of leakage: the STP learns authorized columns and all parties (regular and STP) observe the cardinalities of inputs and outputs to an intermediate relation. Nevertheless, we claim that Conclave achieves MPC's standard security guarantee (1) with the aforementioned leakage provided to the adversary and simulator. In particular, the STP's knowledge of a key column composes gracefully with push-down and push-up transformations; that is, any relation processed by a single party in the clear is trivially simulatable by that party. This claim can be proved by partitioning the computation into three distinct MPC stages before, during, and after the hybrid operator and using sequential composition of simulation-based security definitions; due to space constraints, we defer details to an extended technical report.

7.2 Market concentration query

The market concentration query computes the Herfindahl Hirschman Index (HHI) [30] over the market shares of several vehicle-for-hire (VFH) companies, whose sales books we model using six years of public NYC taxi trip fare information [51]. We randomly divide the trips across three imaginary VFH companies and filter out any trips with a zero fare. This results in a total of 1.3 billion trips across all parties. We subsample different numbers of rows from the input data sets, and measure the end-to-end query execution time for different input sizes.

Figure 5 shows the results over eight orders of magnitude in input size. It is evident that this query scales poorly when run entirely under MPC in Sharemind: at 1,000 input rows, Sharemind already takes 2.5 hours to complete it. The query contains both an aggregation and a self-join, and the runtime of these expensive operators (cf. §2.3, Fig. 1) dominates all others. In particular, the poor scalability of the Sharemind aggregation leads to a rapid increase in execution time at 100 records. Conclave, by contrast, scales roughly linearly in the size of the input data, since it pushes the MPC frontier past aggregations for the per-party revenue. All data-intensive processing happens outside MPC in local Spark jobs, and only a few records enter the final MPC, which consequently completes quickly. The same query executed insecurely on the joint data using Spark performs similarly (slightly slower up to 10M as it runs one job, not three parallel jobs); at 1.3B records, Conclave is slower because the MPC step dominates.

7.3 Hybrid operator performance

The market concentration query benefits from Conclave pushing down the MPC frontier and splitting the initial aggregation. Conclave also transforms queries to use hybrid MPC-cleartext protocols if a semi-trusted party — *e.g.*, a government regulator — participates (§5.3). We measure the impact of these hybrid protocols with synthetic input data ranging from 100 to 100k input records and run queries consisting of only a single join or aggregation. We annotate the input relations' columns with semi-trusted parties, allowing Conclave to apply hybrid operator transformations, and measure query runtimes. We expect Conclave's hybrid operators to reduce query runtimes, even though the rewritten query contains additional operators.

Figure 6 confirms this. Without an STP, Conclave must run the query entirely under MPC, which exhibits performance similar to earlier benchmarks (§2.3). If using a hybrid operator, however, scalability and runtimes improve substantially: a hybrid join on 100k records takes just over ten minutes. The hybrid join improves asymptotically over the MPC join; it replaces $\mathcal{O}(n^2)$ non-linear operations under MPC with oblivious shuffles and indexing protocols, which require $\mathcal{O}((n+m)\log(n+m))$ non-linear operations, where n is the input size and m the output size of the join.

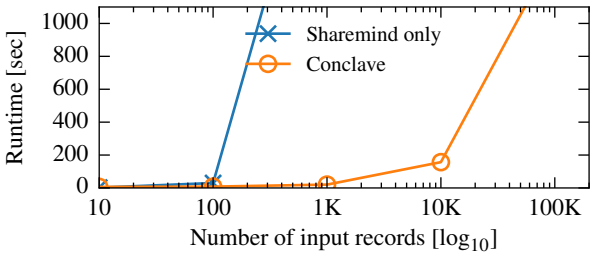


Figure 7. Conclave scales to two orders of magnitude more data on the credit card regulation query than pure Sharemind due to its hybrid join and aggregation.

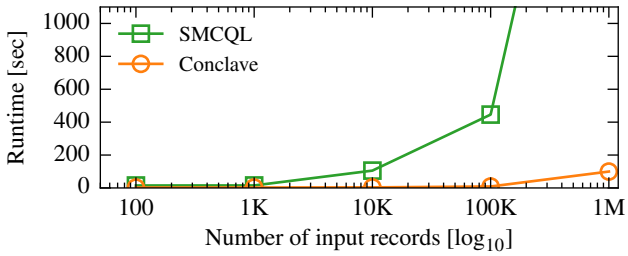


Figure 8. Conclave outperforms SMCQL on the *aspirin count* query [3, §2.2.1] as Conclave's public join lifts additional work out of MPC. SMCQL did not complete at 1M records.

The public variant of the join operator performs even better since it avoids the use of MPC altogether; the bottleneck for the public join is sending the input data to the party selected to perform the join operation.

The hybrid aggregation demonstrates similar speedups. This boost in performance is due to an asymptotic improvement: an oblivious shuffle with $\mathcal{O}(n \log n)$ complexity replaces a sorting-network requiring $\mathcal{O}(n \log^2 n)$ comparisons. In addition to the asymptotic improvement, the hybrid operator also avoids performing oblivious comparison and equality operations, which are slow in secret-sharing MPC.

7.4 Credit card regulation query

Conclave's hybrid operators can offer substantial benefits for the common case of queries whose performance is dominated by aggregations and joins. The credit card regulation query is an example: it first joins the regulator's demographic information with the credit scores, and then computes an aggregate (*viz.*, the average score grouped by ZIP code). The credit card companies trust the regulator, but not their competitors, with the SSNs of their customers, and the regulator wishes to keep the mapping from SSNs to ZIP codes private. Hence, Conclave can apply both the hybrid join and the hybrid aggregation operator transformations to this query. Even though these optimizations increase query complexity, we expect them to reduce runtime, as both the join and the aggregation work can now happen outside MPC. Here, the Sharemind baseline uses a join implementation that leaks output size, matching Conclave's hybrid operators' leakage.

Figure 7 confirms this. Running the query entirely under MPC in Sharemind fails to scale beyond 1,000 customers per company; at 10k customers, the query takes 139 minutes to complete. With Conclave's hybrid operators, however, the query only takes a few minutes even at large scales. The experiment highlights that hybrid operators are crucial to obtaining good performance for this query: its the first operator is a join, so Conclave cannot push the MPC frontier down. Hence, without the hybrid operators, Conclave would have to run the entire query under MPC.

7.5 Comparison with SMCQL

Finally, we compare Conclave with the most similar state-of-the-art system, SMCQL [3]. SMCQL and Conclave make different, complementary optimizations to speed up MPC on for relational queries. To compare, we configure Conclave to be as similar to SMCQL as possible. We disable the MPC frontier pushdown past local filters over private columns (to match SMCQL's security guarantee), and manually implement SMCQL's "sliced join" optimization (which composes with Conclave's optimizations). Since SMCQL currently uses the ObliVM [42] garbled-circuit framework for its MPC backend, it only supports two-party computations and sees slower MPC operations than Conclave — which uses secret-sharing Sharemind — on large data. This difference is not fundamental, however: SMCQL could generate code for Sharemind. However, we expect Conclave's extra optimization to help it consistently outperform SMCQL in this experiment.

Conclave and SMCQL make comparable security guarantees here. Both protect against a passive adversary, although at different MPC corruption thresholds: SMCQL's ObliVM [42] backend requires two parties and tolerates one corrupted party, while Conclave's Sharemind backend requires three parties and tolerates one corrupt party. In Conclave, two parties provide input and the third participates in the MPC without inputs. SMCQL's also has a third, "honest broker" party, which doesn't participate in the MPC. Because SMCQL requires more memory, this experiment uses larger VMs with 32 GB RAM and 8 vCPUs.

Setup. We benchmark the *aspirin count* query from the SMCQL paper. This query measures the efficiency of Aspirin for preventing heart attacks by "counting how many heart disease sufferers who were prescribed Aspirin" [3, §2.2.1]. The query consists of a join on public columns (anonymized patient IDs), followed by filters on private attributes (the patient's diagnosis and prescribed medication), and a distinct count aggregation. It has two input relations, `diagnoses` and `medications` from the HealthLNK dataset [29]. These inputs to the join are partitioned across two parties (different hospitals), *i.e.*, each party holds part of the `diagnoses` table and part of the `medications` table.

SMCQL's "slicing" partitions the data on the public patient ID column. Slices containing patient IDs that only exist at one party are processed locally at that party, while slices with keys extant at both parties must be processed under MPC. For the experiment, we combine SMCQL's slicing with Conclave's public join. Slicing makes the query sensitive to the patient ID distributions, since their overlap across the parties determines how many rows enter MPC. The SMCQL paper benchmarks overlaps of one in 50 [3, §7.2]; we hence generate input data with a 2% overlap between the parties' uniformly-random patient IDs (using up to 500k patient IDs).[2] We increase the number of records in each input relation — *i.e.*, the total input is four times the x-axis value — and measure query runtime.

Results. Figure 8 shows that Conclave consistently outperforms SMCQL. The difference is most pronounced for larger queries; at 100k rows Conclave completes in 8.9 seconds, while SMCQL takes 7.5 minutes (50× slower than Conclave). Conclave completes the 1M query in under two minutes; SMCQL runs out of memory error after ≈1 hour.[3] This improvement stems primarily from two of Conclave's optimizations, the public join and the sort push-up. The public join combined with slicing allows Conclave to pre-compute the initial join in the clear, sending only rows with have patient IDs extant at both parties into the MPC. By contrast, SMCQL only slices the data on patient IDs, and cannot carry out the entire join in the clear. For each private slice, SMCQL still performs the join and the subsequent operations obliviously, which has quadratic cost in the size of the slice.

Sort push-up allows Conclave to avoid an expensive oblivious sort step otherwise required for the distinct count by moving it up into clear-text processing. Conclave can pre-sort the data in the clear since the sort is by a public column, and since all operations under MPC are order-preserving. This reduces the complexity of the MPC from $\mathcal{O}(n \log n)$ to $\mathcal{O}(n)$. SMCQL could likewise benefit from this optimization.

8 Related Work

We now highlight elements of Conclave that relate to other approaches to building privacy-protecting systems. We omit prior work in MPC algorithms, frameworks, and deployments already discussed in §2. Instead, we survey efforts that have made innovations in "mixed mode" operations, query rewriting, and query scalability for MPC.

MPC with mixed mode operation. Wysteria [50] performs mixed mode computations that move between MPC and local work. Wysteria programs are written in a DSL that creates the programmer illusion of a single thread of control, but require familiarity with said DSL. Unlike Wysteria, Conclave programs are standard relational queries, and Conclave supports distributed backend systems for parallel processing: it connects with data-parallel frameworks like Spark.

Query rewriting. There are several efforts to optimize MPC via query rewriting, like Conclave does, at different levels of abstraction. Kerschbaum [39] operates at the circuit level, transforming a manually assembled circuit into a different one with faster execution under MPC (*e.g.*, using the distributive law to reduce the number of multiplications). Other systems perform query rewriting at the relational algebra level, such as SMCQL [3] and Opaque [63]. SMCQL, like Conclave, uses column-level annotations, but differentiates only between public and private columns. However, SMCQL shares some optimization, such as parts of the MPC frontier push-down, with Conclave, and supports other complementary optimizations (slicing and secure semi-joins). Conclave's annotations are more expressive, and Conclave's hybrid protocols allow for additional performance improvements. Opaque, by contrast, runs most computation in the clear inside a protected Intel SGX enclave; its query rewriting focuses on reducing the number of oblivious sorts required in distributed computation across multiple SGX machines.

Protected databases and scalability. The protected database community has produced decades of research on scaling secure query execution to the gigabyte-to-terabyte range [13; 22; 27]. This includes work on optimizations for boolean keyword search [21; 48], as well as large, general subsets of relational algebra [37]. These works largely target querying a single protected database, as opposed to Conclave's distributed scenario. Investigations into the scalability of secure MPC often involve laborious hand-optimization by groups of cryptographers on specific queries like set intersection [31; 38], linear algebra [24], or matching [19].

Inference and privacy. MPC protects sensitive state during computation but provides no restriction on the ability to *infer* sensitive inputs from the provided outputs. Differential privacy (DP) [20] provably ensures that the output of an analysis reveals nothing about any individual input, but often uses a trusted curator to perform the analysis. Several prior systems have combined MPC and DP to avoid computing and output parties jointly reconstructing sensitive input data. DJoin [44] does so for SQL-style relational operations (with query rewriting, but without Conclave's automation and hybrid protocols), DStress [47] does so for graph analysis, and He *et al.* [28] do so for private record linkage. Conclave does not currently leverage DP in any way, but its query rewriting and code generation components are both sufficiently extensible to support addition of DP.

9 Conclusion and future work

Conclave speeds up secure MPCs on "big data" by rewriting queries to minimize expensive processing under MPC.

[2] We use synthetic data as we were unable to obtain the HealthLNK dataset from the SMCQL authors in time for submission due to a missing approval.
[3] Bater *et al.* also benchmarked *aspirin count* for larger inputs of 42M `diagnoses` and 23M `medications`, but used eight servers to parallelize sliced MPCs, taking 23 hours to complete the query [3, §7.3]. Conclave can likewise run additional Sharemind servers, but we lacked the resources to do so. We expect Conclave to still outperform SMCQL on the full data set.

Conclave runs queries that would have been impractical with previous MPC frameworks, or would have required domain-specific knowledge, in minutes.

In the future, we plan to integrate other MPC backends into Conclave, and to make Conclave choose the most performant MPC protocol for a query. We are also interested in whether verifiable computation techniques can be combined with Conclave, and whether Conclave can use adaptive padding to avoid or reduce hybrid operators' size leakage (*e.g.*, by padding to a known total number of extant keys).

Conclave will be available as open-source software.

References

[1] T. Araki, A. Barak, J. Furukawa, T. Lichter, Y. Lindell, A. Nof, K. Ohara, et al. "Optimized Honest-Majority MPC for Malicious Adversaries – Breaking the 1 Billion-Gate Per Second Barrier". In: *Proceedings of the 38th IEEE Symposium on Security and Privacy (SP)*. May 2017, pp. 843–862.

[2] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. "High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. 2016, pp. 805–817.

[3] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel Kho, and Jennie Rogers. "SMCQL: Secure Querying for Federated Databases". In: *Proceedings of the VLDB Endowment* 10.6 (Feb. 2017), pp. 673–684.

[4] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization". In: *Proceedings of the 11th Annual International Cryptology Conference (CRYPTO)*. Santa Barbara, California, USA, Aug. 1991, pp. 420–432.

[5] Donald Beaver. "Precomputing Oblivious Transfer". In: *Proceedings of the 15th Annual International Cryptology Conference (CRYPTO)*. Santa Barbara, California, USA, Aug. 1995, pp. 97–109.

[6] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)". In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (TC)*. Chicago, Illinois, USA: ACM, May 1988, pp. 1–10.

[7] Azer Bestavros, Andrei Lapets, and Mayank Varia. "User-centric distributed solutions for privacy-preserving analytics". In: *Commun. ACM* 60.2 (2017), pp. 37–39.

[8] Dimitrios Bisias, Mark Flood, Andrew W. Lo, and Stavros Valavanis. "A Survey of Systemic Risk Analytics". In: *Annual Review of Financial Economics* 4.1 (2012), pp. 255–296. eprint: https://doi.org/10.1146/annurev-financial-110311-101754.

[9] Dan Bogdanov, Marko Jõemets, Sander Siim, and Meril Vaht. "How the Estonian Tax and Customs Board Evaluated a Tax Fraud Detection System Based on Secure Multi-party Computation". In: *Proceedings of the 19th International Conference on Financial Cryptography and Data Security*. San Juan, Puerto Rico, Jan. 2015, pp. 227–234.

[10] Dan Bogdanov, Liina Kamm, Baldur Kubo, Reimo Rebane, Ville Sokk, and Riivo Talviste. "Students and Taxes: a Privacy-Preserving Study Using Secure Computation". In: *PoPETs* 2016.3 (2016), 117?135.

[11] Dan Bogdanov, Sven Laur, and Jan Willemson. "Sharemind: A Framework for Fast Privacy-Preserving Computations". In: *Proceedings of the 13th European Symposium on Research in Computer Security*. Ed. by Sushil Jajodia and Javier Lopez. Vol. 5283. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2008, pp. 192–206.

[12] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, et al. "Financial Cryptography and Data Security". In: ed. by Roger Dingledine and Philippe Golle. Berlin, Heidelberg: Springer-Verlag, 2009. Chap. Secure Multiparty Computation Goes Live, pp. 325–343.

[13] Christoph Bösch, Pieter H. Hartel, Willem Jonker, and Andreas Peter. "A Survey of Provably Secure Searchable Encryption". In: *ACM Comput. Surv.* 47.2 (2014), 18:1–18:51.

[14] Elette Boyle, Kai-Min Chung, and Rafael Pass. "Large-Scale Secure Computation: Multi-party Computation for (Parallel) RAM Programs". In: *Proceedings of the 35th Annual International Cryptology Conference (CRYPTO)*. Santa Barbara, California, USA, Aug. 2015, pp. 742–762.

[15] Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. "SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets". In: *Proceedings of the VLDB Endowment* 1.2 (Aug. 2008), pp. 1265–1276.

[16] Nishanth Chandran, Wutichai Chongchitmate, Juan A. Garay, Shafi Goldwasser, Rafail Ostrovsky, and Vassilis Zikas. "The Hidden Graph Model: Communication Locality and Optimal Resiliency with Adaptive Faults". In: *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*. 2015, pp. 153–162.

[17] Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. "Quorums Quicken Queries: Efficient Asynchronous Secure Multiparty Computation". In: *Distributed Computing and Networking - 15th International Conference, ICDCN 2014, Coimbatore, India, January 4-7, 2014. Proceedings*. 2014, pp. 242–256.

[18] Varsha Dani, Valerie King, Mahnush Movahedi, Jared Saia, and Mahdi Zamani. "Secure multi-party computation in large networks". In: *Distributed Computing* 30.3 (2017), pp. 193–229.

[19] Jack Doerner, David Evans, and Abhi Shelat. "Secure Stable Matching at Scale". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1602–1613.

[20] Cynthia Dwork. "Differential Privacy: A Survey of Results". In: *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation*. TAMC'08. Xi'an, China: Springer-Verlag, 2008, pp. 1–19.

[21] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. "Rich Queries on Encrypted Data: Beyond Exact Matches". In: *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II*. 2015, pp. 123–145.

[22] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, et al. "SoK: Cryptographically Protected Database Search". In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. 2017, pp. 172–191.

[23] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. "High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority". In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*. 2017, pp. 225–255.

[24] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. "Privacy Preserving Distributed Linear Regression on High-Dimensional Data". In: *Proceedings on Privacy Enhancing Technologies*. Oct. 2017, pp. 345–364.

[25] Ionel Gog, Malte Schwarzkopf, Natacha Crooks, Matthew P. Grosvenor, Allen Clement, and Steven Hand. "Musketeer: all for one, one for all in data processing systems". In: *Proceedings of the 10th ACM European Conference on Computer Systems (EuroSys)*. Bordeaux, France, Apr. 2015.

[26] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (TC)*. New York City, New York, USA, 1987, pp. 218–229.

[27] Ariel Hamlin, Nabil Schear, Emily Shen, Mayank Varia, Sophia Yakoubov, and Arkady Yerukhimovich. "Cryptography for Big Data Security". In: *Big Data: Storage, Sharing, and Security*. Ed. by Fei Hu. Taylor & Francis LLC, CRC Press, 2016.

[28] Xi He, Ashwin Machanavajjhala, Cheryl J. Flynn, and Divesh Srivastava. "Scaling Private Record Linkage using Output Constrained Differential Privacy". In: *CoRR* abs/1702.00535 (2017). arXiv: 1702.00535.

[29] CHIP Center for Health Information Partnerships. *HealthLNK Data Repository (HDR)*. 2014. URL: http://www.healthinformationforall.org/healthlnk-overview/.

[30] Albert O. Hirschman. "The Paternity of an Index". In: *The American Economic Review* 54.5 (1964), pp. 761–762.

[31] Mihaela Ion, Ben Kreuter, Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, David Shanahan, et al. "Private Intersection-Sum Protocol with Applications to Attributing Aggregate Ad Conversions". In: *IACR Cryptology ePrint Archive* (2017).

[32] Mohammad Islam, Angelo K. Huang, Mohamed Battisha, Michelle Chiang, Santhosh Srinivasan, Craig Peters, Andreas Neumann, et al. "Oozie: Towards a Scalable Workflow Management System for Hadoop". In: *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies (SWEET)*. Scottsdale, Arizona, USA, 2012, 4:1–4:10.

[33] Roman Jagomägis. "SecreC: a privacy-aware programming language with applications in data mining". In: *Master's thesis, University of Tartu* (2010).

[34] Noah M. Johnson, Joseph P. Near, and Dawn Song. "Practical Differential Privacy for SQL Queries Using Elastic Sensitivity". In: *CoRR* abs/1706.09479 (2017). arXiv: 1706.09479.

[35] Noah M. Johnson, Joseph P. Near, and Dawn Xiaodong Song. "Practical Differential Privacy for SQL Queries Using Elastic Sensitivity". In: *CoRR* abs/1706.09479 (2017). arXiv: 1706.09479.

[36] Kristján Valur Jónsson, Gunnar Kreitz, and Misbah Uddin. "Secure multi-party sorting and applications". In: *Proceedings of the 9th International Conference on Applied Cryptography and Network Security (ACNS)*. Nerja (Malaga), Spain, June 2011.

[37] Seny Kamara and Tarik Moataz. "SQL on Structurally-Encrypted Databases". In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 453.

[38] Seny Kamara, Payman Mohassel, Mariana Raykova, and Saeed Sadeghian. "Scaling private set intersection to billion-element sets". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2014, pp. 195–215.

[39] Florian Kerschbaum. "Expression Rewriting for Optimizing Secure Computation". In: *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy (CODASPY)*. San Antonio, Texas, USA, 2013, pp. 49–58.

[40] Peeter Laud. "Parallel Oblivious Array Access for Secure Multiparty Computation and Privacy-Preserving Minimum Spanning Trees". In: *PoPETs* 2015.2 (2015), pp. 188–205.

[41] Yehuda Lindell and Benny Pinkas. "Secure multiparty computation for privacy-preserving data mining". In: *Journal of Privacy and Confidentiality* 1.1 (2009), p. 5.

[42] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. "ObliVM: A Programming Framework for Secure Computation". In: *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. 2015, pp. 359–376.

[43] Erik Meijer, Brian Beckman, and Gavin Bierman. "LINQ: Reconciling Object, Relations and XML in the .NET Framework". In: *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. Chicago, Illinois, USA, 2006, pp. 706–706.

[44] Arjun Narayan and Andreas Haeberlen. "DJoin: Differentially Private Join Queries over Distributed Databases". In: *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI)*. Hollywood, California, USA, Oct. 2012, pp. 149–162.

[45] New York City Taxi & Limousine Commission. *Taxicab Factbook*. 2014. URL: http://www.nyc.gov/html/tlc/downloads/pdf/2014_taxicab_fact_book.pdf.

[46] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. "Pig Latin: A Not-So-Foreign Language for Data Processing". In: *Proceedings of SIGMOD*. 2008, pp. 1099–1110.

[47] Antonis Papadimitriou, Arjun Narayan, and Andreas Haeberlen. "DStress: Efficient Differentially Private Computations on Distributed Data". In: *Proceedings of the 12th European Conference on Computer Systems (EuroSys)*. Belgrade, Serbia, 2017, pp. 560–574.

[48] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, et al. "Blind Seer: A Scalable Private DBMS". In: *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. 2014, pp. 359–374.

[49] Tal Rabin and Michael Ben-Or. "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract)". In: *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (TC)*. Seattle, Washington, USA, May 1989, pp. 73–85.

[50] Aseem Rastogi, Matthew A. Hammer, and Michael Hicks. "Wysteria: A Programming Language for Generic, Mixed-Mode Multiparty Computations". In: *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. Washington, DC, USA, 2014, pp. 655–670.

[51] Todd W. Schneider. *NYC taxi trip data*. https://github.com/toddwschneider/nyc-taxi-data. Accessed 03/08/2016.

[52] Adi Shamir. "How to share a secret". In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[53] A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, et al. "Hive – A Warehousing Solution over a Map-Reduce Framework". In: *Proceedings of the VLDB Endowment* 2.2 (2009), pp. 1626–1629.

[54] U.S. Census Bureau. "Table 1188 – Credit Cards-Holders, Number, Spending, Debt, and Projections". In: *Statistical Abstract of the United States: 2012*. 131st ed. Aug. 2011. Chap. 25: Banking, Finance, and Insurance.

[55] U.S. Department of Justice and U.S. Federal Trade Commission. *Horizontal Merger Guidelines*. Available at https://www.justice.gov/atr/horizontal-merger-guidelines-08192010. Aug. 2010.

[56] U.S. Office of the Comptroller of the Currency. URL: https://www.occ.treas.gov/.

[57] U.S. Social Security Administration. *Social Security FAQs*. Q20. URL: https://www.ssa.gov/history/hfaq.html.

[58] *VIFF, the Virtual Ideal Functionality Framework*. http://viff.dk/. Accessed: 2015-08-14.

[59] Andrew C. Yao. "Protocols for Secure Computations". In: *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (AFCS)*. Washington, DC, USA, 1982, pp. 160–164.

[60] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. "DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language". In: *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. San Diego, California, USA, Dec. 2008.

[61] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, et al. "Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing". In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI)*. San Jose, California, USA, Apr. 2012, pp. 15–28.

[62] Samee Zahur and David Evans. *Obliv-C: A Language for Extensible Data-Oblivious Computation*. Cryptology ePrint Archive, Report 2015/1153. http://eprint.iacr.org/2015/1153. 2015.

[63] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. "Opaque: An Oblivious and Encrypted Distributed Analytics Platform". In: *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Boston, Massachusetts, USA, 2017, pp. 283–298.