

2019

# Conclave: secure multi-party computation on big data

---

Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, Azer Bestavros. 2019. "Conclave." Proceedings of the Fourteenth EuroSys Conference 2019 - EuroSys '19, <https://doi.org/10.1145/3302424.3303982>

<https://hdl.handle.net/2144/38234>

*"Downloaded from OpenBU. Boston University's institutional repository."*

# Conclave: secure multi-party computation on big data\*

## Extended Technical Report

Nikolaj Volgushev<sup>†</sup>  
Alexandra Institute

Malte Schwarzkopf  
MIT CSAIL

Ben Getchell  
Boston University

Mayank Varia  
Boston University

Andrei Lapets  
Boston University

Azer Bestavros  
Boston University

### Abstract

Secure Multi-Party Computation (MPC) allows mutually distrusting parties to run joint computations without revealing private data. Current MPC algorithms scale poorly with data size, which makes MPC on “big data” prohibitively slow and inhibits its practical use.

Many relational analytics queries 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 them into a combination of data-parallel, local cleartext processing and small 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 further.

Our Conclave prototype generates code for cleartext processing in Python and 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. Thanks to its hybrid protocols and additional optimizations, Conclave also substantially outperforms SMCQL, the most similar existing system.

### 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; 49; 69]; banks and financial regulators can assess systemic risk without revealing their private portfolios [8; 53]; 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, as well as meta-data about them, such as value frequency distributions.

Unfortunately, implementing a performant secure MPC currently requires domain-specific expertise that makes it impractical for most data analysts. Moreover, 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; 69]. These query processors improve the accessibility of MPC, as they allow data analysts to write convenient relational queries; they also improve MPC 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 [69] 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. Like prior systems [3; 49; 50], Conclave is designed to withstand passive (“semi-honest”) adversaries.

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 protocols* that combine cleartext and MPC processing to 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

\*In this extended technical report version of the Conclave paper [64], we include additional material on Conclave’s security guarantees, including full proofs in the Appendix.

<sup>†</sup>Part of the work completed at Boston University.

systems (e.g., Spark [67]) with secure, but slow, cross-party MPC systems (e.g., Sharemind [12] or Obliv-C [68]).

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 in a server-aided setting [41; 43], 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. query analysis techniques 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 Conclave prototype, which applies these ideas to generate efficient code for execution using Python, Spark [67], Obliv-C [68] and Sharemind [12] (§6).

We measured the performance of our prototype using microbenchmarks and end-to-end analytics queries (§7). Even with minimal annotations on input relations and basic optimizations, Conclave scales queries to inputs orders of magnitude larger than existing MPC frameworks support on their own. When users annotate specific input columns, our new hybrid MPC–cleartext protocols speed up join and aggregation operators by 7× or more compared to execution in Sharemind [12], a fast, commercial MPC framework. Compared to SMCQL, Conclave’s extra optimizations help scale a medical research query to orders of magnitude larger inputs with comparable security guarantees.

Nevertheless, our prototype has some limitations. It defends only against passive adversaries, though Conclave’s approach of minimizing MPC is compatible with stronger threat models. While our 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 jointly executes an agreed-upon computation across several parties’ private data without a trusted party. MPC has

served purposes from detecting VAT tax fraud by analyzing business transactions [10], to setting sugar beet prices via auction [13], relating graduation rates to employment [11], and evaluating the gender pay gap across businesses [7]. These applications all compute on hundreds to thousands of records, but many useful computations on large data that might 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.

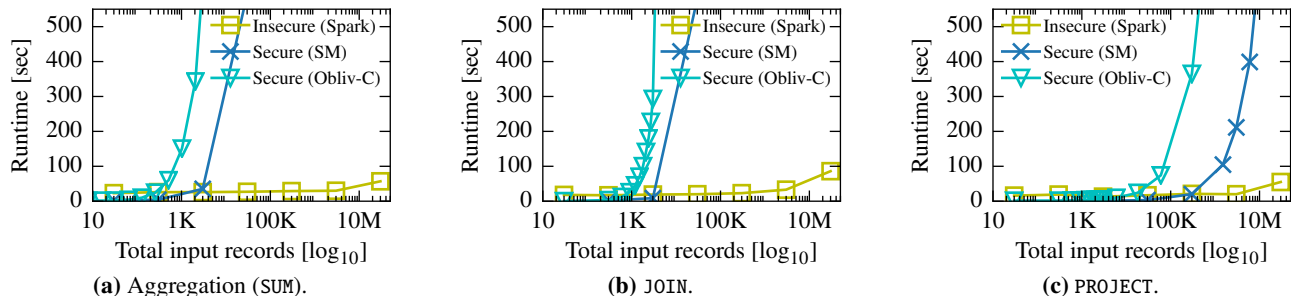
**Credit card regulation.** A government regulator (in the U.S., the OCC [62]) who oversees consumer credit reporting agencies (in the U.S., e.g., TransUnion) may wish to estimate the average credit score by geographic area (e.g., ZIP code). The government regulator holds the social security numbers (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 [63] and 167M credit cards [60] 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 [61, §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 [51]. 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 [46].

MPC provides these guarantees under a specific model of its adversary (i.e., dishonest participant). Commonly, the adversary is presumed to be *passive*, respecting the protocol but trying to learn other participants’ data (the “semi-honest” model), or *actively malicious*, deviating from the protocol’s



**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).

rules to expose private data or compromise the integrity of outputs. Generic techniques for information-theoretically secure MPC require an honest majority [6; 55] whereas techniques that leverage computational assumptions only require a single honest party (an “anytrust” model) [30; 65]. Relaxing security guarantees makes MPC faster: honest (super-)majority techniques perform better than anytrust ones [1; 12; 25], and a passive adversary can be withstood with at least  $7\times$  lower overhead than an actively malicious adversary [2].

### 2.3 MPC techniques and scalability

Two MPC techniques dominate today: garbled circuits and secret sharing. Cryptographers have made efforts to scale these to many *parties* [15; 17–19], but scalability to *large data* remains a challenge. In this work, we focus on scaling to large data sizes, and assume a fixed, small number of parties.

In **garbled circuits** [65], one party encrypts each input bit 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.*,  $80\text{--}128\times$  for typical security parameters), which makes processing of large data impractical.

**Secret sharing** [58], by contrast, splits each sensitive input (*i.e.*, each integer, rather than each bit) into “secret shares”, which in combination yield the original data. In a common encoding, secret shares cancel out into the 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 still requires sending at least one bit between parties [1].

**Scalability in practice.** Figure 1 compares the performance of three relational operators in MPC frameworks based on secret-sharing (Sharemind [12]) and garbled-circuits (Obliv-C [68]) to insecure plaintext execution. Each experiment inputs random integers and runs a single operator. The MPC frameworks run with two (Obliv-C) or three (Sharemind) 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. MPC scales poorly for aggregations (Figure 1a) and joins (Figure 1b). These operators require communication in secret sharing, and Obliv-C’s state grows fast (*e.g.*, the join runs out of memory at 30k records). Even a projection (Figure 1c), which requires only a single pass over the input and no communication, fails to scale in practice. Again, 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 ( $\approx 37$  MB) due to overheads of secret-sharing and in its storage layer. Thus, MPC in practice only scales to a few thousand input records.

These results are consistent with prior studies: Sharemind takes 200s to sort 16,000 elements [39], and DJoin takes an hour to join 15,000 records [49]. 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 60% of privacy-sensitive analytics queries use joins, and over 34% contain aggregations [38, §2]. Short of new cryptographic techniques, the only way run MPCs 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. Intuitively, for example, 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 and reveals their result.

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. This helps scale MPC to large data by using cheaper algorithms, local computation, and data-parallel processing systems for crucial parts of the query.

### 3.1 Threat model

Like many practical MPC systems, Conclave focuses on withstanding a passive, semi-honest adversary. The adversary can observe all network communication during execution of the protocol, and can also statically (*i.e.*, before the protocol begins) choose to compromise some parties to view data stored in their local file system and memory. Because the adversary can monitor 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. Yet, all parties — including compromised ones — faithfully execute the MPC protocol and submit valid input data.

In the following, we restrict our attention to semi-honest security. Appendix A.5 sketches how Conclave could be extended to provide security against malicious adversaries.

### 3.2 Security guarantees and assumptions

Conclave is a query compiler that generates code for execution in external MPC systems (“backends”). Consequently, Conclave inherits the security guarantees and assumptions of the MPC backend used, but must also uphold them. In particular, Conclave hides all private data processed, along with meta-data such as value frequencies. Conclave provides these guarantees against the same threshold of colluding adversarial parties that its MPC backend can tolerate.

Consistent with MPC literature, Conclave treats the sizes of all *input relations* as public, and hides the sizes of intermediate relations processed under MPC. However, after Conclave rewrites a query to move operations out of MPC, the sizes of inputs to the remaining MPC may differ from the original input sizes. Depending on the query, this may leak information. Conclave’s rewrites are safe if the sizes of the new MPC input relations are data-independent. For data-dependent sizes, Conclave only proceeds with a rewrite if all affected parties have authorized it, choosing a slower query plan otherwise.

Additionally, Conclave provides *hybrid protocols* that provide parties with the option to trade some security for better performance. These hybrid protocols outsource work for clear-text processing at a chosen party, and selectively reveal some data to it. Conclave applies such value-leaking rewrites *only* if parties supply explicit input annotations and Conclave can derive an authorization.

Hybrid protocols essentially create a “server-aided” setting [41; 43] with leakage: the aiding *selectively-trusted party* (STP) catalyzes the MPC by performing otherwise expensive operations outside of MPC’s cryptographic guarantees. Only a single STP can exist in a Conclave execution; all other participants are *regular parties*. Which party takes on the role of an STP depends on the specific trust assumptions of a given deployment; one of the data contributors may act as an STP,

---

```

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).

but it may also be a party without inputs whose sole role is to assist the MPC. Conclave retains MPC’s security guarantees against any adversary who compromises the STP alone or a subset of regular parties that the MPC backend can withstand. Conclave makes no guarantees against an adversary who compromises both regular parties and the STP.

## 4 Specifying Conclave queries

Conclave compiles a relational query into executable code. It is similar to plaintext-only big data query compilers (*e.g.*, Hive [59], Pig [52], or Scope [16]) and “workflow managers”, like Apache Oozie [36] or Musketeer [27]. Like these systems, Conclave transforms the query into a directed acyclic graph (DAG) of relational operators, and it executes this DAG on one or more backend systems. Unlike prior query compilers, Conclave’s query rewrite rules must preserve MPC’s security guarantees while aiming to improve performance.

### 4.1 Assumptions

Conclave assumes that analysts write relational queries using SQL or LINQ [48], 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 that communicates with the other parties and manages local and MPC jobs, (*ii*) a local MPC endpoint (*e.g.*, an Obliv-C or Sharemind node), all on private infrastructure and (*iii*) optionally, a parallel data processing system (*e.g.*, Spark). Absent a parallel data processing system, Conclave runs local computation in sequential Python.

```

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).

## 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 [66].

Even though the input data to a Conclave query is distributed across multiple parties, Conclave largely abstracts this fact away from analysts. Instead, 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).

The only difference to a simple query is that each input relation has an “owner”, *viz.*, the party storing it, supplied via an `at` annotation (lines 6–11 and 5–10). This information helps Conclave both locate the data and derive 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 (`to`). These parties receive 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 languages (*e.g.*, SecreC [37] or Obliv-C [68]). Such interfaces are expressive, but 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 locations, Conclave also supports optional, light-weight *trust annotations* that help it apply further optimizations. These annotations specify parties who are authorized to learn values in specific input schema columns in the clear to compute more efficiently on them.

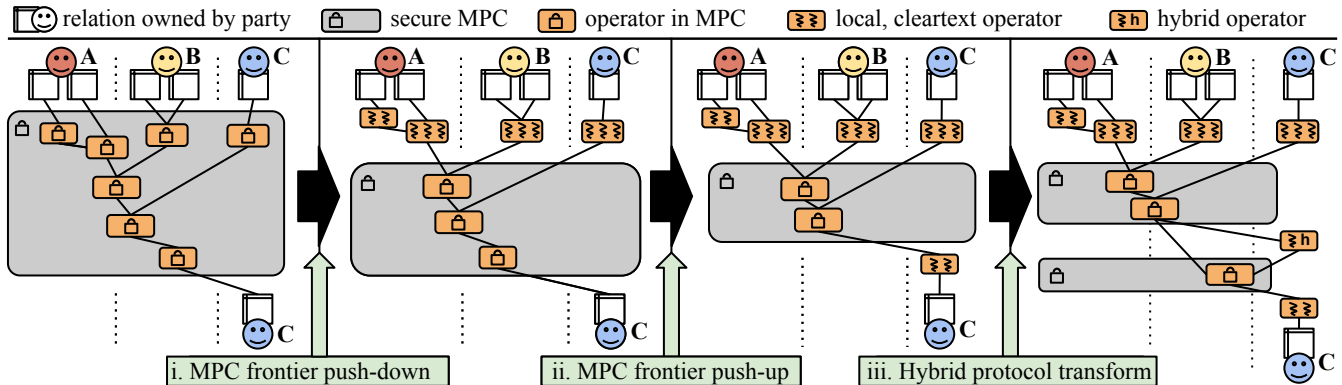
The intuition behind trust annotations is that the sensitivity of data within a relation often varies by column. Consider a relation that holds information about a company’s branches: it may have public address and zip columns (as this information is readily available from public sources), but privately-owned columns manager or turnover. Other columns may be private, but the owning party might be happy to reveal them to a specific *selectively-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, who are competitors). Hence, the parties agree to make the regulator an STP for the `ssn` column of the credit card companies’ customer relations (see Listing 1, line 8). Such selective revealing of columns helps 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 an STP 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 implicitly in the trust set for all its columns, as are recipients of an output relation. Finally, a public column 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 data analysts from manual labor and to avoid subtle mistakes. 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 MPC’s security guarantees. To achieve this, Conclave applies a combination of static analysis, query rewriting transformations, and partitioning heuristics.

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



**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. In this example, the rightmost party (C, blue) contributes data and also acts as a selectively-trusted party for the hybrid join operator.

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 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 protocol 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 do improve end-to-end query runtime, but they do *not* strictly reduce the overall work. In fact, some transformations create *additional* local processing work for parties, or they 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 join 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 in two passes, which infer the execution constraints on its operators.

The **first pass** 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. For each intermediate operator, the propagation derives the *owner* of its result relation. A party “owns” a relation if it can derive it locally given only its own data. Input and output relations are owned by the parties that store them.

Relation ownership propagates along edges according to inference rules. 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, that owner propagates to the output relation; if they have different owners, the output relation has no owner. This process captures the fact that no single party can compute output that combines different parties’ data. Operators with output relations that lack an owner *must* run under MPC.

In a **second pass**, Conclave determines the trust set (cf. §4.3) for each intermediate relation’s columns. A party is “trusted” with an intermediate result column if it is entrusted with enough input data in order to calculate that column in the clear. Propagating this notion of trust enables Conclave to use hybrid protocols to compute intermediate relations, while ensuring that no more information is leaked than was explicitly authorized via the parties’ trust annotations on their input schema. Columns within the same relation may have

different trust sets, because knowing one column of a relation does not imply the ability to calculate any other columns.

The trust set for each input column is defined by the trust annotation. Conclave propagates these trust annotations down the DAG in topological order. For each result column  $c$  of each operator  $o$ , Conclave determines which of  $o$ 's operand columns contribute toward the calculation of  $c$ . Conclave sets the trust set of  $c$  as the intersection of trust sets of these operand columns. The dependencies between operand and result columns are defined by operator semantics in two ways. First, a result column depends on all operand columns that directly contribute rows to it. For example, the first result column of a `concat` depends on the first column of each concatenated relation, since its rows are derived from those operand columns. Second, a result column depends on all operand columns that affect how its rows are combined, filtered, or reordered. For example, the computed column of a `sum` depends both on the column aggregated over *and* the group-by columns, since the group-by columns determine how the aggregated column rows are combined. Likewise, all result columns of a `join` depend on the join key columns, which determine whether a row is part of the output. Conclave encodes column dependencies for all operator types it supports.

This propagation algorithm helps Conclave maintain an important security invariant: Conclave only reveals a column to a party if the column can be derived from input columns that the party is authorized to learn. We use this guarantee to ensure that Conclave's use of hybrid protocols is safe.

## 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 preprocessing operators and a smaller MPC step. These transformations push the *MPC frontier*—*i.e.*, the boundary between MPC operators and local cleartext operators—deeper into the DAG, where a clique of operators remains under MPC. The transformations have little to no impact on the cryptographic security guaranteed by Conclave's backend.

**MPC frontier push-down.** Starting from the input relations, Conclave pushes the MPC frontier down the DAG as far as possible while preserving correctness and security. 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  $pN$ ,

$$\text{op}(R_{pA} \mid \dots \mid R_{pN}) \equiv \text{op}(R_{pA}) \mid \dots \mid \text{op}(R_{pN}).$$

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, to split an aggregation that groups a partitioned relation by key, Conclave adds per-party aggregations over singleton relations, followed by a secondary aggregation. This transformation may have security implications, which we discuss at the end of this section. While the secondary aggregation must remain under MPC, Conclave can pull the local preaggregations 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 (Line 17, Listing 2), 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 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 reordering projections; additionally, while aggregations are generally not reversible, special cases are. If `count` occurs as a leaf operator, then 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.

**Security implications.** Conclave's push-down and push-up query transformations can affect security guarantees. Without any transformations (*i.e.*, if the entire query remains

within the MPC frontier), Conclave directly inherits the security guarantees provided by its MPC backend. Conclave’s push-up operations have no impact on security because they are reversible, and thus the input to the operator is simulatable from its output. However, push-downs do have a security implication: they change the lengths of inputs to the MPC backend. For instance, splitting an aggregation into a local step and an MPC step (even as part of a “secondary aggregation” as described above) 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.

In the appendix, we formally prove that the leaked input length information is the only security implication of Conclave’s MPC frontier changes (see Theorem A.2).

### 5.3 Hybrid operators

In this rewrite pass, Conclave splits work-intensive operators, such as joins and aggregations, into *hybrid operators*. Hybrid operators outsource expensive portions of an operator to a selectively-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 (lines 1–2 of Figure 3).
2. The parties project away all columns other than the join key columns, and reveal the resulting key-column-only relations to the STP (lines 3 and 5).
3. For each key-column relation, the STP enumerates the rows (lines 4 and 6). 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 (line 7). 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 the row identifier columns for the left and right relations into two new relations and secret-shares these to the untrusted parties (lines 8–11).
6. Back under MPC, the parties perform an oblivious indexing protocol (the `select` operator) akin to the one by Laud [45] for each shuffled input relation (lines 12 and 13). Given a set of secret indexes, the indexing protocol obliviously selects the rows at the corresponding positions from a relation. Using the index relations and shuffled inputs, the parties select the left and right rows comprising the result, producing two relations `left_rows` and `right_rows`.
7. The parties concatenate the two relations column-wise and obliviously reshuffle the result (lines 14–16).

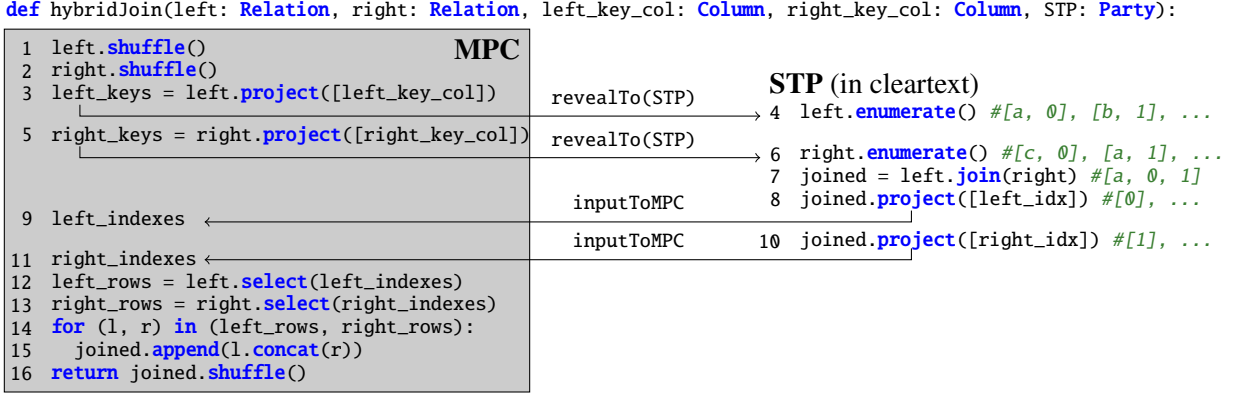
The hybrid join gives an asymptotic improvement over a standard MPC join: the oblivious indexing protocol in Step 6 requires  $\mathcal{O}((n + m) \log(n + m))$  non-linear operations, where  $n$  is the input size and  $m$  the result size, whereas an MPC join requires  $\mathcal{O}(n^2)$  non-linear 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 a randomly selected party; the party enumerates and joins the rows, and 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.

**Hybrid aggregation.** Conclave can transform an MPC aggregation into a hybrid aggregation if the trust set on the group-by column contains an STP.

Conclave’s hybrid aggregation protocol adapts the sorting-based MPC protocol by Jónsson *et al.* [39]. 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. It proceeds as follows:

1. The parties obliviously shuffle the input and reveal the shuffled group-by column to the STP. We refer to the entries in the group-by column as *keys*.
2. Locally, the STP enumerates the revealed keys, producing a relation with a key column and an index column. The STP sorts the relation on the key column. This groups together all rows with equal keys, and creates



**Figure 3.** For the hybrid join, Conclave augments MPC with local computation to speed up execution. The protocol performs oblivious shuffles (lines 1–2) followed by a cleartext enumeration, a join, and a projection (lines 4–10) outside MPC (at the STP), and finishes with inexpensive oblivious indexing to reconstruct the join result under MPC (lines 12–16). The `revealTo` and `inputToMPC` operations move data in and out of MPC; `revealTo` reveals secret data to a specific party, while `inputToMPC` inputs a local dataset into MPC, for instance via secret-sharing.

a mapping for each row in the sorted relation to its original position via the indexes.

3. The STP scans over the relation and computes an *equality flag* for each row that indicates whether the keys for this row and the previous row are equal.
4. The STP projects away the key column from the sorted relation, leaving only the indexes. The STP sends the indexes to the other parties *in the clear*.
5. The STP secret-shares the equality flags.
6. Using the plain-text ordering information, the untrusted parties reorder the rows of the shuffled relation so that the rows are sorted by group-by column values.
7. Under MPC, the parties scan over the result. For each entry, they obliviously aggregate the previous value into the current if the corresponding secret equality flag is one. 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.
8. The parties shuffle the result and reveal the equality flags; they discard all entries with the flag set.

The hybrid aggregation improves asymptotically over the regular MPC protocol: the oblivious sorting step of the original protocol 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. However, the hybrid aggregation leaks the size of the result to all untrusted parties.

**Security implications.** Conclave’s hybrid operators 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 the hybrid relation. Conclave only uses hybrid operators if it can derive appropriate authorizations for this leakage from input column annotations.

In the appendix, we prove that, modulo this leakage, Conclave continues to achieve MPC’s simulation-based security guarantee when using hybrid operators (Theorem A.4). At a high level, we prove this by partitioning the computation into three distinct MPC stages — before, during, and after the hybrid operator — using the fact that simulation-based security definitions provide sequential composition. We show that the leakage stated above makes the view of the STP simple to simulate and has little impact on Conclave’s security against a coalition of regular parties.

#### 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 protocol by Jónsson *et al.* [39] uses sorts).

These operations, especially sorts, are expensive under MPC, since they must remain control-flow agnostic. However, if an operator produces a sorted relation, subsequent sorts can in some cases be eliminated. For instance, if a query consists of an order-by operation followed by an oblivious aggregation, it is unnecessary to sort as part of the aggregation, as the relation is already in the correct order. Conclave thus minimizes the use of oblivious sorts by traversing the DAG, tracking the columns by which intermediate relations are sorted (if any), and eliminating redundant sorts. Some operations, such as shuffles, do not preserve row order; Conclave’s propagation therefore also tracks when a sorted relation is permuted and marks the result as unsorted. This optimization yields especially high performance gains when a relation is sorted in the clear (for instance by a public column) as it allows Conclave, depending on the query, to avoid oblivious sorting altogether.

The above optimization only tracks redundant sorts; we can further improve it by moving sorts 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 obviously merges them, which is cheaper than obviously sorting the entire data.

In future work, we plan to extend Conclave with these improvements. 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 [27]. Our prototype consists of 8,000 lines of Python, and currently integrates sequential Python and data-parallel Spark [67] as cleartext backends, and Sharemind [12] and Obliv-C [68] 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 appears, for example, sufficient to express 88% of 8M sensitive real-world queries at Uber [38].

We implemented the same standard MPC algorithms for joins (a Cartesian product approach) and aggregations [39] in both Sharemind and Obliv-C.

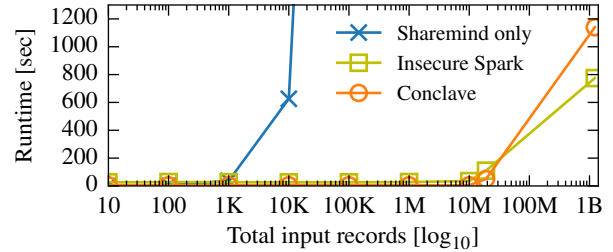
## 7 Evaluation

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

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

**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.



**Figure 4.** Conclave runs the market concentration query in <20 minutes for 1B input records; under Sharemind MPC, the query cannot scale past 10k input records.

### 7.1 Market concentration query

The market concentration query computes the Herfindahl Hirschman Index (HHI) [34] 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 [57]. We randomly divide the trips across three imaginary VFH companies and filter out any trips with a zero fare, resulting in a total 1.3 billion trips across all parties. We subsample different numbers of rows from the input data, and measure the query execution time for different input sizes.

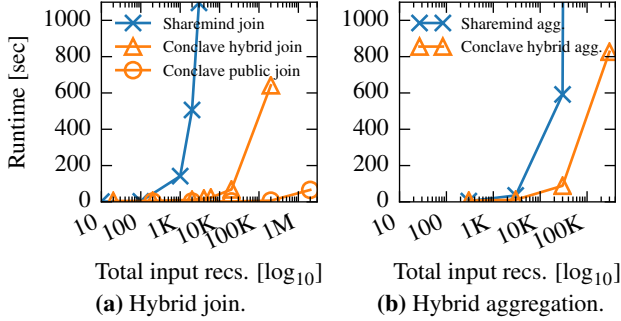
The query contains both an aggregation and a self-join, and the runtime of these expensive operators dominates all others. Consequently, we expect the query to scale poorly when run entirely under MPC in Sharemind, but for Conclave to improve its performance by precomputing the aggregation. Figure 4 shows that Sharemind indeed takes over an hour to complete the query at 100k input rows, while Conclave scales roughly linearly in the size of the input data. This comes because Conclave 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.

Finally, we run the same query insecurely on the joint data using a single, nine-node Spark cluster that processes all parties’ combined data. In this insecure setting, we observe similar performance. Up to 10M, this insecure setup is slightly slower than Conclave, as it runs one job rather than three parallel jobs, but at 1.3B records, insecure Spark benefits from the additional parallelism of the joint nine-node cluster and completes quicker.

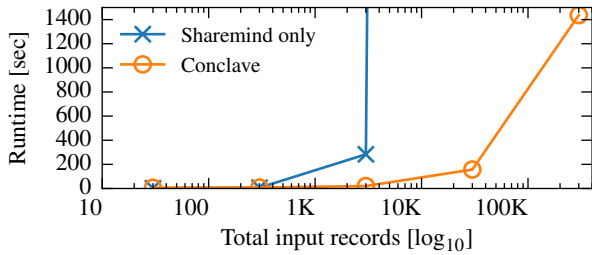
The market concentration query benefits from Conclave pushing down the MPC frontier and splitting the initial aggregation, but does not use hybrid protocols.

### 7.2 Hybrid operator performance

Conclave transforms queries to use hybrid protocols if a selectively-trusted party participates (§5.3). We measure the impact of these hybrid protocols with queries with only a single join or aggregation, and a growing amount of synthetic input data. We annotate the input relations’ columns with



**Figure 5.** Conclave’s hybrid operators help scale joins and aggregations to large data by combining MPC and cleartext compute; At 10k records per party, Sharemind alone takes ten minutes (aggregation) and over twenty minutes (join).



**Figure 6.** 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.

STPs, 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 5 confirms this. Without an STP, Conclave must run the query entirely under MPC, which exhibits performance similar to earlier benchmarks (§2.3). Conclave’s 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, for input size  $n$  and join output size  $m$ . Consequently, using a hybrid join operator substantially improves scalability: a hybrid join on 200k records takes just over ten minutes. (At 2M input records, Sharemind runs out of memory while executing the MPC part of the hybrid join.)

The public variant of the join operator scales even better since it avoids the use of MPC altogether (hence, it completes at 2M records). The public join’s bottleneck is the local join.

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

### 7.3 Credit card regulation query

Conclave’s hybrid operators offer substantial benefits for queries whose performance is dominated by aggregations and joins (a common case). 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.

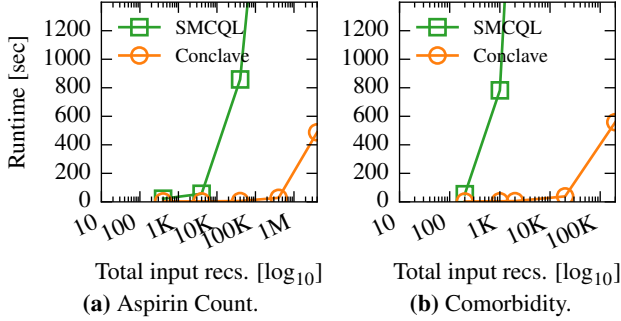
Figure 6 confirms this. Running the query entirely under MPC in Sharemind fails to scale beyond 3,000 total records; at 30k, the query does not complete within two hours. This comes despite the Sharemind baseline using a join implementation that leaks output size, matching the leakage of Conclave’s hybrid join operator. With Conclave’s hybrid operators, however, the query processes 300k records in under 25 minutes. The experiment highlights that hybrid operators are crucial to obtaining good performance for this query. The query’s first operator is a join, so Conclave cannot push the MPC frontier down and, without hybrid operators, would have to run the entire query under MPC.

### 7.4 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 optimizations, which compose with Conclave’s optimizations.

SMCQL relies on the OblivM [47] garbled-circuit framework for MPC. OblivM supports only two-party computations and is slower than Sharemind, particularly on large data. This difference is not fundamental: SMCQL could generate code for Sharemind. Conclave uses the Sharemind backend in these experiments, and has two parties provide input, while the third participates in the MPC without inputs. Conclave and SMCQL make comparable security guarantees, with the exception that their backends’ corruption thresholds differ (OblivM: one of two, Sharemind: one of three). Because SMCQL requires more memory, the experiments use larger VMs with 32 GB RAM and 8 vCPUs.

We benchmark the *aspirin count* and *comorbidity* queries from the SMCQL paper. We omit running the third query



**Figure 7.** Conclave outperforms SMCQL on the *aspirin count* and *comorbidity* queries [3, §2.2.1]. For *aspirin count*, Conclave’s optimizations lift additional work out of MPC. At 200k records, SMCQL ran for over an hour for both queries.

(*recurrent c. diff.*), as Conclave does not yet support window aggregates, but summarize the performance we would expect.

**Aspirin Count** [3, §2.2.1]. The query joins two input relations, diagnoses and medications on public, anonymized patient IDs, filters by patient diagnosis and prescribed medication (both private columns), and counts the results. These inputs to the join are partitioned across two hospitals, *i.e.*, each party holds part of diagnoses and part of medications.

SMCQL’s “slicing” partitions the data on the public patient ID column. Slices with patient IDs only one party has are processed locally at that party, while the other slices must be processed under MPC. For this experiment, we manually implement SMCQL’s slicing and combine it with Conclave’s public join. We generated input data with a 2% overlap between the parties’ uniformly-random patient IDs, similar to the HealthLNK data [3, §7.2]. We measure query runtime for increasing numbers of input records per party.

Figure 7a shows that Conclave consistently outperforms SMCQL, and that it scales better. At 40k rows, Conclave completes in 3.7 seconds, while SMCQL takes 14.3 minutes; SMCQL does not finish within an hour for 400k or more rows, while Conclave takes under two minutes. Finally, Conclave processes 4M input records in 8 minutes.<sup>1</sup> This improvement is due Conclave’s public join and its sort elimination optimization. By combining the public join with slicing, Conclave can compute the initial join in the clear, and sends only rows for patient IDs present at both parties into MPC. By contrast, SMCQL still runs the join and the subsequent operations obliviously for each private slice, which has quadratic cost in the size of the slice.

Sort elimination allows Conclave to avoid an expensive oblivious sort step otherwise required for the distinct count. Conclave performs the sort in the clear, as part of the public join; since all operations under MPC are order-preserving no

<sup>1</sup>Bater *et al.* also benchmarked *aspirin count* for larger inputs of 42M diagnoses and 23M medications using 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.

subsequent sorts are necessary. This reduces the complexity of the MPC from  $\mathcal{O}(n \log n)$  to  $\mathcal{O}(n)$ . SMCQL could likewise benefit from this optimization.

**Comorbidity.** The comorbidity query determines the most common conditions that affect patients suffering from *c. diff.* ([3, §2.2.1]). It consists of a group-by aggregation on a private column (the patient’s diagnosis) and an order-by (with a limit clause) of the result. As with aspirin count, the data is drawn from the HealthLNK dataset [33]. The input relation diagnoses is partitioned horizontally across two parties.

Conclave and SMCQL both split the aggregation into a local and an MPC step via an MPC-pushdown, and execute the rest of the query under MPC. The local aggregation reduces the amount of data entering MPC since neither Conclave nor SMCQL pad the result. As such, the runtime of the query depends on the number of distinct keys in the patient diagnosis column. In our experiments, we set the number of distinct keys to 10% of the total number of input rows. We increase the number of records in the input relation — *i.e.*, the total input is twice the  $x$ -axis value — and measure query runtime.

As per Figure 7b, Conclave outperforms SMCQL, with the gap increasing with data size. Conclave scales to 200k total rows (20k rows entering MPC) while SMCQL takes over an hour at the 20k mark. As Conclave and SMCQL apply the same optimizations, the improvement is due to a difference in MPC backends. Conclave’s Sharemind backend is better suited for arithmetic-heavy relational queries and considerably outperforms ObliVM [47]. The relative performance improvement and scalability of Conclave is lower than for aspirin count since the query does not lend itself to Conclave’s additional optimizations and requires two expensive operations (aggregation and order-by) to run under MPC.

**Recurrent c. diff.** The third query in the SMCQL paper, *recurrent c. diff.* consists of a window aggregate, a join on public columns (patient IDs), and a selection of distinct patient IDs from the result. SMCQL performs the window aggregate and join in sliced mode, and runs the distinct operator in the clear. Like in *aspirin count*, the query’s performance is bottlenecked on the join [3, §7.3]. This query is amenable to Conclave’s public join optimization, so we expect Conclave to outperform SMCQL on the MPC steps due to Conclave’s use of Sharemind. Based on our results for aspirin count (cf. Figure 7a), we expect Conclave’s public join (in combination with slicing) to match or exceed the performance gain SMCQL realizes through sliced-mode execution.

## 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 [56] 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. However, the programmer must still manually annotate which blocks of the computation run under MPC, and which blocks are to be carried out locally. These annotations are much more fine-grained than Conclave’s input annotations, and require MPC proficiency. By contrast, Conclave targets data analysts with minimal MPC knowledge, and focuses on automation. Conclave automatically optimizes queries to reduce the use of MPC, for instance via the MPC pushdown, whereas Wysteria requires the programmer to manually identify and implement such optimizations via the aforementioned per-block annotations. Furthermore, Conclave supports parallel cleartext processing using existing frameworks like Spark.

**Query rewriting and MPC alternatives.** There are several efforts to optimize MPC via query rewriting, like Conclave does, at different levels of abstraction. Kerschbaum [44] 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 [69]. 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 as an alternative to MPC; its query rewriting focuses on reducing the number of oblivious sorts required in distributed computation across multiple SGX machines. Similarly, Prochlo [9] combines the use of SGX, secret-sharing based techniques, and differential privacy [22] to implement large-scale application usage monitoring. In contrast to Conclave, Prochlo targets a setting where a large number of parties, *i.e.*, users, contribute data to the computation. It furthermore relies on specialized hardware, and does not match MPC’s full privacy guarantees.

**Protected databases and scalability.** The protected database community has produced decades of research on scaling secure query execution to the gigabyte-to-terabyte range [14; 24; 31]. This includes work on optimizations for boolean keyword search [23; 54], as well as large, general subsets of relational algebra [40]. 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 [35; 42], linear algebra [26], or matching [21].

**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) [22] 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 the threat of parties jointly reconstructing sensitive input data. DJoin [49] does so for SQL-style relational operations (with query rewriting, but without Conclave’s automation and hybrid protocols), DStress [53] does so for graph analysis, and He *et al.* [32] do so for private record linkage. Conclave does not currently leverage DP, but adding it would require no fundamental changes to the query compilation.

## 9 Conclusion and future work

Conclave speeds up secure MPCs on “big data” by rewriting queries to minimize expensive processing under MPC. Conclave runs queries in minutes that were impractical with previous MPC frameworks or would have required domain-specific knowledge to implement.

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 leaking relation sizes on the MPC boundary.

Conclave is open-source and available at:

<https://github.com/multiparty/conclave>.

## Acknowledgements

We thank Ran Canetti, Tore Kasper Frederiksen, Derek Leung, and Nickolai Zeldovich for their helpful feedback on drafts of this paper. We are also grateful to the helpful comments we received from our anonymous reviewers, and our shepherd, Christian Cachin. This work was funded through NSF awards CNS-1413920, CNS-1414119, CNS-1718135, and OAC-1739000, and by the EU’s Horizon 2020 research and innovation programme under grant agreement 731583.

## A Security analysis

To prove Conclave’s security guarantees, we analyze the security of Conclave’s push-down, push-up, and hybrid transformations individually and collectively.

### A.1 Definitions

A secure MPC protocol  $\pi$  *securely computes* function  $f$  subject to a (monotone-decreasing) adversary structure  $\Delta$  if  $\pi$  enables parties with inputs  $\vec{x}$  to learn  $f(\vec{x})$  while ensuring that no sufficiently small colluding subset of parties can learn any additional “useful” information from everything they view during the protocol. We codify this statement in pieces.

First, an adversarial set of colluding parties  $\mathcal{C}$  is deemed to be *permissible* if  $\mathcal{C} \in \Delta$ ; these are the only adversaries that

$\pi$  vouches to withstand. Second, the *view* of an adversary is defined as the state of all colluding parties in  $\mathcal{C}$  together with the set of all network messages they observe. Third, we simultaneously guarantee MPC’s correctness and security by requiring that  $\mathcal{C}$ ’s view can be simulated given only the colluding parties’ inputs ( $\vec{x}_{\mathcal{C}} = \{x_i : i \in \mathcal{C}\}$ ) and the size of all parties’ inputs ( $\vec{\ell}$ , where  $\ell_i = |x_i|$ ); ergo,  $\mathcal{C}$  cannot learn more information than this from its own view. Formally, we require that for all permissible adversaries  $\mathcal{C}$ , there exists a simulator  $\mathcal{S}$  such that the following two ensembles are computationally indistinguishable [28] when the distinguisher and  $\mathcal{S}$  run in time polynomial in the security parameter:

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

Correctness follows because  $\text{out}^{\pi} = f(\vec{x})$  and security stems from the fact that the adversary’s view contains “no more information” than the adversary’s own input and the length of the honest parties’ inputs. If the adversarial set  $\mathcal{C}$  includes the receiving party, then  $\mathcal{S}$  is also given  $f(\vec{x})$  since the adversary is supposed to learn it through the execution of  $\pi$ .

## A.2 Semi-Honest Security without Hybrid Operators

Without hybrid operators, the execution of Conclave on some function  $g$  proceeds in three phases as shown in Figure 2.

- Local pre-processing  $d$ : each party calculates  $d_i(x_i)$ .
- A single MPC calculation over a set of operators  $f$  that ultimately reveals some value  $y$  to the receiving party.
- Local post-processing  $u$ : receiving party outputs  $z = u(y)$ , where  $u$  is an invertible function.

Conclave’s push-down and push-up transformations choose the local pre- and post-processing operators.

The security of this transformation can be proved via a standard composition argument. We use the composition lemma from Goldreich [29, §7.3.1].

**Lemma A.1.** *Suppose there exists an MPC protocol  $\pi^f$  that securely computes  $f$  and another protocol  $\pi^{g \circ f}$  that securely computes  $g$  but is permitted to make oracle queries to  $f$ , where both protocols provide semi-honest security with respect to the same adversary structure  $\Delta$ . Then,  $g$  can be securely computed (without oracle queries) via the protocol  $\pi^g$  that runs  $\pi^{g \circ f}$  but substitutes every oracle call to  $f$  with an execution of  $\pi^f$ ; this protocol leaks the length of the input to  $f$  in the process.*

This composition statement applies to all secure computation protocols, whether based on secret sharing or garbled circuits. Ergo, the lemma applies to all Conclave backends.

**Theorem A.2.** *Consider Conclave with an MPC backend that provides semi-honest secure computation under adversary structure  $\Delta$ . Suppose Conclave splits the function to be computed  $g = u \circ f \circ d$  into a set of local per-party pre-processing operators  $\{d_i\}$ , an operator  $f$  intended for secure computation, and a local post-processing operation  $u$  performed by the receiving party. Then, Conclave securely*

*computes  $g$  as per equation (1) with the same adversary structure  $\Delta$ , except that the protocol reveals the input lengths of  $f$  and not  $g$  (i.e.,  $\ell_i = |d_i(x_i)|$ ).*

For simplicity, we assert without proof that Conclave’s query transformations preserve correctness (mostly due to the distributive law). The proof below focuses on security.

*Proof.* In order to apply Lemma A.1 to  $g = u \circ f \circ d$ , we must show the existence of two MPC protocols  $\pi^f$  and  $\pi^{g \circ f}$ .

Conclave’s existing backend already provides us with a protocol  $\pi^f$  that securely computes  $f$  under some given adversary structure  $\Delta$ . Furthermore, we can directly construct a protocol  $\pi^{g \circ f}$  that securely computes  $g$  under the same adversary structure  $\Delta$  given oracle access to  $f$ . This protocol  $\pi^{g \circ f}$  proceeds as follows. First, each party locally computes its own  $d_i(x_i)$  operation and feeds the result to the  $f$  oracle. Second, the receiving party takes the oracle’s response  $y$  and calculates the output  $z = u(y)$ .

We show how to simulate the view of any adversarial coalition  $\mathcal{C} \in \Delta$ . If  $\mathcal{C}$  does not include the receiving party, then the adversary never receives any messages and the only intermediate state it computes is  $d_{\mathcal{C}} = \{d_i(x_i) : i \in \mathcal{C}\}$ , which the simulator can also compute directly from its inputs. If  $\mathcal{C}$  includes the receiving party, then the only additional item in the view is  $y$ , which the simulator can reconstruct as  $u^{-1}(g(\vec{x}))$  since  $u$  is invertible and the simulator for this coalition is given  $g(\vec{x})$ .

Finally, with protocols  $\pi^f$  and  $\pi^{g \circ f}$  constructed, we may apply Lemma A.1 to produce a protocol  $\pi^g$  that securely computes  $g$ . Note that equation (1) permits the simulator for  $\pi^f$  to use the input lengths to  $f$ , whereas the simulator for  $\pi^{g \circ f}$  never utilized any input length information. Ergo, simulation of  $\pi^g$  only requires the input lengths to  $f$ .  $\square$

## A.3 Hybrid Operators Stand-Alone Security

In this section, we provide simulation-based proofs of semi-honest static security for the three hybrid operators described in §5.3. Each proof is stand-alone; that is, it presumes that the entire Conclave calculation contains a single operator. Once again, we assert correctness without proof, and we focus solely on the simulation security arguments. The proofs are partitioned based upon whether the colluding set includes the selectively-trusted party (STP). If so, then recall that the STP must operate alone; it is not permitted to collude with other participants.

**Theorem A.3.** *Suppose that Conclave uses a secure MPC backend whose adversary structure  $\Delta$  includes  $\{\text{STP}\} \in \Delta$  but  $\text{STP} \notin \delta$  for any other set  $\delta \in \Delta$ , and it supports secure protocols for oblivious shuffles and indexing. Then the algorithm for each hybrid operator in §5.3 results in a standalone secure computation of its corresponding operator as per equation (1), subject to the following leakage: the STP learns the join*

or aggregation key column, and all parties learn the number of rows for the input and result of the operator.

Because the theorem focuses on standalone security, we demonstrate it separately for each of the 3 hybrid operators.

*Proof for hybrid join.* First, we analyze security against the STP. In step 2 of the algorithm, the STP receives the set of key columns corresponding to the join operator, which the simulator can emulate because it receives this information as leakage. In all other steps, the STP is either not involved or it participates as a single party within an MPC protocol, which is simulatable because the STP on its own is a permissible set.

Second, we construct a simulator  $\mathcal{S}$  for a permissible coalition of regular parties  $\mathcal{C} \in \Delta$ . This coalition participates in a secure computation involving the other parties' input data along with (in steps 5–7) the STP's mapping of index relations between parties. Because  $\mathcal{C}$  is a permissible set, the Conclave MPC backend guarantees the existence of a simulator  $\mathcal{S}'$  that can emulate  $\mathcal{C}$ 's view when provided with  $\mathcal{C}$ 's inputs  $\vec{x}_{\mathcal{C}}$  as well as the input lengths for both the honest parties' relations  $\vec{\ell}$  and the STP's index-mappings  $\ell_{\text{STP}}$ . Since  $\mathcal{S}$  receives  $\vec{x}_{\mathcal{C}}$  and  $\vec{\ell}$  by definition and can compute  $\ell_{\text{STP}}$  from its leakage (the eventual number of rows in the joined result),  $\mathcal{S}$  can execute  $\mathcal{S}'$  to emulate  $\mathcal{C}$ 's view.  $\square$

*Proof for public join.* Recall that the public join operator is not conducted under MPC at all: the parties disseminate their key columns in the clear and the aiding server calculates the join in the clear as well. This operator requires a very large amount of leakage: every participant agrees to disclose the relevant key columns to all other participants. Given such leakage, the view of a public join can be trivially simulated by re-calculating the operator in the clear following the same procedure as the aiding server.  $\square$

*Proof for hybrid aggregation.* This proof is similar to that for the hybrid join. For the STP: the view of the message it receives in step 1 can be simulated by taking a random permutation of the multiset of group-by key values that it receives as leakage. In the remaining steps, the STP merely participates in the MPC protocol so its view is simulatable because the STP on its own is a permissible coalition.

Generating a simulator  $\mathcal{S}$  for a permissible coalition of regular parties  $\mathcal{C} \in \Delta$  once again relies upon executing the simulator  $\mathcal{S}'$  provided by the MPC backend for the MPC steps of the aggregation (steps 5–8), and again the only challenge is to ensure that  $\mathcal{S}$  can calculate the length information to provide to  $\mathcal{S}'$ . In a hybrid aggregation, the length of the STP's input in this protocol only depends on the input lengths by the regular parties, and the length of the STP's output equals the output length of the entire aggregation. As a result,  $\mathcal{S}$  and  $\mathcal{S}'$  require precisely the same length information.  $\square$

#### A.4 Semi-Honest Security with Hybrid Operators

To complete the security analysis of Conclave in the semi-honest setting, we augment the composition argument to account for the presence of hybrid operators. In its full generality, Conclave's static analysis splits the function  $g$  to be computed into the following series of  $2k + 1$  functions:

$$g = u \circ e^k \circ e^{k-1} \circ \dots \circ e^2 \circ f^2 \circ e^1 \circ f^1 \circ d, \quad (2)$$

where  $d$  and  $u$  denote local pre- and post-processing, the  $f^j$  functions are intended for (generic) secure computation using the Conclave backend, and the  $e^j$  functions are hybrid operators. We assume without loss of generality that the distributed computation begins and ends with non-hybrid MPC steps, which might be identity function operators.

To invoke composition when several operators in a row are processed using secure computation, we “lift” the functions to their equivalents that operate over shared state. That is:

- $\hat{d}$  = local pre-processing and randomized sharing.
- Each  $\hat{e}^j$  and  $\hat{f}^j$  receives a sharing of the corresponding input for  $e^j$  or  $f^j$ , respectively, and produces a random sharing of the corresponding output.
- $\hat{u}$  = reconstruction and local post-processing by the receiving party.

When we say “sharing” here, we use the term generically in order to consider both secret sharing-based MPC and garbled circuit-based MPC. For the latter, we follow the approach of Demmler et al. [20] to consider one bit of secret state in a garbled circuit as being “shared” by having the garbler know the values corresponding to the two wire labels and the evaluator know exactly one of the wire labels.

Like most MPC protocols, the backends for Conclave are *reactive* and permit calculations directly over the shares. That is, they can securely compute the  $\hat{f}^j$  or  $\hat{e}^j$  functions over an already-shared state, without the need to perform an initial sharing of inputs or reconstruction of the output. We use this observation in the following theorem.

**Theorem A.4.** *Consider Conclave with an MPC backend that provides semi-honest secure computation under adversary structure  $\Delta$  such that  $\{\text{STP}\} \in \Delta$  and furthermore this is the only set in  $\Delta$  that includes the STP. Suppose also that Conclave's backend is reactive, with well-defined algorithms to share and reconstruct state and with all intermediate state during a calculation being shared among the participants. Given a function  $g$  that Conclave partitions into the sequence of operators in equation (2), Conclave securely computes  $g$  as per equation (1) in the server-aided setting with adversary structure  $\Delta$  subject to the following leakage: every party learns the input and output lengths of  $d$  and all  $e^j$ , and the regular and selectively-trusted parties receive the stand-alone leakage of each constituent hybrid operator  $\hat{e}^j$ .*

As before, we presume Conclave preserves correctness and focus on proving security of the composed construction.

*Proof.* Theorem A.3 provides us with MPC protocols for each  $\hat{f}^j$  and  $\hat{e}^j$ . Additionally, we can construct secure computation for  $\hat{d}$  and  $\hat{u}$  just as was done in the proof of Theorem A.2 with minor changes: simulating the initial sharing step within  $\hat{d}$  requires generating shares of  $d_i(x_i)$  for  $i \in \mathcal{C}$  and shares of random values for the parties outside of the coalition, and inverting  $\hat{u}$  requires computing  $u^{-1}$  and then generating random shares of the result.

All that remains is to apply Lemma A.1. We observe that the simulator for each  $\hat{f}^j$  or  $\hat{e}^j$  operator is provided with the length of its own input, plus optionally the leakage stated for each hybrid operator  $\hat{e}^j$ . We have already provided the overall simulator  $\mathcal{S}$  for  $\pi^g$  with all of the length information as leakage; note in particular that the output length of  $\hat{f}^j$  equals the input length of  $\hat{e}^j$  and the input length of  $\hat{f}^j$  equals the output length of  $\hat{e}^{j-1}$  (or  $\hat{d}$  in case  $j = 1$ ).

Ergo,  $\mathcal{S}$  can emulate the view during an execution of  $g$  by executing all of the simulators for  $\pi^{\hat{d}}, \pi^{\hat{f}^1}, \pi^{\hat{e}^1}, \dots, \pi^{\hat{u}}$  in sequence. We can show that  $\mathcal{S}$  is indistinguishable from the real world via a simple hybrid argument, where in the  $i^{\text{th}}$  hybrid for  $i \in \{0, 1, \dots, 2k + 1\}$ , the view of the first  $i$  operators is real and the view of the remaining operators is simulated. Each pair of adjacent hybrids is indistinguishable due to the simulation security of the  $i^{\text{th}}$  operator, and the endpoints correspond to the real and simulated views.  $\square$

The union of all leakages stated in Theorem A.4 can potentially be large; its potential for harm in practice depends strongly on the parties’ privacy concerns about their input data, as well as on the number and type of hybrid operators invoked. That having been said, our annotation propagation method ensures that even the composition of all leakages of intermediate state to the STP is no worse than simply revealing all of the columns that the regular parties are willing to share with the STP. That is, applying our recursive definition of trust sets to Theorem A.4 yields the following corollary.

**Corollary A.5.** *Suppose Conclave and its MPC backend satisfy all of the requirements listed in Theorem A.4 and that Conclave is tasked to compute function  $g$  as given in equation (1). Then, Conclave securely computes function  $g$  subject to the following leakage: every party learns the input and output lengths of  $d$  and all  $e^j$ , and the STP receives the contents of all input columns annotated by the regular parties.*

*Proof.* The only difference between this theorem and Theorem A.4 is the leakage to the STP. We assert without proof the correctness of Conclave’s manual specification of the column dependency for each operator, and focus on security.

Conclave deploys a hybrid join (respectively, hybrid aggregation) only if the STP is in the trust set of the key (resp., group-by) column of the hybrid operator’s result. Based on Conclave’s recursive definition of the “trust set” of an intermediate operator, it follows that there exists a propagation

algorithm  $\mathcal{I}$  that, when given all input data that has been annotated for sharing with the STP, can calculate sufficiently many columns of the DAG to derive all join key (respectively, group-by) columns of the result and all operands for the hybrid join (resp., hybrid aggregation). Hence, we can construct a simulator  $\mathcal{S}$  to satisfy this theorem simply by running  $\mathcal{I}$  on the leaked input data to reconstruct the leakage required to run the simulator within Theorem A.4.  $\square$

## A.5 Malicious Security without Hybrid Operators

Unlike the rest of this paper, in this section we briefly and informally examine Conclave’s ability to withstand malicious attack. With a few simple modifications, Conclave can provide malicious security up to abort and without guaranteed output delivery or fairness.

To begin, we examine where a malicious actor  $\mathcal{A}$  could render moot the analysis of Theorem A.2 by deviating from the secure computation of  $g = u \circ f \circ d$ . There exists a more complex version of Lemma A.1 that provides malicious security as long as both  $\pi^f$  and  $\pi^{g \circ f}$  can withstand malicious attacks [29, §7.4.2]. To compute  $f$  securely, Conclave simply needs to include a backend that withstands malicious actors. Ergo, the only remaining question is the security of  $\pi^{g \circ f}$  (i.e., the calculations of  $u$  and  $d$ ) under malicious attack.

For the post-processing operator  $u$ : if  $\mathcal{A}$  doesn’t corrupt the receiving party then she has no way to influence  $u$ . If  $\mathcal{A}$  does corrupt the receiving party then she can influence  $u$  arbitrarily or refuse to calculate it altogether, thereby eliminating any chance of guaranteeing output delivery or fairness. However, because  $u$  is a local computation any of these deviations must be simulatable so  $\mathcal{A}$  cannot learn any new information from deviating at this stage.

For the pre-processing operator  $d$ :  $\mathcal{A}$  could potentially learn information if a malformed calculation of  $d_i(x_i)$  is fed into subsequent operators, so we must ensure that the local operators are calculated correctly. Because each party has the right to choose its own input  $x_i$ , it suffices to ensure that its calculation of  $x'_i = d_i(x_i)$  is (i) independent from the other parties’ pre-processing calculations and (ii) a value in the support of  $d_i$ . These conditions can be satisfied with a commitment scheme and a zero knowledge proof, respectively.

In summary, Conclave can provide malicious security if:

1. It uses a malicious secure backend,
2. It adds an initial round of communication during which each party commits to its local pre-processing output.
3. It ties  $d$  and  $f$  together by having each party prove in zero knowledge that its contribution to  $\pi^f$  equals the value previously committed and is in the range of  $d_i$ .

We omit formal definitions and analysis. The current Conclave prototype does not yet support malicious security.

## References

- [1] 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 (CCS)*. Vienna, Austria, Oct. 2016, pp. 805–817.
- [2] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, et al. “Optimized Honest-Majority MPC for Malicious Adversaries – Breaking the 1 Billion-Gate Per Second Barrier”. In: *Proceedings of the 38<sup>th</sup> IEEE Symposium on Security and Privacy (SP)*. May 2017, pp. 843–862.
- [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 11<sup>th</sup> Annual International Cryptology Conference (CRYPTO)*. Santa Barbara, California, USA, Aug. 1991, pp. 420–432.
- [5] Donald Beaver. “Precomputing Oblivious Transfer”. In: *Proceedings of the 15<sup>th</sup> 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 20<sup>th</sup> Annual ACM Symposium on Theory of Computing (TC)*. Chicago, Illinois, USA, May 1988, pp. 1–10.
- [7] Azer Bestavros, Andrei Lapets, and Mayank Varia. “User-centric distributed solutions for privacy-preserving analytics”. In: *Communications of the ACM* 60.2 (2017), pp. 37–39.
- [8] Dimitrios Biskas, 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] Andrea Bittau, Ulfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, et al. “Prochlo: Strong privacy for analytics in the crowd”. In: *Proceedings of the 26<sup>th</sup> Symposium on Operating Systems Principles (SOSP)*. Shanghai, China, 2017, pp. 441–459.
- [10] 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 19<sup>th</sup> International Conference on Financial Cryptography and Data Security*. San Juan, Puerto Rico, Jan. 2015, pp. 227–234.
- [11] Dan Bogdanov, Liina Kamm, Baldur Kubo, Reimo Rebane, Ville Sokk, and Riivo Talviste. “Students and Taxes: a Privacy-Preserving Study Using Secure Computation”. In: *Proceedings on Privacy Enhancing Technologies (PoPETS) 2016.3* (2016), pp. 117–135.
- [12] Dan Bogdanov, Sven Laur, and Jan Willemson. “Sharemind: A Framework for Fast Privacy-Preserving Computations”. In: *Proceedings of the 13<sup>th</sup> 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.
- [13] 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.
- [14] Christoph Bösch, Pieter H. Hartel, Willem Jonker, and Andreas Peter. “A Survey of Provably Secure Searchable Encryption”. In: *ACM Computing Surveys* 47.2 (2014), 18:1–18:51.
- [15] Elette Boyle, Kai-Min Chung, and Rafael Pass. “Large-Scale Secure Computation: Multi-party Computation for (Parallel) RAM Programs”. In: *Proceedings of the 35<sup>th</sup> Annual International Cryptology Conference (CRYPTO)*. Santa Barbara, California, USA, Aug. 2015, pp. 742–762.
- [16] 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.
- [17] 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)*. Rehovot, Israel, Jan. 2015, pp. 153–162.
- [18] Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. “Quorums Quicken Queries: Efficient Asynchronous Secure Multiparty Computation”. In: *Proceedings of the 15<sup>th</sup> International Conference on Distributed Computing and Networking (ICDCN)*. Coimbatore, India, Jan. 2014, pp. 242–256.
- [19] 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.

- [20] Daniel Demmler, Thomas Schneider, and Michael Zohner. “ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation”. In: *22nd Annual Network and Distributed System Security Symposium (NDSS)*. San Diego, California, USA, Feb. 2015.
- [21] 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*. 2016, pp. 1602–1613.
- [22] Cynthia Dwork. “Differential Privacy: A Survey of Results”. In: *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation*. Xi’an, China, 2008, pp. 1–19.
- [23] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. “Rich Queries on Encrypted Data: Beyond Exact Matches”. In: *Proceedings of the 20th European Symposium on Research in Computer Security (ESORICS)*. Vienna, Austria, Sept. 2015, pp. 123–145.
- [24] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, et al. “SoK: Cryptographically Protected Database Search”. In: *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP)*. San Jose, California, USA, May 2017, pp. 172–191.
- [25] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. “High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority”. In: *Proceedings of the 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Paris, France, Apr. 2017, pp. 225–255.
- [26] 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 (PoPETs)*. Oct. 2017, pp. 345–364.
- [27] 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.
- [28] Oded Goldreich. *The Foundations of Cryptography – Volume 1, Basic Techniques*. Cambridge University Press, 2004.
- [29] Oded Goldreich. *The Foundations of Cryptography – Volume 2, Basic Applications*. Cambridge University Press, 2004.
- [30] 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.
- [31] 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.
- [32] 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.
- [33] CHIP Center for Health Information Partnerships. *HealthLNK Data Repository (HDR)*. 2014. URL: <http://www.healthinformationforall.org/healthlnk-overview/>.
- [34] Albert O. Hirschman. “The Paternity of an Index”. In: *The American Economic Review* 54.5 (1964), pp. 761–762.
- [35] 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* (July 2017). IACR: 2017/738.
- [36] 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.
- [37] Roman Jagomägis. “SecreC: a privacy-aware programming language with applications in data mining”. In: *Master’s thesis, University of Tartu* (2010).
- [38] 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.
- [39] 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.
- [40] Seny Kamara and Tarik Moataz. *SQL on Structurally-Encrypted Databases*. 2016. IACR: 2016/453.
- [41] Seny Kamara, Payman Mohassel, and Mariana Raykova. “Outsourcing Multi-Party Computation”. In: *IACR Cryptology ePrint Archive* 2011 (2011), p. 272. IACR: 2011/272.
- [42] 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.

- [43] Seny Kamara, Payman Mohassel, and Ben Riva. “Salus: a system for server-aided secure function evaluation”. In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. 2012, pp. 797–808.
- [44] Florian Kerschbaum. “Expression Rewriting for Optimizing Secure Computation”. In: *Proceedings of the 3<sup>rd</sup> ACM Conference on Data and Application Security and Privacy (CODASPY)*. San Antonio, Texas, USA, 2013, pp. 49–58.
- [45] Peeter Laud. “Parallel Oblivious Array Access for Secure Multiparty Computation and Privacy-Preserving Minimum Spanning Trees”. In: *Proceedings on Privacy Enhancing Technologies (PoPETS) 2015.2* (2015), pp. 188–205.
- [46] Yehuda Lindell and Benny Pinkas. “Secure multiparty computation for privacy-preserving data mining”. In: *Journal of Privacy and Confidentiality* 1.1 (2009), p. 5.
- [47] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. “OblivM: A Programming Framework for Secure Computation”. In: *Proceedings of the 2015 IEEE Symposium on Security and Privacy (SP)*. San Jose, California, USA, May 2015, pp. 359–376.
- [48] 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.
- [49] Arjun Narayan and Andreas Haeberlen. “DJoin: Differentially Private Join Queries over Distributed Databases”. In: *Proceedings of the 10<sup>th</sup> USENIX Conference on Operating Systems Design and Implementation (OSDI)*. Hollywood, California, USA, Oct. 2012, pp. 149–162.
- [50] Kartik Nayak, Xiao Shaun Wang, Stratis Ioannidis, Udi Weinsberg, Nina Taft, and Elaine Shi. “GraphSC: Parallel Secure Computation Made Easy”. In: *Proceedings of the 2015 IEEE Symposium on Security and Privacy (SP)*. San Jose, California, USA, May 2015, pp. 377–394.
- [51] New York City Taxi & Limousine Commission. *Taxicab Factbook*. 2014. URL: [http://www.nyc.gov/html/tlc/downloads/pdf/2014\\_taxicab\\_fact\\_book.pdf](http://www.nyc.gov/html/tlc/downloads/pdf/2014_taxicab_fact_book.pdf).
- [52] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. “Pig Latin: A Not-So-Foreign Language for Data Processing”. In: *Proceedings of SIGMOD*. 2008, pp. 1099–1110.
- [53] Antonis Papadimitriou, Arjun Narayan, and Andreas Haeberlen. “DStress: Efficient Differentially Private Computations on Distributed Data”. In: *Proceedings of the 12<sup>th</sup> European Conference on Computer Systems (EuroSys)*. Belgrade, Serbia, 2017, pp. 560–574.
- [54] 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)*. Berkeley, California, USA, May 2014, pp. 359–374.
- [55] Tal Rabin and Michael Ben-Or. “Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract)”. In: *Proceedings of the 21<sup>st</sup> Annual ACM Symposium on Theory of Computing (TC)*. Seattle, Washington, USA, May 1989, pp. 73–85.
- [56] 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.
- [57] Todd W. Schneider. *NYC taxi trip data*. <https://github.com/toddwschneider/nyc-taxi-data>. Accessed 03/08/2016.
- [58] Adi Shamir. “How to share a secret”. In: *Communications of the ACM* 22.11 (1979), pp. 612–613.
- [59] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, et al. “Hive – A Warehousing Solution over a Map-Reduce Framework”. In: *Proceedings of the VLDB Endowment* 2.2 (2009), pp. 1626–1629.
- [60] 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.
- [61] 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.
- [62] U.S. Office of the Comptroller of the Currency. URL: <https://www.occ.treas.gov/>.
- [63] U.S. Social Security Administration. *Social Security FAQs*. Q20. URL: <https://www.ssa.gov/history/hfaq.html>.
- [64] Nikolaj Volgushev, Malte Schwarzkopf, Andrei Lapets, Mayank Varia, and Azer Bestavros. “Conclave: Secure Multi-Party Computation on Big Data”. In: *Proceedings of the 14<sup>th</sup> ACM European Conference on Computer Systems (EuroSys)*. Mar. 2019.
- [65] Andrew C. Yao. “Protocols for Secure Computations”. In: *Proceedings of the 23<sup>rd</sup> Annual Symposium on Foundations of Computer Science (AFCS)*. Washington, DC, USA, 1982, pp. 160–164.
- [66] 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 8<sup>th</sup> USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. San Diego, California, USA, Dec. 2008.

- [67] 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 9<sup>th</sup> USENIX Conference on Networked Systems Design and Implementation (NSDI)*. San Jose, California, USA, Apr. 2012, pp. 15–28.
- [68] Samee Zahur and David Evans. *Obliv-C: A Language for Extensible Data-Oblivious Computation*. <http://eprint.iacr.org/2015/1153>. 2015. IACR: 2015/1153.
- [69] 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 14<sup>th</sup> USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Boston, Massachusetts, USA, 2017, pp. 283–298.