Scalable Oblivious Data Analytics

D2.2 Application-Oriented MPC Protocols

Mark Abspoel (PHI), Sakina Asadova (PHI), Frank Blom (TUE), Niek J. Bouman (TUE), Tore Kasper Frederiksen (ALX), Paul Koster (PHI), Berry Schoenmakers (TUE), Meilof Veeningen (PHI), Nikolaj Volgushev (ALX), Niels de Vreede (TUE)

The project SODA has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731583.

Project Information

Scalable Oblivious Data Analytics

Project number:	731583
Strategic objective:	H2020-ICT-2016-1
Starting date:	2017-01-01
Ending date:	2019-12-31
Website:	https://www.soda-project.eu/

Document Information

Title: D2.2 Application-Oriented MPC Protocols

ID:	D2.2	Type:	R	Dissemination level:	PU
		Month:	M21	Release date:	30.09.2018

Contributors, Editor & Reviewer Information

Contributors (person(partner): sections)	Mark Abspoel (PHI): Sect. 2 Sakina Asadova (PHI): Sect. 3 Frank Blom (TUE): Sect. 1 Niek J. Bouman (TUE): Sect. 2,1 Tore Kasper Frederiksen (ALX): Sect. 6 Paul Koster (PHI): Sect. 3 Berry Schoenmakers (TUE): Sect. 3,4 Meilof Veeningen (PHI): Sect. 3 Nikolaj Volgushev (ALX): Sect. 5 Niels de Vreede (TUE): Sect. 2
Editor (person/partner)	Niek J. Bouman (TUE), Frank Blom (TUE)
Reviewer (person/partner)	Paul Koster(PHI), Prastudy Fauzi(AU)

Release number	**Date issued**	**SVN version**	**Release description / changes made**
1	2018-09-30	Rev. 692	Initial release

SODA Consortium

Full Name	**Abbreviated Name**	**Country**
Philips Electronics Nederland B.V.	PHI	Netherlands
Alexandra Institute	ALX	Denmark
Aarhus University	AU	Denmark
Göttingen University	GU	Germany
Eindhoven University of Technology	TUE	Netherlands

Table 1: Consortium Members

Executive summary

This WP2 deliverable contains research results on the topic of secure multiparty computation and have been obtained during the first half of the SODA project. The scope of this deliverable is application-oriented, with a strong focus on data mining applications. Our scope includes MPC primitives specifically developed with an application in mind, building blocks from applied mathematics that are particularly relevant for data mining (such as linear algebra), as well as application-specific research work (e.g., the evaluation of a neural network in MPC). The main objective is to devise optimized multiparty computation protocols that enable or make a step forward towards the application of MPC to large volumes of data. In the light of the above objective, this deliverable presents several contributions; below we give a summary per chapter.

Secure Linear Algebra over $\mathbb{Q}$. Suppose that A is some non-singular matrix, belonging to the ring of n-by-n matrices with integer coefficients. We are interested, when given a secret sharing $[\![A]\!]$ of the entries of A, in securely computing the inverse $[\![A^{-1}]\!]$ or securely solving a linear system $AX = B$, given an n-by-m matrix B with integer coefficients, for the unknown matrix X. In general, A^{-1} is an element of $\mathbb{Q}^{n\times n}$, which poses a challenge since secure multiparty computation naturally offers arithmetic in a finite field. One approach to deal with this "mismatch" would be to apply *rational reconstruction* [110, 111, 25]. Unfortunately, performing rational reconstruction *obliviously* would be rather costly in terms of computation and round complexity.

In this work we present an approach, inspired by integer-preserving Gaussian elimination [6], to securely compute the inverse of A over $\mathbb{Q}$ without using rational reconstruction. The main idea is to circumvent the occurrence of rational entries in the result by representing the inverse as the pair of the adjugate and the determinant of A, which are both integer-valued whenever A is integer-valued. Our protocol is inspired by Cramer and Damgård's protocol [21] for secure computation of the determinant over a finite field, which builds on work by Bar-Ilan-Beaver [5]. In the final section, we present secure ridge regression as a machine-learning application of our protocol.

Fast Secure Comparison for Medium-Sized Integers. In 1994, Feige, Killian and Naor proposed a three-party[1] protocol for secure comparison in $\mathbb{F}_7$ for inputs $a, b \in [0, 2]$. Their protocol is based on the observation that the Legendre symbol of $x := a - b$, $\left(\frac{x}{p}\right)$, for $p = 7$ and for all $x \in [-2, 2]$ coincides with $\mathrm{sgn}(x)$, the sign of x. Yu applies the idea of using the Legendre symbol for secure comparison in the context of N-party MPC (for arbitrary N), and presents a constant-rounds protocol for comparing integers from the interval $[-d, d]$, where $d = O(\log p)$.

In this work, we present new quadratic-residuosity-based comparison protocols that employ some form of error-correction to increase d by a constant factor, while enjoying a one-round (resp. two-round) online phase. We investigate the size of the prime (i.e. the finite-field size) needed to compare integers from the interval $[-d, d]$ with our method, via asymptotic bounds as well as numerically. In particular, we introduce a novel way to find primes that give rise to patterns of quadratic residues and non-residues that are "noisy encodings" of the sign function, where the errors are sufficiently sparse such that our error-correction method can correct them. We demonstrate the practical relevance of our protocols by means of applying them to secure neural network evaluation.

[1] In this three-party scenario, two players provide inputs a resp. b and one "helper party" learns whether $a < b$, $a = b$ or $a > b$, but nothing beyond this.

Secure DNA String Matching. The Burrows–Wheeler transform (BWT) was introduced in 1994 [17] as a compression algorithm. Apart from compression, it can also be used to solve the approximate string-matching problem, where one aims to find sub-strings of a reference string X that approximately match a given query string W. When we consider W and X to be genomic data, solving the approximate matching problem allows one to query for specific DNA mutations. In real-world applications, a genomic query string W can be considered as secret input, since it can contain information that is specific to an illness or condition. The reference string X could also be regarded as secret, since it might be associated to a specific patient, which would make it privacy-sensitive data.

In this chapter, we present a Python implementation of BWT-based approximate string-matching utilizing the MPyC framework, which is based on Shamir secret sharing. We first analyze the details of an MPC implementation of the BWT-based approximate string matching algorithm with private query string W. Secondly, we analyze the secure algorithm where both W and the reference string X are regarded as secret input. We conclude with some experimental results.

Secure Evaluation of Convolutional Neural Networks. Convolutional neural networks (CNNs) have become a popular and effective class of machine learning algorithms. We have investigated performing classication of images of hand-written digits in MPC. In particular, we focus on an outsourcing scenario in which the parameters of the network as well as the inputs remain in secret-shared form. Keeping the parameters secret prevents theft of the model by third parties; the input images are kept secret because they may contain sensitive information, such as credit card numbers. We have designed and implemented several alternative secure evaluation of CNNs, either using integer arithmetic only or using fixed-point arithmetic. We describe a number of optimizations that we used in our implementations and we report some experimental results.

Conclave: a Secure Query Compiler. MPC makes it possible to perform data analysis jointly over large private data sets, and this application has a wide range of use cases. However, two main problems arise when integrating MPC into the "big data" analytics context: (1) data analysts typically lack MPC expertise; (2) MPC frameworks do not yet scale well to large data sizes. *Conclave* is an MPC-enabled query compiler that lets data analysts write relational queries as if they had access to all parties' data in the clear. Conclave then turns the queries into a combination of efficient local processing steps and secure MPC steps. This *hybrid* approach gives a clear speedup over MPC data processing frameworks that perform the entire computation securely. Compared to earlier hybrid solutions, Conclave has minimal annotation burden — none by default, and optional column-level annotations on input tables to improve performance.

Distributed RSA Key Generation When working on large amounts of data coming from many different clients, it is not desirable to have all the different input sources participate in an MPC execution. Ideally we would like these parties to be able to preprocess their own data and allow this to be obliviously given as input to an MPC computation carried out by a small number of servers. This is achievable if the clients can encrypt, and send to the servers, their preprocessed data using a public key encryption scheme; assuming servers have a secret sharing of the private key. However, for popular schemes such as RSA or Paillier it is not easy to construct a public key with a secret shared private key, as this requires distributed random prime generation. Our work [35], presented at CRYPTO 2018, shows how to do this for RSA keys in the crucial two-party setting. The previous state-of-the-art work in this area achieves this task, only in the semi-honest setting, in an average of 15 minutes. We manage to push this to an average of 42 seconds on a standard machine, but in the malicious setting. We achieve this by getting malicious security almost for free with a slight modification of the "standard" ideal functionality in this setting, allowing us to only require a single lightweight zero-knowledge proof at the end of the protocol to get full malicious security.

About this Document

Role of the deliverable

This deliverable contains several research results on the topic of MPC tailored to the problems in data mining, and have been obtained during the first half of the SODA project. This deliverable is part of Work Package 2.

The scope of this deliverable is application-oriented, with a strong focus on data mining applications. More precisely, our scope includes MPC primitives specifically developed with an application in mind, building blocks from applied mathematics that are particularly relevant for data mining (such as linear algebra), as well as application-specific research work (e.g., the evaluation of a neural network in MPC). Within this scope, our objective is to devise optimized multi-party computation protocols that enable or make a step forward towards the application of MPC to large volumes of data. In the light of the above objective, this deliverable presents several contributions, like protocols for solving linear systems, computing the sign of an integer in a bounded range, inexact string matching, neural network evaluation, distributed RSA key generation, and a framework for performing SQL-like queries in an MPC context.

Results of this deliverable have the potential to support proof of concept and demonstrators to be developed in WP4. What results precisely will be actually used to build on top of is undecided at this moment. A candidate for example is the work on secure comparisons for medium-sized integers, which may be leveraged in model learning algorithms for e.g. predictive analytics use cases. Similarly, insights gained with secure neural network evaluation may be leveraged in the use case for predictive models that may be using neural networks in combination with federated learning.

Relationship to other SODA deliverables

This deliverable builds upon the deliverables D1.1 and D2.1, which survey the state of the art in privacy-preserving data mining. Whereas the latter deliverables survey existing work, this deliverable presents new results. In deliverable D2.3, we plan to focus on secure streaming algorithms.

Most results presented here are accompanied with a prototype / PoC implementation, which are covered by deliverable D4.2.

Relationship to other versions of this deliverable

n/a

Structure of this document

The document contains five chapters covering recent research. Section 1 presents a new protocol for securely solving full-rank linear systems over the rationals. Section 2 introduces a new technique for performing a secure comparison for medium-sized integers in a single round of communication in the online phase, which is especially useful for fast neural-network evaluation. Section 3 presents a secure protocol for inexact string matching, which has applications in DNA matching. Section 4 presents research results of a project on secure neural network evaluation. Section 5 presents Conclave, a new query compiler that makes MPC available and usable to data analysts, by means of automatically converting SQL queries into a mix of local processing steps and distributed secure MPC steps.

Table of Contents

1 Linear Algebra over $\mathbb{Q}$ in the context of MPC

1.1 Introduction

Let $\mathscr{M}_n(\mathbb{Z})$ denote the ring of n-by-n matrices over $\mathbb{Z}$. Let $A \in \mathscr{M}_n(\mathbb{Z})$ be any non-singular matrix. We suppose that A is *secret-shared* between the players, denoted as $[\![A]\!]$, which implies that the elements of A must be represented in some finite ring or finite field. Note that the dimensions of the matrix A, along with the property that A is non-singular, is public information.

We would like to either compute the inverse of A, i.e., A^{-1}, or solve the system $Ax = b$, where $b \in \mathbb{Z}^n$ is some arbitrary vector, for the unknown vector x (of length n), or, more generally, the system $AX = B$, with $B \in \mathbb{Z}^{n\times m}$ and the unknown matrix X having the same dimension as B. In general, the inverse A^{-1} will have *rational* coefficients (in the field $\mathbb{Q}$). A^{-1} has integer coefficients if and only if $A \in \mathrm{GL}_n(\mathbb{Z})$ (the general linear group over $\mathbb{Z}$), which is the case iff $\det A \in \{-1, 1\}$. Likewise, vector x, resp. matrix X will in general have rational coefficients. In this section, we will answer the following questions.

- *How can we securely compute, and represent, secret sharings of the inverse $A^{-1} \in \mathbb{Q}^{n\times n}$ and the solution $X \in \mathbb{Q}^{n\times m}$ of a linear system $AX = B$ with $A \in \mathbb{Z}^{n\times n}$ having full rank and $B \in \mathbb{Z}^{n\times m}$, when given an arithmetic black-box (ABB), i.e., some MPC framework that provides secure arithmetic in a finite ring or field?*
- *Assuming that the above can be achieved, then how large should we choose the ring or field order (the modulus) from the ABB, in order to be able to represent a secret-shared version of the result (i.e., $[\![A^{-1}]\!]$, the solution $[\![X]\!]$, etc.) without loss of information?*

To securely represent the elements of A and perform secure operations on them, we must represent them within some finite algebraic structure. Although we must compute over this structure, the problem we aim to solve is defined over the integers and the result we obtain must correspond to the solution to the integer problem. In the remainder of this section we consider the case where the elements of A are represented in the finite field of integers modulo prime p, which we denote as $\mathbb{F}_p$. Because we must be able to identify the elements of both A and the result of our computation with the solution over the integers, this means that the modulus should be chosen "large enough". We will specify later what "large enough" should mean.

1.1.1 Related Work

Secure Linear Algebra over the Rationals and over Finite Fields. In the multiparty scenario, Toft performs secure linear algebra over $\mathbb{Q}$ for securely solving linear programs [102]. Many of the earlier works on secure linear algebra (in the multiparty case) focus on linear algebra over finite fields. Bar-Ilan and Beaver [5] propose a constant-round MPC protocol for the secure inversion of field elements and matrices. Cramer and Damgård [21] propose constant-round MPC protocols for secure computation of the determinant, characteristic polynomial, rank, and the solution space of linear systems of equations. Cramer et al. [23] improve the computational complexity (the number of secure multiplications) of securely solving a linear system of n equations in m variables with $m \geq n$ from m^5 (when using the protocol of [21]) to $n^4 + m^2 n$.

Interplay between Finite Fields and Rational Numbers. An important technique to exploit the correspondence between rational arithmetic and integer arithmetic modulo a prime is p-adic lifting, also known as *rational reconstruction*, see [110, 111, 25] and references therein. Given a computation that would yield a rational result, the idea is to perform this computation in arithmetic modulo a prime, and, as the final step, recover the numerator and denominator of the rational number by means of performing basis reduction in a two-dimensional lattice (e.g., by running the Lagrange–Gauss algorithm). Formally, a fraction a/b for integers a and b is represented in the finite field as the element $x = a \cdot b^{-1}$. As long as $|a|, |b| \leq \sqrt{p/2}$, we can uniquely reconstruct a and b from x by reducing the lattice basis $\{(p,0),(x,1)\}$, in the sense that the reduced basis will contain the vector (a,b).

More recently, researchers have realized the usefulness of this technique in secure computation [33, 58, 40], where it is not to be confused with rational (in the game-theoretic sense) reconstruction in the context of rational secret sharing [45]. In each of those works, however, the scenario is such that the rational reconstruction procedure itself can be performed "in the clear."

Applying rational reconstruction *obliviously* seems to be impractical: one would have to run $K(p)$ iterations of the Lagrange–Gauss algorithm, where $K(p)$ is the number of iterations required to reduce the worst-case input lattice basis, asymptotically $K(p) = O(\log p)$ as proved by Vallée [108].[2] Moreover, in each such iteration one has to perform a secure (non-exact) division, whose "cost" (computational and round complexity) is comparable to a secure comparison.

1.1.2 Contribution

We propose in Section 1.3 a constant-round protocol for computing the inverse of A over $\mathbb{Q}$. Our protocol is an adapted version of Cramer and Damgård's protocol for secure computation of the determinant over a finite field [21], which builds upon Bar-Ilan–Beaver's inversion protocol [5].

Conceptually, one way to obtain the inverse of A over $\mathbb{Q}$ would be to first compute the inverse of A over $\mathbb{F}_p$ (using the techniques from [5, 21]), followed by an oblivious version of rational reconstruction. As argued above, however, applying oblivious rational reconstruction would be impractical.

Our main idea, inspired by integer-preserving Gaussian elimination [6], is to circumvent the need for rational reconstruction by preventing the occurrence of rational numbers while solving a linear system. The key point is that the *adjugate* of a A, $\operatorname{adj} A$, is integer-valued whenever A is integer-valued. Hence, whenever we want to securely compute the inverse of A (or solve for X in the system $AX = B$), we omit dividing by the determinant of A and return the answer in the form of the pair $(\det A, \operatorname{adj} A)$ or, respectively, $(\det A, (\operatorname{adj} A)B)$, because it is precisely the division by $\det A$ that is responsible for introducing rational numbers in the result.

1.2 Preliminaries

Let R be a finite commutative ring, and let $n \in \mathbb{N}$ be nonzero. For any $n \times n$ matrix A with entries in R, we write $\det A$ for the determinant of A, and we define the *(i, j)-th minor of* A, denoted as $\operatorname{minor}(A)_{i,j}$, as the determinant of the submatrix of A obtained by removing the ith row and jth column of A. The *leading principal minor of order k* of A is defined as the determinant of the submatrix of A obtained by taking the first k rows and the first k columns of A. Hence, the leading principal minor of order n coincides with $\det A$. We define the *adjugate* of A as $\operatorname{adj} A$, where $(\operatorname{adj} A)_{i,j} := (-1)^{i+j} \operatorname{minor}(A)_{j,i}$, for all $1 \leq i, j \leq n$. The following bound is due to Hadamard. For any square n-by-n matrix M with coefficients $m_{i,j} \in [-B, B]$ for all $i, j \in [n]$, it holds that

$$|\det M| \leq \mathrm{Had}_B(n) := B^n n^{n/2}.$$

[2] Vallée [108] expresses her upper bound on the number of iterations of the Lagrange–Gauss algorithm in terms of the "inertia" ℓ of a basis. For rational reconstruction in the worst case, we have that $\ell = p^2 + (p-1)^2 + 1 = O(p^2)$.

Definition 1.1. We say that a square $n \times n$ matrix A is *LU-decomposable* if and only if there exists an $n \times n$ lower triangular matrix L with ones on its diagonal and an $n \times n$ upper triangular matrix U such that $A = LU$. Such L and U are called A's LU decomposition.

Proposition 1 ([51, Corollary 3.5.5])**.** *Let A be any square and invertible matrix. Then, A admits an LU factorization (i.e., without any pivoting) if and only if all its leading principal minors are nonzero.*

Proposition 2. *The probability that a uniformly random matrix $R \in \mathscr{M}_n(\mathbb{F}_p)$ has the property that all leading principal minors (i.e., including the matrix R itself) are non-zero equals*

$$P_{pivot\text{-}free} = P_{pivot\text{-}free}(n, p) := \left(1 - \frac{1}{p}\right)^n.$$

Proof. Trivially, a 1-by-1 random matrix over $\mathbb{F}_p$ is non-singular with probability $1 - 1/p$. For any $k \in \mathbb{N}, k > 1$, the probability that a uniformly random matrix in $\mathscr{M}_k(\mathbb{F}_p)$ has non-zero determinant, *when given that its leading principal minor of order $k-1$ is non-zero,* also equals $1 - 1/p$. This follows from *Schur's determinant identity*,

$$\det \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \det(A) \det(D - CA^{-1}B).$$

which we apply with A a $(k-1)$-by-$(k-1)$ matrix that is non-singular in the conditional probability space that we consider here, B a $(k-1)$-by-1 matrix, C a 1-by-$(k-1)$ matrix, and D a uniformly random field element. In this case, the term $D - CA^{-1}B$ represents just a single field element. Because D is uniformly random and (statistically) independent from $CA^{-1}B$, $D - CA^{-1}B$ is uniformly random, and non-zero with probability $1 - 1/p$. We obtain the claim by induction on k. □

In order to interpret finite-field elements as signed integers, we use the following map

$$\begin{aligned} \sigma : \mathbb{F}_p &\to \mathbb{Z} \\ x &\mapsto \begin{cases} x & \text{if } x \leq \frac{1}{2}(p-1), \\ x - p & \text{otherwise.} \end{cases} \end{aligned}$$

For any matrix $A \in \mathbb{F}_p^{n \times m}$, for arbitrary non-zero $n, m \in \mathbb{N}$, we write $\sigma(A)$ for the element-wise application of σ to A.

Let $\mathscr{X}$ be a finite set and let $\mathscr{P}_{\mathscr{X}}$ be the set of non-negative real-valued functions on $\mathscr{X}$. *Statistical distance* is the function defined as

$$\begin{aligned} \mathrm{SD} : \mathscr{P}_{\mathscr{X}} \times \mathscr{P}_{\mathscr{X}} &\to \mathbb{R} \\ (p, q) &\mapsto \tfrac{1}{2} \sum_{x \in \mathscr{X}} |p(x) - q(x)|. \end{aligned}$$

The name *statistical distance* stems from its typical use as a distance measure for probability distributions. For random variables X and Y that have the same range, we will also write $\mathrm{SD}(X, Y)$, where the latter should be understood as the statistical distance between their distributions P_X and P_Y.

1.3 Computing $\det A$ and $\operatorname{adj} A$ via Random Self-Reducibility

In the context of matrices, the basic idea of Bar-Ilan–Beaver's inversion protocol is that the map $f : \mathrm{GL}(\mathbb{F}_p) \to \mathrm{GL}(\mathbb{F}_p)$, $A \mapsto MA$ is a bijection for any matrix $M \in \mathrm{GL}(\mathbb{F}_p)$. Hence, by sampling a matrix $[\![G]\!]$ uniformly at random from $\mathrm{GL}(\mathbb{F}_p)$, we can open the product $[\![GA]\!] = [\![G]\!][\![A]\!]$, without revealing anything about A (beyond the assertion that A is invertible over $\mathrm{GL}(\mathbb{F}_p)$, which we anyway assume is public information).

This approach might be viewed as a random self-reduction [32] (hence the title of this section), because we reduce (in polynomial time) our input problem instance to a random problem instance that is statistically independent of the input instance. The main advantage of this "detour" is that the bulk of the work required to actually solve a linear system can now be performed in the clear.

Up to the opening of GA, our protocol actually coincides with the original inversion protocol of Bar-Ilan and Beaver. After opening GA, Bar-Ilan–Beaver's protocol proceeds by computing the inverse of GA over $\mathbb{F}_p$, however, we must proceed differently because the inverse of $\sigma(GA)$ over $\mathbb{Q}$ is most probably not directly[3] representable in $\mathbb{F}_p$. Instead, we will compute the determinant and adjugate of GA separately. The details can be found in Protocol 1.

Similar to [21], we assume that there exists a protocol Π_0 that is parameterized by $n, p \in \mathbb{N}$ and produces (in constant rounds) the pair $([\![G]\!], [\![\det G]\!])$, where G is chosen uniformly at random from $\mathrm{GL}_n(\mathbb{F}_p)$.

Protocol 1 $([\![\operatorname{adj} A]\!], [\![\det A]\!]) \leftarrow \mathrm{AdjDet}([\![A]\!])$, $[\![A]\!] \in \mathbb{F}_p^{n\times n}$, $[\![\operatorname{adj} A]\!] \in \mathbb{F}_p^{n\times n}$, $[\![\det A]\!] \in \mathbb{F}_p$

```
1: Run Protocol $\Pi_0$ with parameters $(n, p)$, to obtain a matrix $[\![G]\!] \in \mathrm{GL}_n(\mathbb{F}_p)$ and its determinant
   $[\![\det G]\!] \in \mathbb{F}_p$.
2: Compute $[\![d]\!] := [\![\det G]\!]^{-1}$.
3: Compute $[\![GA]\!] = [\![G]\!][\![A]\!]$, and open this product, i.e., $GA \leftarrow [\![GA]\!]$.
4: Compute $\operatorname{adj} GA$ and $\det GA$ (both "in the clear").
5: Output $[\![\det A]\!] := (\det GA) \cdot [\![d]\!]$ and $[\![\operatorname{adj} A]\!] = (\operatorname{adj} GA)[\![G]\!] \cdot [\![d]\!]$.
```

Remark. Besides hiding A, the matrix G also acts as a generic-rank-profile preconditioner for A [20], hence if the (non-oblivious) computation of $\operatorname{adj} GA$ and $\det GA$ is performed via integer-preserving Gaussian elimination, then pivoting can be safely omitted.

1.3.1 An Improved Protocol for Π_0

Although we could instantiate Protocol Π_0 with the protocol as given by Cramer and Damgård [21], we present a slightly improved version in Protocol 2.

Protocol 2 improves upon Cramer and Damgård's protocol Π_0 in three ways: (i) our protocol samples n fewer random elements because every element of the diagonal of L is set to one, (ii) our protocol saves some secure multiplications, because the secure computation of the determinant now only involves a product of n values instead of $2n$ values, and in our protocol the first row of R is actually equal to the first row of U (hence it need not be computed), and (iii) the use of the unique LU decomposition (with ones on L's diagonal) gives a simpler proof.

Remark. Sampling a secret-shared value from $\mathbb{F}_p^*$ uniformly at random with perfect correctness can be achieved as follows. Sample two secret-shared values $[\![a]\!], [\![b]\!] \in \mathbb{F}_p$ independently and uniformly

[3]As opposed to an "indirect" representation of some rational number f as a modular image, from which the numerator and denominator of f can be recovered via rational reconstruction [110, 111].

Protocol 2 $(\llbracket R \rrbracket, \llbracket \det R \rrbracket) \leftarrow \mathrm{RandMatDet}(n)$, $\llbracket R \rrbracket \in \mathbb{F}_p^{n\times n}$, $\llbracket \det R \rrbracket \in \mathbb{F}_p$

```
1: Sample $n$ secret-shared elements from $\mathbb{F}_p^*$ uniformly at random and independently. We denote those elements as $\llbracket r_1^* \rrbracket, \dots, \llbracket r_n^* \rrbracket$.
2: Sample $n^2 - n$ secret shared values $\llbracket r_{i,j} \rrbracket \in \mathbb{F}_p$ uniformly at random and independently, for all $1 \le i, j \le n$ such that $i \ne j$.
3: Let $\llbracket L \rrbracket \in \mathbb{F}_p^{n\times n}$ be obtained by placing ones on its diagonal, the value $\llbracket r_{i,j} \rrbracket$ in position $i, j$ for all $i, j$ such that $2 \le i \le n$ and $1 \le j < i$, and zeros elsewhere.
4: Let $\llbracket U \rrbracket \in \mathbb{F}_p^{n\times n}$ be obtained by placing the value $\llbracket r_i^* \rrbracket$ on the diagonal position $i, i$ for all $i$ such that $1 \le i \le n$, the value $\llbracket r_{i,j} \rrbracket$ in position $i, j$ for all $i, j$ such that $2 \le i \le n$ and $i < j \le n$, and zeros elsewhere.
5: Output $\llbracket R \rrbracket = \llbracket LU \rrbracket = \llbracket L \rrbracket \llbracket U \rrbracket$ and $\llbracket \det R \rrbracket = \prod_{i=1}^n \llbracket r_i^* \rrbracket$.
```

at random and open their product $z \leftarrow \llbracket ab \rrbracket$. If $z \ne 0$, output $\llbracket a \rrbracket$ and discard $\llbracket b \rrbracket$. Otherwise, repeat this procedure (rejection sampling). On the other hand, if p is chosen such that the statistical distance between random variables uniformly distributed over $\mathbb{F}_p$ respectively $\mathbb{F}_p^*$ is negligible, and we accept to sacrifice the perfect correctness property, then may of course just sample a random value from $\mathbb{F}_p$.

1.3.2 Analysis

Proposition 3. *For all $n \in \mathbb{N}$, Protocol 2 outputs a matrix R which, when viewed as a random variable, is uniformly distributed over all invertible $n \times n$ LU-decomposable matrices.*

Proof. Let $n \in \mathbb{N}$ be arbitrary. Let $\mathscr{L}^1$ be the set of all $n \times n$ lower-triangular matrices over $\mathbb{F}_p$ with ones on its diagonal, let $\mathscr{U}$ be the set of all invertible $n \times n$ upper-triangular matrices over $\mathbb{F}_p$, and let $\mathscr{R}$ be the set of invertible $n \times n$ LU-decomposable matrices over $\mathbb{F}_p$.

It follows immediately from the protocol description that L, as a random variable, is uniformly distributed over $\mathscr{L}^1$. Furthermore, U, as a random variable, is uniformly distributed over $\mathscr{U}$ and independent from L; invertibility of U is guaranteed because the diagonal elements are sampled from $\mathbb{F}_p^*$, hence $\det U$ (the product of all diagonal elements) cannot become zero. By Definition 1.1 the map $\mathscr{L}^1 \times \mathscr{U} \to \mathscr{R}, \quad (L, U) \mapsto LU = R$ is a bijection, which proves the claim. $\square$

Protocol 1 prescribes G as a matrix that is uniformly distributed over $\mathrm{GL}_n(\mathbb{F}_p)$, whereas Protocol 2 (and similarly, protocol Π_0 in [21]), produces a matrix that is uniformly distributed over the set $\mathscr{R}$. As already noted in [21], this difference does not cause a problem as long as n is negligible compared to p. Below, we repeat this argument formally.

Corollary 4. *Let $\mathscr{R}$ be the set of invertible $n \times n$ LU-decomposable matrices over $\mathbb{F}_p$ and let R be a random variable uniformly distributed over $\mathscr{R}$. Let G be a random variable uniformly and independently distributed over $\mathrm{GL}_n(\mathbb{F}_p)$. Then,*

$$\mathrm{SD}(R, G) \le 1 - \left(1 - \frac{1}{p}\right)^n.$$

Proof.

$$\mathrm{SD}(R,G) = \frac{1}{2} \sum_{x \in \mathscr{M}_n(\mathbb{F}_p)} |P_R(x) - P_G(x)| = \sum_{x \in \mathscr{R}} \left| \frac{1}{|\mathscr{R}|} - \frac{1}{|\mathrm{GL}_n(\mathbb{F}_p)|} \right|$$
$$= 1 - \frac{|\mathscr{R}|}{|\mathrm{GL}_n(\mathbb{F}_p)|} \le 1 - \frac{|\mathscr{R}|}{|\mathscr{M}_n(\mathbb{F}_p)|} = 1 - \left(1 - \frac{1}{p}\right)^n,$$

where the second equality follows because $\mathscr{R} \subseteq \mathrm{GL}_n(\mathbb{F}_p)$. (This holds because for any matrix A, the determinant of A is itself a leading principal minor.) The last equality follows from Proposition 2, which applies here because Proposition 1 states that a matrix is in $\mathscr{R}$ if and only if all its leading principal minors are nonzero. $\square$

1.3.3 Choosing the Field Size

As a necessary condition for correctness of Protocol 1, the results produced by the protocol (the determinant and elements of the adjugate) should be uniquely representable in the field $\mathbb{F}_p$. When given an upper bound B on the magnitude of the elements of A, i.e., $|a_{ij}| \leq B$ for all $i, j \in [n]$, then a (tight) lower bound on p follows immediately from the Hadamard bound:

$$p \geq 2\,\mathrm{Had}_B(n) + 1,$$

where the application of the map $x \mapsto 2x+1$ accommodates for negative numbers and the zero element. Note that we use here the fact that the Hadamard bound also applies to the elements of the adjugate of A (which are themselves determinants) with exactly the same parameters as used for bounding the determinant of A.

2 Fast Secure Comparison for Medium-Sized Integers and Its Application in Binarized Neural Networks

2.1 Introduction

Secure integer comparison has been a primitive of particular interest since the inception of multiparty computation (MPC). In 1982, even before general multiparty computation had been realized, Yao introduced the *Millionaires' Problem* [113], where two millionaires want to determine who of them has greater wealth without revealing any information beyond the outcome of this comparison to each other or to any third party. Secure comparison has been investigated extensively since. A whole range of solutions is available with every solution aiming for a particular trade-off. Nonetheless, with respect to arithmetic-secret-sharing-based MPC, secure comparison remains among the most expensive basic operations in terms of round complexity. Hence, for applications that require many comparisons, achieving high throughput (important for privacy-preserving data processing applications) or low latency (crucial for certain applications, like blind auctions for real-time advertisement sales) can be challenging.

2.1.1 Related Work

Whereas most secure comparison protocols work over finite fields of arbitrary order, Yu [115] presents a comparison protocol that only works for specifically chosen prime moduli. Although this clearly poses a restriction in terms of applicability, the main benefit is that the specifically chosen prime modulus p enables Yu to perform a comparison *in a single round of communication* in the online phase (the offline preprocessing phase requires three communication rounds), albeit in a range of size $O(\log p)$. Namely, he chooses p such that the pattern of quadratic residues and non-residues modulo p coincides with the sign function on a given interval symmetric around zero, which is an idea that goes back to a protocol of Feige, Killian and Naor [31], who use it to compute the sign of an element $x \in [-2,2]$ in $\mathbb{F}_7$. Yu's comparison protocol for comparing arbitrary elements $a,b \in \mathbb{F}_p$ essentially works by breaking up the full-range comparison into several medium-range comparisons of the above type by performing a digit decomposition.

2.1.2 This Paper

In this paper, we pursue the line of work initiated by Yu [115]. Our main contribution is that we improve the hidden constant of Yu's [115] $O(\log p)$-range comparison method. Concretely, we propose a protocol that, for a fixed prime-length, achieves close to a *two-fold increase of the comparison range* (over Yu's results), while still enjoying a single-round online phase, at the cost of a constant amount of additional communication and some additional local computations. Also, we present a two-online-rounds protocol that achieves more than a *three-fold increase in the comparison range* when compared to Yu's approach. In other words, to compare two integers that lie in a given range (symmetric around zero), the size of the required prime modulus is a constant factor smaller than the prime required for the protocol from [115]. Keeping the finite-field modulus as small as possible or within the machine's word size could be important, for example, in a setting where MPC protocols run on constrained hardware platforms. On such platforms, the complexity of prime-field arithmetic (which is directly related to the prime size) can have a significant impact on the runtime performance. Our protocols can be found in Section 2.5 of this work.

The main idea is to somewhat relax the constraints on the prime modulus p: instead of requiring that the Legendre symbols of *all* elements in the interval $[-d,d]$, for a given positive integer d, coincide with the sign function, we only require this coincidence for *most* elements (in a specific sense). Let us, for some fixed prime p, say that there is an *error* at position $x \in [-d,d]$ if $\left(\frac{x}{p}\right) \neq \operatorname{sgn}(x)$. Our improvement is based on exploiting a "local redundancy" property enjoyed by the sign function that lets us correct such errors as long as they are sufficiently "sparse", by means of inspecting also the Legendre symbols of some neighboring positions and then performing a majority vote.

This new approach raises the question of how to find primes that give rise to increased ranges. In Section 2.4, we present some results that considerably simplify this search, and in Section 2.7 we give an explicit list of suitable primes for various bit lengths.

2.1.3 Application: Efficient Neural Network Evaluation in MPC

To demonstrate the practical value of our work, we apply our new comparison protocol to the problem of securely evaluating a neural network, in which the sign function is used as non-linearity. We use a binarized multi-layer perceptron (BMLP) for recognizing handwritten digits, as described in [52], which is trained (in the clear) on the well-known MNIST handwritten-digits data set. We consider an MPC scenario in which the input images are secret-shared between the players, which then securely evaluate the BMLP to obtain the estimated digit in secret-shared form.

2.2 Preliminaries

Arithmetic Black Box. We suppose that we are given a secure arithmetic black-box (ABB) functionality that can securely evaluate multiplication and linear forms over the finite field $\mathbb{F}_p$. We write $[x]$ to mean that we have a residue class $x \in \mathbb{F}_p$ encrypted under the ABB (e.g. x is secret-shared among a set of players, or perhaps encrypted under some homomorphic encryption scheme). Abusing notation, for small $x \in \mathbb{F}_p$ we will also refer to x as an integer in $\mathbb{Z}$, given as the canonical lift of the residue class to the integers $\{-\lfloor \frac{p}{2} \rfloor, \dots, \lfloor \frac{p}{2} \rfloor\}$.

Sign vs. Greater-or-Equal-to-Zero. The sign function and the GEZ ("greater-than-or-equal-to-zero") function are respectively defined as

$$\operatorname{sgn}(z) := \begin{cases} 1 & \text{if } z > 0, \\ 0 & \text{if } z = 0, \\ -1 & \text{if } z < 0. \end{cases} \qquad \mathsf{GEZ}(z) := \begin{cases} 1 & \text{if } z \geq 0, \\ -1 & \text{if } z < 0. \end{cases}$$

Comparing two integers a and b is achieved by evaluating the sign (or GEZ) of their difference $a - b$. The sgn function gives rise to a three-way comparison, while the GEZ function corresponds to two-way comparison. In this paper, we will start our analysis in terms of the sgn function, but for reasons that will become clear later our protocols evaluate the GEZ function (i.e. achieve two-way comparison). We will sometimes be a bit sloppy and use the word "sign" also for the GEZ function; the precise meaning should nonetheless still be clear from its context.

The Legendre symbol. Recall that the Legendre symbol of an integer a is defined as the integer

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } a \equiv 0 \pmod{p}, \\ 1 & \text{if } a \text{ is a quadratic residue modulo } p, \\ & \text{i.e. } \exists w \text{ such that } w^2 \equiv a \pmod{p}, \\ -1 & \text{otherwise.} \end{cases}$$

The Legendre symbol is a completely multiplicative function, i.e. $\left(\frac{a}{p}\right)\left(\frac{b}{p}\right) = \left(\frac{ab}{p}\right)$ for all $a,b \in \mathbb{Z}$. The equality $\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}$ holds for all primes p and all $a \in \mathbb{Z}$, and is known as *Euler's criterion*. The *law of quadratic reciprocity* asserts that for odd primes p and q,

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}.$$

Securely evaluating Legendre symbols. In principle, we can securely evaluate the Legendre symbol via Euler's criterion, which would require $O(\log p)$ secure multiplications. The complete multiplicativity of the Legendre symbol enables the following constant-rounds protocol for securely evaluating the Legendre symbol in the preprocessing model with an single-round online phase. In the preprocessing phase, we generate a secret-shared pair $([r], [(\frac{r}{p})])$ of a random non-zero class r together with its Legendre symbol. In the online (input-dependent) phase, we securely multiply $[a] \cdot [r]$, open the result and then compute

$$\left[\left(\frac{a}{p}\right)\right] = \left(\frac{ar}{p}\right)\left[\left(\frac{r}{p}\right)\right].$$

Note that the security of the protocol requires that $a \not\equiv 0 \pmod{p}$, which should be taken into account when using this protocol.

Blum primes. A prime p for which $p \equiv 3 \pmod{4}$ is called a *Blum prime*. By Euler's criterion, -1 is a quadratic non-residue modulo p if and only if p is a Blum prime. Hence, for any Blum prime p, the map $x \mapsto \left(\frac{x}{p}\right)$ is an odd function for $x \in [-\lfloor p/2 \rfloor, \lfloor p/2 \rfloor]$ (which follows immediately from the multiplicativity property of the Legendre symbol), i.e. it enjoys the same symmetry around the origin as the sign function.

2.3 Evaluating the Sign Function using Legendre Symbols

2.3.1 A Redundancy Property of the Sign Function

In this section we show that the sign function enjoys a "local redundancy" property, which lets us correct sign-flip errors by means of majority-decoding as long as those errors occur sparsely (in a sense defined below).

Definition 2.1. Let $k \geq 0$ be an integer, and let $\mathscr{T} = [t_1, t_2]$ be an interval of integers with $t_2 - t_1 \geq 2k+1$. We say that a function $e : \mathscr{T} \to \{0,1\}$ is an *error function on* $\mathscr{T}$ *admissible for* k if $e(x) = 0$ for all $x \in [-(k+1), k+1] \cap \mathscr{T}$ and if $\sum_{i=-k}^{k} e(y+i) \leq k$ holds for all $y \in [t_1+k, t_2-k]$.

Lemma 5. *Let k and $\mathscr{T}$ be as in Definition 2.1, and let e be an error function on $\mathscr{T}$ admissible for k. Then,*

$$\operatorname{sgn}\left(\sum_{i=-k}^{k} (-1)^{e(x+i)} \operatorname{sgn}(x+i)\right) = \operatorname{sgn}(x)$$

holds for all $x \in [t_1+k, t_2-k]$.

The proof will clarify why we require in Definition 2.1 that an admissible error function $e(x)$ has an "error-free" region around $x = 0$; informally speaking, the reason is that the sign function undergoes its sign change at $x = 0$, which means that there is "less room" for errors under majority-decoding in this region.

Proof. We will prove the statement for $\mathscr{T} = [-a, a]$ where $a \geq k$ is any integer. This implies the claim for any subinterval of $\mathscr{T}$ of cardinality at least $2k+1$. Note that because of symmetry (in the sign function as well as in the definition of an admissible error function), it suffices to prove the statement for $x \geq 0$. We make a case distinction on x. First, suppose that $x = 0$. Because $x = 0 \iff \operatorname{sgn}(x) = 0$, we prove this case as $\sum_{i=-k}^{k}(-1)^{e(i)}\operatorname{sgn}(i) = \sum_{i=-k}^{k}\operatorname{sgn}(i) = 0$, where the first equality follows because e is admissible for k and the second equality follows from the fact that summing an odd function over an interval symmetric around zero gives the value zero.

Second, suppose that $x > k$. We have

$$\sum_{i=-k}^{k}(-1)^{e(x+i)}\operatorname{sgn}(x+i) = \sum_{i=-k}^{k}(-1)^{e(x+i)} > 0,$$

where the equality follows because $\operatorname{sgn}(x+i) = 1$ for all $i \in [-k, k]$ and the inequality follows because e is admissible for k.

For the third (and final) case, suppose that $x \in [1, k]$. We have

$$\begin{aligned}\sum_{i=-k}^{k}(-1)^{e(x+i)}\operatorname{sgn}(x+i) &= \sum_{i=-k}^{k-x+1}\operatorname{sgn}(x+i) + \sum_{i'=k-x+2}^{k}(-1)^{e(x+i')}\operatorname{sgn}(x+i') \\ &= \sum_{j=x-k}^{k+1}\operatorname{sgn}(j) + \sum_{j'=k+2}^{k+x}(-1)^{e(j')}\operatorname{sgn}(j') \\ &= 1 + x + \sum_{j'=k+2}^{k+x}(-1)^{e(j')} \\ &\geq (1+x) + (1-x) = 2\end{aligned}$$

□

2.3.2 Counting admissible error functions

How many admissible error functions can there be on a given set $\mathscr{T}$ and for a given integer k? For the case $k = 1$, we have the following result.

Proposition 6. *Let $\mathscr{T} := [-t, t]$ with $t \geq 2$. Let $\mathscr{E}_1$ be the set of all error functions on $\mathscr{T}$ admissible for $k = 1$. Then,*

$$|\mathscr{E}_1| \leq 2^{2\lambda(t-2)}, \quad \lambda \approx 0.55146\ldots$$

The proof will make use of the following lemma, which is well known.

Lemma 7 ("Volume Bound for Hamming Balls"). *Let $\mathscr{S}_n$ be the set of all bit sequences of length n produced using by concatenating bits from a binary source X. Then, it holds that*

$$2^{H(X)n-o(n)} \leq |\mathscr{S}_n| \leq 2^{H(X)n}$$

where $H(X)$ denotes the Shannon entropy of X.

In Figure 1, we show a finite state machine (FSM) that defines a language of binary strings such that every string in the language has the following property: for every window of three consecutive bit positions, at most one position will be "1". If we impose a probability distribution on the outgoing edges of state (a), then we can view the FSM as a random binary source, which lets us compute the entropy which we need to apply the above lemma.

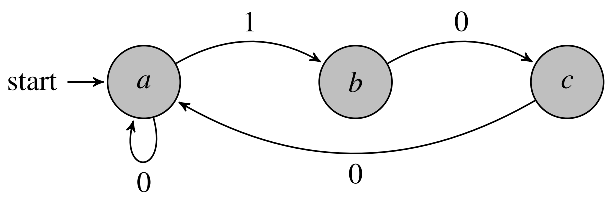


Figure 1: A finite state machine that generates binary strings with the property that for every window of three consecutive bit positions, at most one position will be "1".

Proposition 8. *The average per-bit entropy $H(X)$ of a random binary source X that produces bits according to the FSM depicted in Figure 1, where the outgoing path at state (a) ("0" or "1") is chosen uniformly at random, is given by*

$$\begin{aligned} H(X) &= \lim_{n\to\infty} \frac{H(X_1X_2\ldots X_n)}{n} \\ &= \log_2 \frac{1}{3}\left(1+\sqrt[3]{\frac{1}{2}\left(29-3\sqrt{93}\right)}+\sqrt[3]{\frac{1}{2}\left(29+3\sqrt{93}\right)}\right) \\ &\approx 0.55146\ldots \end{aligned}$$

Proof. As proved by Shannon [95], we can calculate the average per-symbol entropy of a FSM as the logarithm of the largest real root of the equation

$$\det(z^{-1}D-I)=0,$$

where D is the transition matrix of the FSM. We can easily draw up the transition matrix from Figure 1:

$$D=\begin{bmatrix}1&1&0\\0&0&1\\1&0&0\end{bmatrix}.$$

Hence, we need to solve

$$\det\begin{bmatrix}z^{-1}-1&z^{-1}&0\\0&-1&z^{-1}\\z^{-1}&0&-1\end{bmatrix}=0 \quad\Longleftrightarrow\quad \frac{1+z^2-z^3}{z^3}=0$$

We now claim that this equation has one real and two complex roots, where the real root is the expression inside the logarithm in the proposition, which should be straightforward to verify. □

Proof (of Proposition 6). For all error functions $e\in\mathscr{E}_1$, it holds by definition that $e(x)=0$ for $x\in[-2,2]$. We may choose the function values on remaining positions, that is, the intervals $[3,t]$ and $[-t,-3]$, freely under the constraint that $e(x-1)+e(x)+e(x+1)\le 1$ for all $x\in[-t+1,t-1]$. In each such interval, there are, according to Lemma 7 and Proposition 8, $N\le 2^{\lambda(t-2)}$ choices, with $\lambda := H(X)$ the entropy of the FSM (when viewed as a random source) in Figure 1. Because the choices for the two intervals are independent, in total there are N^2 choices for e, hence $|\mathscr{E}_1|\le 2^{2\lambda(t-2)}$. □

2.3.3 The Legendre Symbol as a "Noisy" Sign

Suppose that p is a Blum prime. We can view the Legendre symbol $\left(\frac{x}{p}\right)$ for $x \in \mathbb{F}_p$ as a "noisy" version of the sign of x, i.e.

$$\left(\frac{x}{p}\right) = (-1)^{e(x)}\mathrm{sgn}(x), \tag{1}$$

where $e(x)$ is the error function that is determined by p. If we now plug (1) into Lemma 5, we can conclude that we may compute the sign of x as the sign of the sum of the Legendre symbols of positions in a length-$(2k+1)$ interval centered at x, for all $x \in [t_1+k, t_2-k]$, if e is an error function on the interval $[t_1, t_2]$ admissible for k.

Because p is a Blum prime, the pattern of Legendre symbols has odd symmetry, which implies that we can w.l.o.g. define $\mathscr{T}$ such that it is symmetric around zero. A natural question, for a given Blum prime p, non-negative integer k, and $\mathscr{T} := [-d, d]$ for a positive integer $d \geq k$, is how large d can maximally be such that e is an error function on $\mathscr{T}$ that is admissible for k. This gives rise to the following equivalent definition, in which we leave the error function implicit.

Definition 2.2. Let k be a non-negative integer, and let $p > 2k+1$ be a Blum prime. We define the *k-range of p*, denoted $d_k(p)$, to be the largest integer d such that for all x with $1 \leq x \leq d$ it holds that

$$\sum_{i=-k}^{k} \left(\frac{x+i}{p}\right) > 0, \tag{2}$$

and we set $d_k(p) := 0$ if no such d exists.

Note that $d_0(p)$ tells us the maximum size of Yu's "Consecutive Quadratic Residues and Non-Residues Sign Module" for a given prime p, i.e. in Yu's terminology and notation: a Blum prime p *qualifies* for $\pm\ell$-CQRN for all $\ell \leq d_0(p)$.

Lower bound on $d_k(p)$. If $p > 2k+1$ and $d_0(p) > k$, then $d_k(p) \geq d_0(p)$.

Example. Let us illustrate Definition 2.2 by means of an example. Let us take $p = 23$; note that this is a Blum prime. Below, we have evaluated the first 16 Legendre symbols.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
$\left(\frac{x}{p}\right)$	0	1	1	1	1	−1	1	−1	1	1	−1	−1	1	1	−1	−1	

We can now read off that $d_0(23) = 4$. Furthermore, it is easy to verify that $d_1(23) = 5$, $d_2(23) = 8$, and $d_3(23) = 7$.

2.3.4 Avoiding Zero by Restricting to Odd Positions

As mentioned in the preliminaries, if we use the single-online-round protocol for securely evaluating the Legendre symbol, we may not evaluate the Legendre symbol on the zero element. A simple trick to avoid zero (also used in [115]) is to restrict to evaluation of odd inputs by using the map $x \mapsto 2x+1$. Note that this implies that we cannot compute $\mathrm{sgn}(x)$ using the single-online-round protocol; instead we will evaluate $\mathsf{GEZ}(x)$. Also note that Definition 2.2 is not compatible with the "zero-avoidance" trick; we need to slightly update Definition 2.2 by incorporating the map $x \mapsto 2x+1$, which gives rise to the following definition.

Definition 2.3. Let k be a non-negative integer, and let $p > 2k+1$ be a Blum prime. We define $d_k^*(p)$ as the largest integer d such that for all x with $1 \le x \le d$ it holds that

$$\sum_{i=-k}^{k} \left(\frac{2(x+i)+1}{p}\right) > 0, \tag{3}$$

and we set $d_k^*(p) := 0$ if no such d exists.

Note that for any Blum prime p for which $d_0(p) > 1$ (which implies that $d_0(p)$ is even), it is easy to see that it holds that $d_0^*(p) = \frac{1}{2}d_0(p) - 1$. For $k > 0$, such simple relations do not seem to exist. This means, for example, that a prime p that gives rise to a high value for $d_1(p)$, does not necessarily give a high value for $d_1^*(p)$, and vice versa.

2.3.5 Bounds on $d_0(p)$

The value $d_0(p)$ can be interpreted as the position just before the appearance of the first quadratic non-residue. Let $n_1(p)$ denote the smallest quadratic non-residue. Finding bounds on $n_1(p)$ is a well-known problem in number theory, with important contributions from Polyà, Vinogradov and Burgess, among others. The best explicit upper bound that is currently known (for p a Blum prime) is due to Treviño [103]:

$$d_0(p) + 1 = n_1(p) \le 1.1\sqrt[4]{p}\log p.$$

Graham and Ringrose [44] proved an unconditional asymptotic lower bound (improving on a previous result by, independently,[4] Fridlender [36] and Salié [93]), namely, that there exist infinitely many primes for which

$$d_0(p) = \Omega(\log(p) \cdot \log\log\log p).$$

Lamzouri *et al.* [69] prove that conditional on the Generalized Riemann Hypothesis, for all primes $p \ge 5$ it holds that

$$d_0(p) + 1 = n_1(p) < (\log p)^2.$$

2.3.6 Bounds on $d_1(p)$

Hudson [53] proves an upper bound on the least *pair* of quadratic non-residues. Formally, let $n_2(p)$ be the smallest value such that $n_2(p)$ and $n_2(p)+1$ are quadratic non-residues. For $k = 1$, it must hold that $d_1(p) < n_2(p)$, because an "error pattern" consisting of two consecutive quadratic non-residues (such that $n_2(p) \in [1, (p-3)/2]$) cannot be corrected using a majority vote in a window of length $2k+1 = 3$. Hudson's bound is as follows. For every $p \ge 5$ we have that

$$d_1(p) < n_2(p) \le (n_1(p) - 1)q_2,$$

where q_2 is the second smallest prime that is a quadratic non-residue modulo p. Hildebrand [49] also proves an upper bound on $n_2(p)$:

$$d_1(p) < n_2(p) \le p^{1/(4\sqrt{e})+\varepsilon} \quad p \ge p_0(\varepsilon),$$

for every $\varepsilon > 0$ and $p_0(\varepsilon)$ a sufficiently large constant depending on ε.

Sun [98] gives a construction for generating all elements n in $\mathbb{F}_p$ such that n and $n+1$ are quadratic non-residues.

[4] Ankeny [3] attributes this result to Chowla, but does not provide a reference.

Lemma 9 ([98])**.** *Let p be an odd prime and let g be a primitive root of p. Then,*

$$\begin{aligned}\mathscr{U} &:= \left\{n : \left(\frac{n}{p}\right) = \left(\frac{n+1}{p}\right) = -1\right\} \\ &= \left\{u_k : u_k \equiv \frac{(g^{2k-1}-1)^2}{4g^{2k-1}} \mod p, \quad u_k \in \mathbb{F}_p, \quad k = 1, \dots, \left\lfloor \frac{p-1}{4} \right\rfloor \right\}\end{aligned}$$

We can interpret this lemma as giving a collection of upper bounds on $d_1(p)$, i.e. $d_1(p) < n_2(p) \leq u_k$ holds for every $k = 1, \dots, \lfloor (p-1)/4 \rfloor$.

An error pattern that consists of two quadratic non-residues that are separated by one arbitrary position can also not be corrected using a majority vote in a window of length $2k+1 = 3$. Inspired by Sun, we prove the following lemma.

Lemma 10. *Let p be a Blum prime, let $b := \left(\left(\frac{2}{p}\right)+1\right)/2 \in \{0,1\}$ and let g be a primitive root of p. Then,*

$$\begin{aligned}\mathscr{V} &:= \left\{n : \left(\frac{n}{p}\right) = \left(\frac{n+2}{p}\right) = -1\right\} \\ &= \left\{v_k : v_k \equiv \frac{(g^{2k-b}-1)^2}{2g^{2k-b}} \mod p, \quad v_k \in \mathbb{F}_p, \quad k = 1, \dots, (p-3)/4 \right\}.\end{aligned}$$

Also this lemma can be viewed as giving a collection of upper bounds on $d_1(p)$. If $\left(\frac{n}{p}\right) = \left(\frac{n+2}{p}\right) = -1$, then a decoding error (under majority decoding with $k = 1$) will occur at position $n+1$, hence we have that $d_1(p) \leq v_k$ holds (instead of strict inequality) for every $k = 1, \dots, (p-3)/4$.

Proof. Let $\chi(x) := \left(\frac{x}{p}\right)$ for all $x \in \mathbb{F}_p$. Jacobsthal [56] proves that for p a Blum prime,

$$\left|\{n : \quad \chi(n) = \chi(n+2) = -1, \quad \chi(n+1) = 1, \quad n \in \mathbb{F}_p\}\right| = \frac{p-1+2\left(\frac{2}{p}\right)}{8},$$

and

$$\left|\{n : \quad \chi(n) = \chi(n+1) = \chi(n+2) = -1, \quad n \in \mathbb{F}_p\}\right| = \frac{p-5-2\left(\frac{2}{p}\right)}{8}.$$

Hence, by summing the cardinalities of the above sets, we get that

$$\left|\{n : \chi(n) = \chi(n+2) = -1, n \in \mathbb{F}_p\}\right| = \frac{p-3}{4}.$$

For $j = 1, 2, \dots, (p-3)/2$, let $r_j \equiv (g^j-1)^2/(2g^j) \mod p$. Then, $r_j + 2 \equiv (g^j+1)^2/(2g^j) \mod p$. It now follows that $\chi(r_j) = \chi(r_j+2) = (-1)^j \chi(2)$ for all $j = 1, 2, \dots, (p-3)/2$. Hence, $\chi(r_{2k-(\chi(2)+1)/2}) = \chi(r_{2k-(\chi(2)+1)/2}+2) = -1$ for all $k = 1, 2, \dots, (p-3)/4$.

It remains to prove that $r_s \neq r_t \mod p$ for all $s, t \in [(p-3)/2]$ with $t \neq s$; for this part we can re-use Sun's proof technique used in the proof of Lemma 9. Namely, for all $s, t \in [(p-3)/2]$ with $t \neq s$, we have that $g^{s+t} \not\equiv 1 \mod p$ (since g is a primitive root), which implies that $g^s - g^t \not\equiv (g^s - g^t)/g^{s+t}$ mod p. Hence, $g^s + g^{-s} \not\equiv g^t + g^{-t} \mod p$ from which we obtain that $r_s \not\equiv r_t \mod p$. We can now conclude that

$$\{n : \chi(n) = \chi(n+2) = -1, n \in \mathbb{F}_p\} = \{r_{2k-b} : k \in [(p-3)/4]\},$$

and the claim follows. □

2.4 Finding a Prime for a Desired k-Range

In order to find a prime that, for given integers k and D_k, gives rise to $d_k(p) \geq D_k$, we could in principle take a "brute force" approach by letting a computer run exhaustively through the primes in increasing order, compute $d_k(p)$ explicitly for each prime, and stop when $d_k(p) \geq D_k$. Although this approach works for small values of k and D_k (say for $D_1 < 200$), for larger D_k this will become intractable.

A better approach is to exploit the multiplicative structure of the Legendre symbol, the law of quadratic reciprocity and the Chinese Remainder Theorem. We will first study the problem for the case $k = 0$ and then extend the method, first to $k = 1$ and finally to arbitrary k.

2.4.1 Finding Primes with High $d_0(p)$

Recall that finding a prime p' such that $d_0(p') \geq D$, for some D, means that p' must be a Blum prime such that the elements $1, \ldots, D$ are quadratic residues modulo p'. By the complete multiplicativity of the Legendre symbol, it suffices to find a Blum prime p such that *all prime factors of the integers in* $[D]$ are quadratic residues modulo p.

We will represent these constraints (via the law of quadratic reciprocity) as a *system of simultaneous linear congruences*, because we can efficiently compute the smallest integer that satisfies such a system of congruences via the constructive proof of the Chinese Remainder Theorem (i.e. via Bézout's identity). We note that [115] also uses this approach to find primes for which $d_0(p)$ is high.

Let q be an odd prime. Then, it holds that

$$\left(\frac{q}{p}\right) = \left(\frac{-p}{q}\right).$$

Proof. $\left(\frac{q}{p}\right) = \left(\frac{p}{q}\right)^{-1}(-1)^{\frac{p-1}{2}\frac{q-1}{2}} = \left(\frac{p}{q}\right)(-1)^{\frac{q-1}{2}} = \left(\frac{p}{q}\right)\left(\frac{-1}{q}\right) = \left(\frac{-p}{q}\right)$, where the first equality holds by the law of quadratic reciprocity, the second holds because p is a Blum prime, the third follows from Euler's criterion and the fourth follows from the multiplicativity property of the Legendre symbol. $\square$

Let $\mathscr{R}_q := \{r \in \{0, \ldots, q-1\} : \left(\frac{-r}{q}\right) = 1\}$. Then, q is a quadratic residue modulo p if and only if

$$\exists r \in \mathscr{R}_q \quad \text{such that} \quad p \equiv r \pmod{q}. \tag{4}$$

Let $q_1, \ldots, q_m$ denote all odd prime factors that occur in the integers in $[D]$. This gives rise to the following system of simultaneous linear congruences, in which x represents the unknown integer:

$$\begin{array}{rclll} x & \equiv & 7 & \bmod 8 & \text{(guarantees that } \left(\frac{-1}{x}\right) = -1 \text{ and } \left(\frac{2}{x}\right) = 1\text{)}, \\ x & \equiv & a_1 & \bmod q_1 & a_1 \in \mathscr{R}_{q_1}, \\ x & \equiv & a_2 & \bmod q_2 & a_2 \in \mathscr{R}_{q_2}, \\ \vdots & & \vdots & & \vdots \\ x & \equiv & a_m & \bmod q_m & a_m \in \mathscr{R}_{q_m}. \end{array} \tag{5}$$

Note that the coefficients a_i for all $i \in [m]$ can be chosen freely. Each choice for the vector $(a_1, \ldots, a_m)$ is in one-to-one correspondence with a unique arithmetic progression of solutions to the above system of congruences, i.e. $x, x+Q, x+2Q, \ldots$ where $Q := 8\prod_{i\in[m]} q_i$. Linnik's theorem [73] (combined with Xylouris' bound [112]) asserts that there will be a prime in this arithmetic progression whose size is bounded as $O(Q^5)$.

Finding the smallest such prime. While making an arbitrary choice for $(a_1, \ldots, a_m)$ and then finding *some* prime that satisfies the above system is relatively easy (via the constructive proof of the CRT), finding the *smallest* prime that satisfies the above system is a (much) harder task, as it involves searching over the set of all choices for $(a_1, \ldots, a_m)$, whose cardinality is exponential in m.

An alternative approach to find the smallest prime that satisfies the system of simultaneous congruences is by enumeration. Each congruence in Equation (5) gives rise to a *set of arithmetic progressions*, with the property that every integer solution to (5) will lie in one of the arithmetic progressions in the set. Hence, instead of enumerating the (odd) integers, we can enumerate through such a set of arithmetic progressions.

For example, the second line in (5) corresponds to the following set of arithmetic progressions:

$$\{k \cdot q_1 + a_1 : a_1 \in \mathscr{R}_{q_1}, k \in \mathbb{N}\}.$$

By the CRT, we can also "intersect" congruences, i.e. two congruences

$$\begin{array}{rclll} x & \equiv & a & \mod p, & a \in A \\ x & \equiv & b & \mod q, & b \in B \end{array}$$

can be combined as

$$\begin{array}{rclll} x & \equiv & c & \mod \ell, & c \in C \end{array}$$

with $\ell := \mathrm{lcm}(p,q)$ and

$$C := (A + \{0, p, \ldots, (\ell/p - 1)p\}) \cap (B + \{0, q, \ldots, (\ell/q - 1)q\}), \tag{6}$$

where '$+$' denotes Minkowski addition. Note that we can intersect several congruences by recursively applying the above binary intersection operation.

2.4.2 Finding Primes with High $d_1(p)$

For $k > 0$, to find p with $d_k(p) \geq D'$ for some positive integer D', p no longer has to satisfy all congruences; instead, some subsets suffice. For example, for $d_1(p) \geq 6$ we need $\left(\frac{2}{p}\right) = 1$ and at least one of $\left(\frac{5}{p}\right) = 1$ or $\left(\frac{6}{p}\right) = \left(\frac{2}{p}\right)\left(\frac{3}{p}\right) = \left(\frac{3}{p}\right) = 1$, otherwise Equation (2) fails to hold for $a = 5$. In order for Equation (2) to hold, we have one set of congruences for every length-$(2k+1)$ subinterval of $\{-k, \ldots, d\}$; even for $k = 1$ this quickly grows prohibitively large for non-trivial lower bounds D on $d_k(p)$. While for $k > 0$ the density of primes p satisfying $d_k(p) \geq D$ is greater than for $k = 0$, the search becomes a lot more expensive.

For $k = 1$, we simplify our search for p with $d_1(p) \geq D_1$ with an extra condition: we also require $d_0(p) \geq D_0$ where $D_1 \leq (D_0)^2$. This allows a simpler equivalent condition, where we only get a set of congruences for certain pairs of primes, instead of for every length-3 subinterval of $\{-k, \ldots, d\}$.

Definition 2.4. Let D_0, D_1 be non-negative integers with $D_0 < D_1 \leq (D_0)^2$. Let q, q' be distinct primes. We say that $\{q, q'\}$ is a *related pair* on $(D_0, D_1]$ if $D_0 < q, q' \leq D_1$ and there exist positive integers $x, y < D_0$ such that $|xq - yq'| \leq 2$ and $\max\{xq, yq'\} \leq D_1$.

Proposition 11. *Let D_0, D_1 be non-negative integers with $D_0 < D_1 \leq (D_0)^2$, and let p be a Blum prime with $d_0(p) \geq D_0$. Then $d_1(p) \geq D_1 - 1$ if and only if the following condition holds: for every related pair of primes $\{q, q'\}$ on $(D_0, D_1]$ it holds that $\left(\frac{q}{p}\right) = 1 \vee \left(\frac{q'}{p}\right) = 1$.*

Proof. Let a be any positive integer such that $a \leq D_1$. First, we show that $\left(\frac{a}{p}\right) = -1$ if and only if a has a prime factor $q > D_0$ and $\left(\frac{q}{p}\right) = -1$. Suppose a has a prime factor $q > D_0$ with $\left(\frac{q}{p}\right) = -1$. Since $\frac{a}{q} < \frac{a}{D_0} \leq \frac{D_1}{D_0} \leq D_0$, we have $\left(\frac{a/q}{p}\right) = 1$, hence $\left(\frac{a}{p}\right) = -1$. If a does not have a prime factor $q > D_0$ with $\left(\frac{q}{p}\right) = -1$, then taking any prime factor $q'|a$, it must hold that $q' > D_0$, in which case $\left(\frac{q'}{p}\right) = 1$ by assumption, or $q' \leq D_0$, in which case $\left(\frac{q'}{p}\right) = 1$ by $d_0(p) \geq D_0$.

We now finish the proof by showing $d_1(p) < D_1 - 1$ if and only if there is some related pair q, q' such that $\left(\frac{q}{p}\right) = \left(\frac{q'}{p}\right) = -1$. We have $d_1(p) < D_1 - 1$ if and only if there exists an integer x such that $1 < x \leq D_1 - 1$ and $\left(\frac{x-1}{p}\right) + \left(\frac{x}{p}\right) + \left(\frac{x+1}{p}\right) < 0$. This latter inequality holds if and only if at least two of $\{x-1, x, x+1\}$ have Legendre symbol -1. By the above, this holds if and only if two of these numbers have respective prime factors $q, q' > D_0$ and $\left(\frac{q}{p}\right) = \left(\frac{q'}{p}\right) = -1$. For these q, q', we have that they constitute a related pair, since they each have a multiple in $\{x-1, x, x+1\}$ and $x + 1 \leq D_1$. Conversely, for any related pair there exists such an interval $\{x-1, x, x+1\}$ with $1 < x \leq D_1 - 1$. □

Proposition 11 gives sufficient conditions for $d_1(p) > D_1 - 1$ in terms of related pairs of primes that have to satisfy certain congruences. If we want to include those congruences in the system of simultaneous congruences as shown in Equation (5), we need to represent them in the same form. For every pair of related primes $\{q, q'\}$, the condition that $\left(\frac{q}{p}\right) = 1 \vee \left(\frac{q'}{p}\right) = 1$ in Proposition 11 corresponds to taking the *union* of the associated residue sets $\mathscr{R}_q$ and $\mathscr{R}'_q$ of the related primes q and q'. That is, let $\ell := \operatorname{lcm}(q, q')$, then

$$\mathscr{R}_{q,q'} := (\mathscr{R}_q + \{0, q, \ldots, (\ell/q - 1)q\}) \cup (\mathscr{R}'_q + \{0, q', \ldots, (\ell/q' - 1)q'\}),$$

where '$+$' denotes Minkowski addition. We can now express the related-primes congruence as

$$x \equiv b \mod \ell \quad b \in \mathscr{R}_{q,q'}.$$

Since this congruence has exactly the same form as the other congruences in (5), we can also take intersections (using Equation (6)) between a related-primes congruence $\mathscr{R}_{q,q'}$ and another congruence.

2.4.3 Finding Primes with High $d_k(p)$

We can naturally extend the approach for searching primes with high $d_1(p)$ to an approach for finding primes with high $d_k(p)$ for $k > 1$. This involves imposing constraints on sets of $k+1$ distinct primes that are related in a suitably defined way. Note, however, that the use of this for high k is limited, given that we still constrain $D_k \leq (D_0)^2$.

Definition 2.5. Let D_0, D_k be non-negative integers with $D_0 < D_k \leq (D_0)^2$. Let $Q = \{q_0, \ldots, q_k\}$ be a set of $k+1$ distinct primes. We say that Q is a *related set* on $(D_0, D_k]$ if $Q \subseteq (D_0, D_k]$ and there exist positive integers $x_0, \ldots, x_k < D_0$ such that:

1. for any i with $0 \leq i \leq k$ we have $x_i q_i \leq D_k$
2. for any i, j with $0 \leq i < j \leq k$ it holds that $|x_i q_i - x_j q_j| \leq 2k$

Proposition 12. *Let D_0, D_k be non-negative integers with $D_0 < D_k \leq (D_0)^2$, and let p be a Blum prime with $d_0(p) \geq D_0$. Then $d_k(p) \geq D_k - k$ if and only if the following condition holds: for every set Q of $k+1$ distinct primes related on $(D_0, D_k]$, it holds that there exists some $q \in Q$ with $\left(\frac{q}{p}\right) = 1$.*

The proof goes along the same lines as that of Proposition 11.

Remark. Although we have presented Proposition 12 for general k, in practice we shall mostly use $k \in \{1,2\}$. For larger k, the restriction $D_0 < D_k \leq (D_0)^2$ causes the conditions for $d_0(p) \geq D_0$ to dominate the search.

2.4.4 Finding Primes with High $d_k^*(p)$

We can also apply the method from Section 2.4.3 to find p with large $d_k^*(p)$, but the search procedure needs to be modified slightly. We present the appropriate modifications to the statements of Section 2.4.3.

Definition 2.6. Let D_0, D_k be non-negative integers with $D_0 < D_k \leq (D_0)^2$. Let $Q = \{q_0, \ldots, q_k\}$ be a set of $k+1$ distinct primes. We say that Q is a $*$*-related set* on $(D_0, D_k]$ if $Q \subseteq (D_0, D_k]$ and there exist positive integers $x_0, \ldots, x_k < D_0$ such that:

1. for any i with $0 \leq i \leq k$ we have $x_i q_i \leq D_k$
2. for any i, j with $0 \leq i < j \leq k$ it holds that $|x_i q_i - x_j q_j| \in \{2, 4, 6, \ldots, 4k\}$

Proposition 13. *Let D_0, D_k be non-negative integers with $D_0 < D_k \leq (D_0)^2$, and let p be a Blum prime with $d_0^*(p) \geq \frac{1}{2}D_0$. Then $d_k^*(p) \geq \frac{1}{2}D_k - k$ if and only if the following condition holds: for every set Q of $k+1$ distinct primes $*$-related on $(D_0, D_k]$, it holds that there exists some $q \in Q$ with $\left(\frac{q}{p}\right) = 1$.*

2.5 Secure Protocols for GEZ

In this section we present three protocols for evaluating the GEZ function, for $k = 1$ and $k = 2$. Note that these immediately imply comparison protocols; from the triangle inequality it follows that correctness for comparison is guaranteed if both inputs lie in $[-d/2, d/2]$, where $[-d, d]$ is the input range of the GEZ protocol. Throughout this section, we suppose that p is a Blum prime.

We first describe a protocol for securely evaluating the Legendre symbol, which we call Legendre. We describe the protocol in terms of black-box invocations of subprotocols for sampling a random element from $\mathbb{F}_p^*$ (denoted as RandomElem($\mathbb{F}_p^*$)) and for sampling a random bit $\{0,1\} \subset \mathbb{F}_p$ called RandomBit.

Protocol 3 Legendre($[x]$)

```
   Offline Phase
1: $[a] \leftarrow \mathsf{RandomElem}(\mathbb{F}_p^*)$
2: $[b] \leftarrow \mathsf{RandomBit}()$
3: $[s] \leftarrow 2[b] - 1$
4: $[r] \leftarrow [s] \cdot [a^2]$
5: return $([r], [s])$

   Online Phase
6: $c \leftarrow [x] \cdot [r]$
7: $[z] \leftarrow \left(\frac{c}{p}\right) \cdot [s]$
8: return $[z]$
```

2.5.1 A Secure Medium-Range GEZ Protocol for $k = 1$

In our protocol for $k = 1$, shown as Protocol 4, we compute the sign of the sum of the Legendre symbols by means of the multivariate polynomial

$$f(x,y,z) = \frac{x+y+z-xyz}{2},$$

which can be evaluated securely in two rounds using ordinary secure multiplication. It is easy to verify that f correctly computes the sign of the sum of $x,y,z \in \{-1,+1\}$.

Protocol 4 SimpleGEZ1($[a]$), $|a| \leq d_1^*(p)$

```
1: $[x] \leftarrow \mathsf{Legendre}(2[a]-1)$, $[y] \leftarrow \mathsf{Legendre}(2[a]+1)$, $[z] \leftarrow \mathsf{Legendre}(2[a]+3)$
2: $[s] \leftarrow ([x]+[y]+[z]-[x][y][z])/2$
3: return $[s]$
```

Decreasing the round complexity in the online phase. Protocol SimpleGEZ1 requires three rounds in the online phase. We can bring this down to a single round by premultiplying the random Legendre symbols produced in the offline phase of the Legendre protocol. This is shown in Protocol 5. The random bit protocol has been concretely instantiated in the offline phase of Protocol 5 to show that the product of the three random Legendre symbols can be computed in parallel to the preparation of their corresponding random elements. The offline phase requires two rounds in addition to the round complexity of securely sampling random elements of $\mathbb{F}_p^*$.

Protocol 5 SingleRoundGEZ1($[a]$), $|a| \leq d_1^*(p)$

```
   Offline Phase
1: for $i \in \{1,2,3\}$ do $[t_i] \leftarrow \mathsf{RandomElem}(\mathbb{F}_p^*)$;   $[u_i] \leftarrow \mathsf{RandomElem}(\mathbb{F}_p^*)$
2: for $i \in \{1,2,3\}$ do $[v_i] \leftarrow [t_i]\cdot[t_i]$;   $w_i \leftarrow [u_i]\cdot[u_i]$
   $[m] \leftarrow [u_1]\cdot[u_2]$
3: for $i \in \{1,2,3\}$ do $[r_i] \leftarrow [v_i]\cdot[u_i]\cdot w_i^{-1/2}$;   $[s_i] \leftarrow [u_i]\cdot w_i^{-1/2}$
   $[n] \leftarrow [m]\cdot[u_3]\cdot\prod_{i=1}^{3} w_i^{-1/2}$
4: return $([r_1],[s_1],[r_2],[s_2],[r_3],[s_3],[n])$

   Online Phase
5: for $i \in \{1,2,3\}$ do $c_i \leftarrow (2[a]-1+2i)\cdot[r_i]$
6: return $2^{-1}\left(\sum_{i=1}^{3}[s_i]\cdot\left(\frac{c_i}{p}\right) - [n]\cdot\prod_{i=1}^{3}\left(\frac{c_i}{p}\right)\right)$
```

2.5.2 A Secure Medium-Range GEZ Protocol for $k = 2$

In our protocol for $k = 2$, shown as Protocol 6, we compute the sign of the sum of the five Legendre symbols by means of another invocation of Legendre. In the latter (outer) invocation of Legendre, we do not need to apply the $x \mapsto 2x+1$ map because we sum an odd number of values in $\{-1,+1\}$ which cannot become zero. Note that this requires that $d_0(p) \geq 5$ for correctness of the protocol.

Protocol 6 GEZ2($[a]$), $|a| \le d_2^*(p)$, $d_0(p) \ge 5$

```
1: $[x_1] \leftarrow \mathsf{Legendre}(2[a]-3)$, $[x_2] \leftarrow \mathsf{Legendre}(2[a]-1)$, $[x_3] \leftarrow \mathsf{Legendre}(2[a]+1)$
   $[x_4] \leftarrow \mathsf{Legendre}(2[a]+3)$, $[x_5] \leftarrow \mathsf{Legendre}(2[a]+5)$
2: $[s] \leftarrow \mathsf{Legendre}([x_1]+[x_2]+[x_3]+[x_4]+[x_5])$
3: return $[s]$
```

2.6 Application: Fast Neural Network Evaluation in MPC

In this section we demonstrate the usefulness of our secure sign evaluation technique for securely evaluating a neural network.

2.6.1 A Binarized Multi-Layer Perceptron for MNIST

As a basis for our experiments, we take the binarized multi-layer perceptron (BMLP) as described in [52] for recognizing handwritten digits from the well-known MNIST benchmark data set. This network uses the sign function as its non-linearity, and is designed to be evaluated using integer arithmetic, which allows for a natural MPC implementation.

Let $\mathscr{B} := [0,255] \subset \mathbb{Z}$. The MNIST data set contains images of 28-by-28 pixels, where the intensity of each pixel is represented by an integer from the interval $\mathscr{B}$ (a zero represents black, 255 represents white, and the values in between represent shades of gray). In the BMLP described in [52], an input image is represented as a byte *vector* $x \in \mathscr{B}^{784}$. Note that by reshaping a two-dimensional image into a (one-dimensional) vector the spatial structure is lost, which is not a problem in the context of a MLP, as opposed to, e.g. a convolutional neural network.

Let $n \in \mathbb{N}$ represent the number of neurons per layer. The BMLP as described in [52] consists of four layers, and uses $n = 4096$. We view each layer L_i, for all $i \in [1,4]$, as a map between an input and output vector:

$$\begin{array}{llcll} L_1: & \mathscr{B}^{784} & \to & \{-1,+1\}^n, & \\ L_i: & \{-1,+1\}^n & \to & \{-1,+1\}^n, & i \in \{2,3\} \\ L_4: & \{-1,+1\}^n & \to & \mathbb{Z}^{10}. & \end{array}$$

Let $k_1 = k_2 = k_3 = m_2 = m_3 = m_4 = n$ and $k_4 = 10$ and $m_1 = 784$. In [52], the output of L_i is computed as

$$L_i(x) := \begin{cases} \mathrm{Sign}(\mathrm{BatchNorm}^{k_i}_{\Theta_i}(W_i x + b_i)), & i \in \{1,2,3\} \\ \mathrm{BatchNorm}^{k_i}_{\Theta_i}(W_i x + b_i) & i = 4. \end{cases}$$

Here $W_i \in \{-1,+1\}^{k_i \times m_i}$ is a matrix of weights, and $b_i \in \mathbb{Z}^{k_i}$ is a vector of bias values. The function BatchNorm, which applies *batch normalization* element-wise, is defined as

$$\begin{array}{lccl} \mathrm{BatchNorm}^{\ell}_{\Theta_i}: & \mathbb{Z}^{\ell} & \to & \mathbb{Z}^{\ell} \\ & (x_1,\ldots,x_\ell) & \mapsto & (f_{\Theta_i,1}(x_1),\ldots,f_{\Theta_i,\ell}(x_\ell)) \end{array}$$

where $\Theta_i := (\mu_i, \tilde{\sigma}_i, \gamma_i, \beta_i)$ are the batch norm parameters for the ith layer: $\mu_i = (\mu_{i,1},\ldots,\mu_{i,\ell})$, $\tilde{\sigma}_i = (\tilde{\sigma}_{i,j})_{j\in[1,\ell]}$, $\gamma = (\gamma_{i,j})_{j\in[1,\ell]}$, and $\beta = (\beta_{i,j})_{j\in[1,\ell]}$, and

$$f_{\Theta_i,j}(x) := \gamma_{i,j}\left(\frac{x-\mu_{i,j}}{\tilde{\sigma}_{i,j}}\right) + \beta_{i,j}.$$

The function Sign applies the GEZ function element-wise,

$$\begin{array}{rccc} \text{Sign}: & \mathbb{Z}^n & \to & \{-1,+1\}^n \\ & (x_1,\ldots,x_n) & \mapsto & (\mathsf{GEZ}(x_1),\ldots,\mathsf{GEZ}(x_n)). \end{array}$$

To obtain the final output of the BMLP, which is an integer $y \in [0,9]$, we apply an (oblivious) argmax operation to the output of L_4:

$$y := \arg\max L_4(L_3(L_2(L_1(x)))).$$

2.6.2 Eliminating Redundant Parts of Batch Normalization

In layers 1, 2 and 3, the Sign function is applied directly to the output of the BatchNorm function. Because the sign function is invariant to multiplying its input by a positive scalar, the BatchNorm function might perform some operations that are immediately undone by the sign function. Indeed, it actually turns out that the BatchNorm function (when followed by the Sign function) reduces to an additional bias term; the authors of [52] seem to have overlooked this. Formally,

$$\begin{aligned} f_i(x) = \gamma_i(\frac{x-\mu_i}{\tilde{\sigma}_i}) + \beta_i = \gamma_i\left(\left(\frac{x-\mu_i}{\tilde{\sigma}_i}\right) + \frac{\beta_i}{\gamma_i}\right) = \frac{\gamma_i}{\tilde{\sigma}_i}\left(x-\mu_i+\frac{\beta_i\tilde{\sigma}_i}{\gamma_i}\right), \\ \mathsf{GEZ}(f_i(x)) = \mathsf{GEZ}\left(x-\mu_i+\frac{\beta_i\tilde{\sigma}_i}{\gamma_i}\right), \qquad \gamma_i, \tilde{\sigma}_i > 0. \end{aligned}$$

Hence, we update the bias vector in all layers except the last as follows,

$$b_i' := b_i - \mu_i + \frac{\beta_i\tilde{\sigma}_i}{\gamma_i}$$

where all operations (addition, subtraction, multiplication, and division) in the above expression are performed element-wise. With this modification, evaluation of the BMLP network simplifies to

$$L_i(x) = \begin{cases} \text{Sign}(W_i x + b_i')), & i \in \{1,2,3\} \\ \text{BatchNorm}^{k_i}_{\Theta_i}(W_i x + b_i) & i = 4. \end{cases}$$

2.6.3 The Parameter n vs. the Input Range of our Sign Protocol

We want to use GEZ2 with a Blum prime modulus not exceeding 64-bit that gives rise to the largest known value for $d_2^*(p)$, which is, at the time of writing the prime $p = 13835556230699448671$ for which $d_2^*(p) = 493$.

For layer L_1, the magnitudes of the elements in the vector $W_1 x + b_1'$ for some image $x \in \mathscr{B}^{784}$ will typically be way too large compared to the input range on which our medium-range sign protocol guarantees a correct answer. Hence, for L_1 we will apply an "off-the-shelf" large-range sign protocol, such as Toft's comparison protocol [101].

For layers L_2 and L_3 we apply Protocol GEZ2. Also for these layers, however, there seems to be a mismatch between the input range of GEZ2 on which it guarantees correctness, i.e. $[-493, 493]$, and the magnitudes of the elements in the vector $W_i y + b_i'$ for $i \in \{3,4\}$, where $y \in \{-1,+1\}^n$. The first term in this sum (the vector $W_i y$), can have elements with magnitude equal to n in the worst case, and $n = 4096 \gg 493 = d_2^*(p)$.

Nonetheless, the distribution of values in the vector $W_i y + b_i'$ for all $i \in \{2,3\}$ is strongly concentrated around zero, hence we will just ignore the fact that GEZ2 will be invoked a number of times

Table 2: Classification performance of the BMLP on 10,000 MNIST test images

	Full-Range Sign	GEZ2
Number of misclassifications	248	239
Error rate	0.0248	0.0239

on values outside the range for which it guarantees correctness. As we show quantitatively in Table 2, this does not significantly impact the classification performance compared to a network where the full-range sign protocol is also used in layers L_2 and L_3. (Surprisingly, using GEZ2 even slightly improves the performance on this particular test set.)

2.6.4 Training the Network

We have trained the BMLP on a GPU using Courbariaux' original implementation (described in [52]) which is publicly available on GitHub.

2.6.5 Experimental Results (Neural Network Evaluation)

We have implemented the neural network in MPyC, a Python framework for prototyping MPC protocols. In MPyC in three-player mode on a Intel four-core 4th generation Core i7 3.6 GHz, with all players running on the same machine, secret-sharing the parameters and evaluating the network took 3 minutes and 38 seconds, where we note that MPyC is Python-based and not optimized for runtime performance. Evaluation time in an actual deployment will depend on the MPC framework, CPU speed, network latency and throughput.

We have implemented an identical non-MPC version in Python (including a non-MPC implementation of GEZ2 which produces exactly the same errors outside its input range $[-493, 493]$) to measure the classification error on 10,000 test images; those results are shown in Table 2.

2.7 Some Suitable Primes for Our Protocols

Table 3 shows results of our search for primes that give rise to as high as possible values of $d_1^*(p)$ and $d_2^*(p)$. Table 4 shows primes that give rise to as high as possible values of $d_0(p)$, $d_1(p)$ and $d_2(p)$. The sequence corresponding to $d_0(p)$ coincides with sequence A002223 from Sloane's Encyclopedia of Integer Sequences, and has been taken from [75].

Table 3: Sequence of primes in increasing order (and their bit-lengths ℓ) for which $d_k^*(p)$ is strictly increasing, for $k \in \{1,2\}$. In the subtables for $d_1^*(p)$ and $d_2^*(p)$, primes below the separating lines have been found via our sieving method, which means that there could exist smaller primes (missed by the sieving method) that give rise to the same or higher values of $d_1^*(p)$ resp. $d_2^*(p)$. For all other primes (i.e. all primes listed except those found by sieving), it holds that the prime is the smallest possible for a given $d_k^*(p)$.

ℓ	p	$d_1^*(p)$
5	23	1
6	47	4
7	83	5
8	131	7
8	239	8
8	251	14
10	1019	16
11	1091	24
13	4259	30
14	10331	33
14	12011	34
17	74051	42
17	96851	44
19	420731	47
20	831899	52
20	878099	53
20	954971	68
23	5317259	78
25	19127891	79
25	31585979	94
28	140258219	98
30	697955579	104
31	1452130811	112
31	1919592419	115
33	4323344819	116
33	4499001491	117
33	6024587819	118
34	9259782419	138
35	19846138451	143
36	34613840351	151
37	73773096179	153
37	119607747731	174
38	163030664579	182
38	170361409391	207
43	4754588149211	229
64	9223374592776481199	231
64	9223384461444136499	233
64	9223394717698296659	243
64	9223452305977096799	249
64	18437888437706621771	269
64	18438220770445614311	272

ℓ	p	$d_2^*(p)$
6	47	3
7	83	6
8	131	8
8	179	15
10	1019	16
11	1091	26
11	1427	31
11	1811	36
14	9539	51
15	19211	68
19	334619	78
20	717419	80
21	1204139	104
22	2808251	114
24	8774531	116
24	11532611	117
25	18225611	152
27	98962211	155
28	247330859	166
30	738165419	174
30	1030152059	188
31	1456289579	197
32	2451099251	206
34	11159531291	207
34	13730529419	216
35	17221585499	219
35	19186524419	232
35	26203369331	242
37	92830394411	248
37	128808841619	287
38	232481520059	324
39	408727560491	335
40	807183995411	370
64	9223382101109640239	410
64	18158545127592455759	429
64	18158547872742314231	482
64	9223770987363748331	492
64	13835556230699448671	493

Table 4: Sequence of primes in increasing order (and their bit-lengths ℓ) for which $d_k(p)$ is strictly increasing, for $k \in \{1,2\}$. The primes in the subtable for $d_0(p)$ have been taken from [75]. The primes below the separating lines (for $\ell \geq 37$) have been found via our sieving method, which means that there could exist smaller primes (missed by the sieving method) that give rise to the same or higher values of $d_1(p)$ resp. $d_2(p)$. The primes below "$\star\star\star$" are 64 bit primes with the best known k-range. For all other primes (i.e. all primes listed except those found by sieving), it holds that the prime is the smallest possible for a given $d_k(p)$.

ℓ	p	$d_0(p)$
4	11	1
5	23	4
7	71	6
9	311	10
9	479	12
11	1559	16
13	5711	18
14	10559	22
15	18191	28
15	31391	30
19	366791	42
22	4080359	46
24	12537719	52
25	30706079	58
26	36415991	60
27	82636319	66
27	120293879	72
27	131486759	82
32	2929911599	96
33	7979490791	100
35	33857579279	106
37	89206899239	108
37	121560956039	112
39	328878692999	130
39	513928659191	136
42	4306732833311	150
43	8402847753431	156
47	70864718555231	162
49	317398900373231	166
49	501108392233679	190
53	5551185799073591	198
53	7832488789769159	222
57	102097158739597271	228
59	315759454565514431	232
60	868116409360316399	238
62	3412527725201978759	240
62	3546374752298322551	270

ℓ	p	$d_1(p)$
5	23	5
5	31	10
7	71	11
8	167	13
8	191	19
10	599	20
11	1319	37
12	3119	40
14	9719	45
14	14951	60
17	110039	65
18	211559	66
19	283631	67
19	289511	72
19	333791	109
21	1884791	129
22	2817239	140
24	10522511	149
25	25155191	156
25	29036999	157
27	79107311	179
27	89658791	217
30	927633671	227
31	1514970551	276
36	56709623759	277
36	60221191631	281
37	81720228911	291
37	86345286719	339
38	187800947879	396
43	8714428081631	431
44	10422103551551	437
44	13729797542471	443
47	78991232073599	452
47	100395799811999	461
54	12210981354571991	577
54	13162388389217591	639
	$\star\star\star$	
64	16141115006107484951	577

ℓ	p	$d_2(p)$
5	23	8
5	31	10
7	71	11
8	167	14
8	191	19
8	239	20
9	359	26
9	479	35
11	1151	38
11	1511	41
12	3527	43
12	3911	58
13	6551	59
14	8951	66
14	12239	89
15	25679	140
19	289511	176
20	662639	182
22	2798351	212
24	10328111	223
24	16178399	226
25	17431391	250
25	19632791	255
25	25380911	276
25	30809159	280
26	53422151	290
27	92989511	308
28	246241511	318
29	442696271	329
30	721250351	379
30	984093431	458
35	18233703479	498
35	29919732911	502
37	95110047119	508
38	149120083199	562
38	241922449271	570
40	696567525359	588
40	700217963639	608
41	1291095727151	640
41	2088877265999	668
43	8590297237079	720
	$\star\star\star$	
64	16141221934733667719	1112

3 MPC implementation of BWT algorithm for inexact DNA string search

3.1 Introduction

Solving the approximate string matching problem under tight privacy concerns is not a trivial task to do. We use the BWT transform algorithm to research the problem of sequence comparison. Subsequently, we apply MPC techniques to investigate its applicability and to produce privacy-preserving DNA sequence alignment algorithms. We implement our protocols in Python using the TUeVIFF MPC framework, of which the underlying protocols are based on Shamir secret sharing.

We implement the protocols in a recursive manner as in the original implementation through judicious use of secret indexing and masking techniques. We identify and analyze two different models to implement a solution for inexact string matching. The first model supports a private search query, which is intended to be searched within a public reference string. The second model supports a private search query in combination with a private reference string. As an example use case one may consider a private search query to be DNA mutations that represent a particular illness and a reference string to be human genomes.

Here we present the MPC implementation of the BWT-based inexact search algorithm together with its construction. For brevity reasons background, preliminaries and performance aspects are covered in detail in [4]. A corresponding correctness proof and adaptive function evaluation is included in SODA deliverable D1.2 that includes verifiable computation.

3.2 Preliminaries

3.2.1 Suffix Array

A Suffix Array *SA* [76] can be interpreted as lexicographically ordered list of the suffixes for string X represented by pointers to their entry positions. *SA* provides a simple, space efficient method to store all suffixes of a given string with their positions in the string.

3.2.2 Edit Distance

In computer science, edit distance (or Levenshtein distance) is utilized to measure the similarity between two strings. Given two strings A and B, the edit distance is the minimal number of edit operations needed to transform string A to string B. We denote this as $ED(A,B)$[88]. The edit operations consist of insertion, deletion and substitution of a single character.

3.2.3 Burrows-Wheeler Transform (BWT)

The BWT algorithm was invented by David Wheeler and Michael Burrows in 1994 [17]. The main concept of the algorithm is to rearrange a character string into a sequence of the non-divergent characters. BWT became popular after its recognition as a compression tool, because compressed indexes based on the *BWT* were among the simplest and most space efficient ones. Currently, it is one of the most popular index structures for genomic data processing. It is used by a wide range of sequence alignment tools.

The *BWT* algorithm is an efficient approach in order to align short sequencing reads (up to 100 base-pairs) with the reference genome (from here on referred to as sequence).

The *BWT* consists of a reversible permutation of the characters in a sequence. The algorithm tends to have the convenient property to group characters that appear contiguously in substrings. This feature is useful for data compression, because a sequence of identical characters in several locations is relatively easy to compress. This property can also be exploited in sequence alignment in a transformation of the reference string that contains several substrings that occur often.

In order to analyze the entire transformation procedure we distinguish between two transform categories: *forward transform* or *BWT*, and a *backward transform* (inverse) or *IBWT* which constructs the original sequence back from the transformed one.

The forward transform can be obtained in three major steps:

- Given sequence X, append a special sign '\$' that appears nowhere else within sequence. Alphabetically, it preceeds any character of X.
- Rotations: get the cyclic shifts of the sequence X by taking an element from one end of X, sticking it on to other end, and storing each rotation in a row of a $[n \text{ x } n]$ matrix for the given length n of X.
- Sort and construct BWT: lexicographically sort the rows of the above matrix using the order of $\$ < A < C < G < T$ over alphabet $\sum = \{\$, A, C, G, T\}$. The resulting matrix is the Burrows-Wheeler Matrix (*BWM*). The transformed sequence (*B*) is the concatenation of the last symbols of the cyclic rotations, namely the last column of the *BWM* read from top to bottom.

The other and further details are provided in [4]. Below we focus on the backward transform algorithm for inexact searches, because that will be used as the basis for the private implementation.

3.2.4 BWT Backtrack

Inexact Backward search This search technique concentrates on bounded traversal / backtracking of the search query W within reference string X. It uses a recursive algorithm that searches for suffix array (SA) intervals of substrings of X that correspond to W with a bounded number of mismatches or differences z.

The algorithm uses backward search to pattern distinct substrings from X. The concept behind inexact search is similar to the exact search except that an upper bound for the number of differences is introduced. The algorithm tries to find substrings with a maximum edit distance. An inexact search is equivalent to search for the *SA interval* of substrings of X that match W, allowing z mismatches. The inexact search algorithm is defined below [71].

- Let B be the *BWT* transform of X.
- Let ***C(ch)*** be the number of elements in original string X that is lexicographically smaller than ch.
- Let ***O(ch,index)*** be the number of occurrences of character ch in transform $B[0 : index]$.

Algorithm 1 Inexact Search: Bounded Traversal/Backtracking

```
1: procedure INEXACT SEARCH(INPUT: $X, W, z, SA, B$; OUTPUT: $I$)
   ▷ Pre: $X$: reference string, $W$: search term, $z$: upper mismatch bound, $SA$: suffix array for $X$, $B : BWT$ transform of $X$
   ▷ Post: $I$: set of all $SA$ intervals corresponding to $W'$, such that $X = YW'Z$, $ED(W,W') \leq z$
2:     $I \leftarrow I \cup$ Inexact Recurrence($X$, $W$, $len(W)-1$, $z$, 0, $len(X)-1$, $SA$, $B$)
3:     return I
```

Algorithm 2 Inexact Search Recurrence

```
1: procedure INEXACT RECURRENCE(INPUT: $X, W, i, z, k, l, SA, B$; OUTPUT: $I$)
   ▷ Pre: $X$: reference string, $W$: search term, $i$: index in $W$, $z$: upper mismatch bound, $[k,l]$: $SA$ interval where $0 \leq k \leq l \leq n$, $SA$: suffix array for $X$, $B : BWT$ transform of $X$
   ▷ Post: Let $R$ be the shortest string that corresponds to $SA$ interval of $[k,l]$ , $I$: set of all $SA$ intervals corresponding to $W'$, such that $X = YW'RZ$, $ED(W[:i],W') \leq z$
2:     $I \leftarrow set()$
3:     if $z < 0$ then return $\emptyset$                                  ▷ no occurrences found
4:     if $i < 0$ then return ($[k,l]$)                    ▷ returning interval of the actual match
5:     $I \leftarrow$ Inexact Recurrence($X, W, i-1, z-1, k, l, SA, B$)                     ▷ Insertion
6:     for each $ch \in \{A, C, G, T\}$ do
7:         $k_p \leftarrow C(ch) + O(ch, k-1) + 1$
8:         $l_p \leftarrow C(ch) + O(ch, l)$
9:         if $k_p \leq l_p$:
10:            $I \leftarrow I \cup$ Inexact Recurrence($X, W, i, z-1, k_p, l_p, SA, B$)          ▷ Deletion
11:            if $ch = W[i]$ then
12:                $I \leftarrow I \cup$ Inexact Recurrence($X, W, i-1, z, k_p, l_p, SA, B$)       ▷ Match
13:            else
14:                $I \leftarrow I \cup$ Inexact Recurrence($X, W, i-1, z-1, k_p, l_p, SA, B$)     ▷ Mismatch
15:    return I
```

3.3 Related Work

Recent progress in the study of the human genome not only caused a revolution in biomedical science, but also introduced new privacy concerns. The problem of oblivious genomic data processing within a *MPC* context could be considered a relatively new research domain. Before explaining our novel approach we analyze related work for privacy and security of sensitive genomic records.

The proposed solutions for privacy-preserving genomic data processing can be separated as follows [27]:

- A private query on private genomic data, e.g. inspection for a DNA mutation regarding a specific illness with secret parameters in private patient-specific genomic data.
- A private query on public genomic data, e.g. searching with secret parameters in public healthcare data obtained from human genome libraries.

In the first case it is assumed that private patient-specific genomic data can reveal additional information about the data owner such as date of birth, address, physical attributes (height, weight, blood type, etc.) apart from the sensitive genomic data. Several approaches have been proposed to tackle this problem.

The work of Goodrich [43] introduces an attack model for the private query on private genomic data scenario. The author's approach shows that privacy-preserving protocols reveal extra information about sensitive genomic data. The attack method succeeds by repetitively querying victim participants until one obtains significant information from the protocol output. The possibility of this attack to succeed is high where in realistic use cases actual genomic data can be revealed by a number of queries. This particular attack method also applies to our proposed privacy-preserving approach.

Atallah et al. introduce the first approach for oblivious genomic data processing [1]. They utilize a new edit distance protocol and develope a new protocol in which no party reveals any information about their private input to other parties. Our approach shares some similarities with this work to compute the edit distance obliviously, e.g. usage of the dynamic programming recurrence relation. On the other hand, our privacy-preserving protocol applies to a specific sequence alignment algorithm (i.e. *BWT* [71]). A major drawback of their approach [1] is the performance inefficiency [60]. We observe the same problem in our approach (see Section 3.6.1).

A new strategy to preserve privacy with the Smith-Waterman DNA sequence comparison algorithm is introduced in [99]. This approach identifies similarities between compared sequences with considerable reduction in the amount of false positives with performance comparable to other sequence comparison protocols. The important point to obtain in this work is the selection of the algorithm (Smith-Waterman) for sequence similarities, which is also the reference algorithm in one of the utilized *BWT* algorithms (i.e BWA-SW). However, the method reveals information during a computation procedure and therefore can not achieve strong privacy. On the contrary, our approach achieves full privacy during both the computation and the protocol verification process.

Jha et al. [60] propose privacy-preserving techniques that achieve secure genomic data processing and edit distance computation on sequences. They develope three different protocols for this. The first protocol uses Yao's garbled circuit [114] by exploiting its internal structure in order to achieve secure circuit evaluation. The second protocol applies the method of secure computation with shares [83, Chapter 7] where participants of a circuit can combine their randomly chosen shares to construct the actual bit value of the output. Finally, the third protocol merges the first two protocols in order to handle scalability issues and improve efficiency of the computation. However, an application of this method [60] within a multi-party context could be considered problematic due to an inability to deal with large-scale computations.

Kerschbaum [29] proposes a scheme to efficiently detect and mitigate the attack introduced by Goodrich [43]. The method preserves the privacy of both parties while detecting similarities in genomic inputs using the combination of two cryptographic primitives: fuzzy commitments and secure computations of edit distance. In addition, a zero-knowledge proof prevents client detection and ensures that both parties used the same input [27]. This contribution is similar to our novel approach to verify the correctness of the oblivious genomic data processing. On the other hand, we have used secure computations on edit distance and succinct [109] for verification of our proposed approach.

For the second case regarding a private query on a public database no known solution exists to our knowledge. However, this particular problem can be considered as part of the above defined problem of a private query on a private genomic database. Hence, above solutions can be applied [27].

3.4 BWT Short-Read Alignment with Multi-Party Computation

The ability to implement the MPC algorithm using TUeVIFF is a requirement. Because computations on a string of characters are not support in the VIFF framework, input strings should be represented as arrays of numbers modulo some prime P in order to be used in the BWT algorithm. Here, DNA sequences consist of the characters A,C,G,T encoded with $1,2,3,4$, respectively. Additionally, 0 represents the special character '\$'. This special character is not present anywhere within the sequence, but appears only at the end.

3.4.1 Private Inexact Search

An inexact search is equivalent to the search for the SA interval of substrings of the reference string X that matches the search query W, allowing z mismatches or differences. The private inexact search algorithm looks for W as private data in the public X.

The private inexact search algorithm is defined below in Algorithm 3 and Algorithm 4:

Algorithm 3 Private Inexact Search: Bounded Traversal/Backtracking

```
1: procedure PRIVATE INEXACT SEARCH(INPUT: $X, [\![W]\!], z, SA, B$; OUTPUT: $[\![I]\!]$)
     ▷ Pre: $X$: reference string, $[\![W]\!]$: search term, $z$: upper mismatch bound, $SA$: suffix array for $X$, $B : BWT$ transform of $X$
     ▷ Post: $[\![I]\!]$: set of all $SA$ intervals corresponding to $W'$, such that, $\exists Y,Z : X = YW'Z, ED(W,W') \leq z$
2:     $[\![I]\!] \leftarrow$ Private Inexact Recurrence($X$, $[\![W]\!]$, $len(W)-1$, $z$, 0, $len(X)-1$, $SA$, $B$)
3:     return [[I]]
```

Algorithm 4 Private Inexact Search: Bounded Traversal/Backtracking Recurrence

```
1: procedure PRIVATE INEXACT RECURRENCE(INPUT: $X, [\![W]\!], i, z, k, l, SA, B$; OUTPUT: $[\![I]\!]$)
     ▷ Pre: $X$: reference string, $[\![W]\!]$: search term, $i$: index in $W$, $z$: upper mismatch bound, $[k,l]$ : $SA$
       interval where $0 \leq k \leq l \leq n$, $SA$: suffix array for $X$, $B$: $BWT$ transform of $X$
     ▷ Post: let $R$ be the shortest string that corresponds to $SA$ interval $[k,l]$ , $[\![I]\!]$: set of all $SA$ intervals
       corresponding to $W'$ such that, $\exists Y, Z$: $X = YW'RZ$, $ED(W[:i], W') \leq z$
2:     $[\![I]\!] \leftarrow$[0],...,[0]
3:     if $z < 0$ then return $[\![I]\!]$                                   ▷ no occurrences found
4:     if $i < 0$ then
5:         for each $j \in \{k, \dots, l\}$ do
6:             $[\![I]\!]_j \leftarrow [1]$
7:         return $[\![I]\!]$                                   ▷ returning interval of the actual match
8:     $[\![I]\!] \leftarrow$Private Inexact Recurrence($[\![W]\!], i-1, z-1, k, l, SA, B$)            ▷ Insertion
9:     for each $ch \in \{A, C, G, T\}$ do
10:        $k_p \leftarrow C(ch) + O(ch, k-1) + 1$
11:        $l_p \leftarrow C(ch) + O(ch, l)$
12:        if $k_p \leq l_p$:
13:          $[\![I_1]\!] \leftarrow$Private Inexact Recurrence($[\![W]\!], i, z-1, k_p, l_p, SA, B$)          ▷ Deletion
14:          $[\![I]\!] = logicor([\![I]\!], [\![I_1]\!])$
15:          $[\![equality]\!] = (ch == [\![W]\!]_i)$
16:          $[\![I_2]\!] \leftarrow$Private Inexact Recurrence($[\![W]\!], i-1, z, k_p, l_p, SA, B$)          ▷ Match
17:          $[\![I_3]\!] \leftarrow$Private Inexact Recurrence($[\![W]\!], i-1, z-1, k_p, l_p, SA, B$)        ▷ Mismatch
18:          for each $j \in \{[0], \dots, [n]\}$ do
19:              $[\![I_4]\!]_j = (1 - [\![equality]\!]) * [\![I_3]\!]_j + [\![equality]\!] * [\![I_2]\!]_j$
20:          $[\![I]\!] \leftarrow logicor([\![I]\!], [\![I_4]\!])$
21:    return $[\![I]\!]$
```

Let us analyze the main differences in input and output, which makes our private inexact search algorithm more secure than the original version of the algorithm. In this version only W is private input while all other input remains public. Another point is the public set returned in each recursion of the original algorithm. This is converted into a secret shared bit vector in the private inexact search (Algorithm 4). A union of these bit vectors is obtained by simply taking $\vee(OR)$ element-wise. Overlap between two bit vectors I_1 and I_2 can be handled as follows:

1. As an initial idea, we can think of simply adding I_1 and I_2 entry-wise. However, in overlapping positions the sum of entries will result in 2, which is not a valid bit vector entry anymore. When the result is opened later these numbers will leak some information.

2. Instead, we can take the entry-wise logical OR of the bit vectors I_1 and I_2. To ensure the obliviousness of this procedure consider the formula $\{a + b - a * b\}$ for each two entries from the bit vectors. This operation costs only 1 multiplication for each entry while addition is for free. Hereby, this computes a new set as a union of two sets (e.g. Algorithm 4, line 14)

This is possible by the simple *logicor* function implemented in Algorithm 5:

Algorithm 5 LogicOR function

```
1: procedure LOGICOR(INPUT: $[\![list1]\!], [\![list2]\!]$; OUTPUT: $[\![res]\!]$)
   ▷ Pre: $[\![list1]\!], [\![list2]\!]$: vector
   ▷ Post: $res$ : union of the two input lists: $res$
2:     $res \leftarrow [0], \dots, [0]$                                        ▷ initialization
3:     for each $i \in \{0, \dots, len(list1)\}$ do
4:         $[\![res]\!]_i \leftarrow [\![list1]\!]_i + [\![list2]\!]_i - [\![list1]\!]_i * [\![list2]\!]_i$      ▷ formula: a+b-a*b
5:     end for
6:     return $[\![res]\!]$                                    ▷ union of two lists returned
```

In Line 11-14 of the original inexact search we execute either a ‘match’ or ‘mismatch’ recursion call depending on whether the character *ch* from W is equal to the one in X or not. However, in the private inexact search this distinction is not allowed. In order to not reveal any information about the decision, i.e. if *ch* is equivalent to the queried character in W, we need to execute both recursions and continue with the rest of function.

The VIFF *sgn* comparison function can be used to obtain a one bit value for the decision in Algorithm 4 (Line 15). It results in 1 if character *ch* from X is equivalent to the current character $W[i]$ and 0 otherwise. However, after detailed analysis of different comparison methods in VIFF as a case study, we introduce an interpolation based comparison that might increase efficiency approximately by 5 times at least. The motivation behind this comparison mainly depends on the fact that compared strings can only consist of 5 different characters, namely $\{\$, A, C, G, T\}$. Therefore, we have introduced new *compare_eq* comparison function.

We define the oblivious equality comparison function that maintains one formula for all possible results of the comparison. This is possible due to the Lagrange interpolation formula, as the range for the possible inputs is small and therefore allows efficient comparison by interpolation. All possible input comparison decisions (x) and corresponding outputs(res) can be defined as:

$$res = \begin{cases} 0, & \text{if } -4 \le x \le -1, \text{or if } 1 \le x \le 4 \\ 1, & \text{if } x = 0 \end{cases} \tag{7}$$

Given equation (7), we have used Wolfram Alpha to generate the unique interpolation formula. Let us define the *compare_eq* function in order to succeed on character-wise oblivious comparison for equality:

Algorithm 6 Character-wise oblivious DNA comparison for equality

```
1: procedure $compare\_eq$(INPUT: $[\![el1]\!], [\![el2]\!]$; OUTPUT: $[\![res]\!]$)
   ▷ Pre: $[\![el1]\!], [\![el2]\!] \in \{0, 1, 2, 3, 4\}$
   ▷ Post: $[\![res]\!]$: 1 if $(el1 = el2)$, 0 otherwise
2:     $[\![x]\!] \leftarrow ([\![el1]\!] - [\![el2]\!]) * ([\![el1]\!] - [\![el2]\!])$
3:     $[\![res]\!] \leftarrow ([\![x]\!] - 1) * ([\![x]\!] - 4) * ([\![x]\!] - 9) * ([\![x]\!] - 16)/576$
       ▷ Interpolation formula for all possible inputs
4:     return $[\![res]\!]$
```

The private inexact search considers both input and output of the C and O functions as public and therefore the functions are computed normally as described in earlier sections.

3.4.2 Fully Privacy-Preserving Inexact Search

The fully privacy-preserving inexact search assumes both search query W and reference string X to be private data. Prior to investigating the proposed approach in more detail, an oblivious inverse transformation algorithm has been implemented (see Algorithm 7). This case study has been helpful to understand secret indexing and masking techniques in order to be applied in *BWT* as a next step.

Algorithm 7 Oblivious IBWT

```
1: procedure $privateIBWT$(INPUT: $[\![B]\!], [\![SA]\!]$; OUTPUT: $[\![X]\!]$)
       ▷ Pre: $[\![B]\!]$: last column of the $BWM$, $[\![SA]\!]$: suffix array
       ▷ Post: $[\![X]\!]$: reference string without '$'
2:     $[\![X']\!] \leftarrow [0], \dots, [0]$
3:     for each $index \in \{0, \dots, len(SA)\}$ do
4:         $[\![target\_index]\!] = [\![SA[index]]\!]$
5:         $[\![X'[target\_index]]\!] \leftarrow [\![B[index]]\!]$
6:     end for
7:     $[\![X]\!] \leftarrow [\![X'_1]\!], \dots, [\![X'_{len(X')-1}]\!]$        ▷ eliminate '$'
8:     return $[\![X]\!]$
```

Algorithm 8 Private *BWT*

```
1: procedure $privateBWT$(INPUT: $[\![X]\!]$; OUTPUT: $[\![B]\!], [\![SA]\!]$)
       ▷ Pre: $[\![X]\!]$: vector including also '$'
       ▷ Post: $[\![B]\!]$: transform, last column of the $BWM$, $[\![SA]\!]$: suffix array
2:     for each $i \in \{0, \dots, len(X)\}$ do
3:         $[\![tab]\!]_i = [\![X[i+1, len(X)]]\!] \mid\mid [\![X[0, i+1]]\!] \mid\mid [\![i]\!]$        ▷ create table of cyclic rotations
4:     end for
5:     $[\![s\_tab]\!] \leftarrow$ lexicographic sort of $[\![tab]\!]$ with $comparelst$        ▷ sorted table
       ▷ get transform
6:     $B \leftarrow [\![s\_tab]\!]_{0,len(X)-2}, \dots, [\![s\_tab]\!]_{len(X)-1,len(X)-2}$
       ▷ get suffix array
7:     $SA \leftarrow [\![s\_tab]\!]_{0,len(X)-1}, \dots, [\![s\_tab]\!]_{len(X)-1,len(X)-1}$
8:     return $[\![B]\!], [\![SA]\!]$
```

There are two major differences between the simple and privacy-preserving model of BWT:

- Private SA computation: in private *BWT*, although the rows are public, during execution of cyclic rotations the ordering of indices in *BWM* results into private data, namely the suffix array which is secured by a *secret indexing* function.
- Secure comparison-based sort: a variant of the bitonic sort which exists in original VIFF documentation has been applied. The main change is the adaptation of the function to the matrix sorting instead of sorting lists as in the original version. Given the function input from the set $\sum = \{A, C, G, T\}$ the efficient interpolation method is used for the comparisons. As the elements are secret shares, by giving all possible comparisons as input for selected lists, collecting all possible outputs in one single formula maintains the concept of the interpolation. The possibilities for input and output can be defined in following form.

Let us define the *comparelst* function in order to succeed on interpolation-based oblivious comparison for bitonic sort:

Algorithm 9 Oblivious DNA comparison: sequence

```
1: procedure $comparelst$(INPUT: $[\![lst1]\!], [\![lst2]\!]$; OUTPUT: $[\![res]\!]$)
     ▷ Pre: $[\![lst1]\!], [\![lst2]\!]$ :sequence including '\$'
     ▷ Post: $[\![res]\!]$: 1 if $(lst1 < lst2)$ lexicographically, 0 otherwise
2:     $[\![x]\!] \leftarrow [\![lst1]\!]_{[0]} - [\![lst2]\!]_{[0]}$
       ▷ In case of equality
3:     $[\![r]\!] \leftarrow comparelst([\![lst1]\!][1:], [\![lst2]\!][1:])$
       ▷ Interpolation formula for all possible inputs
4:     $[\![res]\!] \leftarrow (70 * [\![r]\!] * ([\![x]\!] * [\![x]\!] - 1) * ([\![x]\!] * [\![x]\!] - 4) * ([\![x]\!] * [\![x]\!] - 9) * ([\![x]\!] * [\![x]\!] - 16) - [\![x]\!] * (1066 + 5 * [\![x]\!] * (205 + [\![x]\!] * (66 + 7 * [\![x]\!])))) / 40320$
5:     return $[\![res]\!]$
```

It is important that the sort must be executed for the entire row to ensure security even if the decision could already have been made at the first character comparison between *BWM* rows. If comparison of the elements at Line 2 of Algorithm 9 results in equality then the result is ignored and the algorithm recursively runs again with the remaining elements of both lists as input parameters. As the range for the possible inputs is small, comparison can be done efficiently using one general interpolation formula. This is possible due to the concept of polynomial interpolation evaluation, in particular, Lagrange interpolation formula. All possible input comparison decisions (x) and corresponding outputs (res) can be defined below (recursion defined as r):

$$res = \begin{cases} 0, & \text{if } -4 \le x \le -1 \\ r, & \text{if } x = 0 \\ 1, & \text{if } 1 \le x \le 4 \end{cases} \tag{8}$$

Given equation 8, we used Wolfram Alpha to generate the unique interpolation formula represented in Algorithm 9 (Line 4).

Previously, the private inexact search (Algorithm 4) considered C and O as public and calculated them as a function each time before starting execution of recursions. However, in the fully privacy-preserving inexact search the algorithm uses secret shares as input to these functions which results in secret shared C and O functions. Therefore, the functions C and O should be pre-computed for all possible arguments and passed as parameters. Further, they should be defined as dictionaries instead of calculating them as a function each time before executing recursions. Let us analyze how this problem is solved:

- Recall that the $C(ch)$ function uses character comparison to compute lexicographically smaller elements for each secret ch in private reference string X. Element-wise comparison (*compare_el*) is defined specifically for the C function, as it gets only one secret shared element $ch \in \{\$, A, C, G, T\}$ as an input value. To achieve oblivious comparison we define the function that maintains one formula for all possible results of the comparison given all possible inputs:

$$res = \begin{cases} 1, & \text{if } -4 \le x \le -1 \\ 0, & \text{if } x = 0 \end{cases} \tag{9}$$

Given equation 9, we have used Wolfram Alpha to generate the unique interpolation formula. Let us define *compare_el* function (Algorithm 10) in order to succeed on a character-wise oblivious comparison:

Algorithm 10 Character-wise oblivious DNA comparison

```
1: procedure compare_el(INPUT: $[\![el1]\!], [\![el2]\!]$; OUTPUT: $[\![res]\!]$)
   ▷ Pre: $[\![el1]\!], [\![el2]\!] \in \{0,1,2,3,4\}$: representing $\{\$, A, C, G, T\}$
   ▷ Post: $[\![res]\!]$: 1 if $(el1 < el2)$, 0 otherwise
2:     $[\![x]\!] \leftarrow [\![el1]\!] - [\![el2]\!]$
3:     $[\![res]\!] \leftarrow (x-4)*(x-3)*(x-2)*(x-1)*(0-x*(1066+5*x*(205+x*(66+7*x))))/40320$
       ▷ Interpolation formula for all possible inputs
4:     return $[\![res]\!]$
```

To understand the motivation behind the C function, let us analyze Lines 2-5 of Algorithm 11. After initialization of C as dictionary in Line 7, the algorithm updates the values of C corresponding to the keys from character set {\$, A, C, G, T }. To tackle this problem, in Line 5 the value of C is updated with the comparison result of the *compare_el* function. This finalizes the computation of the elements that are alphabetically smaller than *ch* in X.

- The comparison function (*compare_eq*) is used for the O function, as it gets as input values the secret shared element $ch \in \{\$, A, C, G, T\}$ and the start-point of the *SA* interval, i.e. k. Recall, the O function computes the number of occurrences of input character *ch* in the *BWT* transform with the given index k, i.e. $[\![B[0 : [\![k]\!]]]\!]$.

  To understand the motivation behind the O function, let us analyze Lines 6-11 of Algorithm 11. After initialization of O as dictionary in Line 7 the algorithm sets the count and increments it with the comparison result of the *compare_el* function. In particular, the secret variable *count* computes number of occurrences of *ch* in secret shared transform B and updates value of the function O with the corresponding key accordingly in line 11.

Let us also define the full private inexact search algorithm with Algorithm 11 and Algorithm 12.

While analyzing the stop conditions of the algorithm it can be clearly observed that in order to point all *SA* intervals, i.e. $[k,l]$, all character comparisons should be made. In previous versions of the inexact search algorithm this interval can be determined by succeeding two comparisons in the public string X (e.g. Algorithm 4, Line 5-6). However, as in the new full private version X is also private, *SA* interval, k and l should be secret shares, therefore they cannot be opened during the function execution. Therefore, comparison should be held on all elements of X in order to keep obliviousness. That is, checking every entry one by one and to put all 1's for $[k,l]$ distances in a bit vector (Algorithm 12, Line 6-8).

In Line 12-13, the pre-computation step for the k_p and l_p formulas, i.e. the computation of the O function for both k and l, succeeds. As k_p and l_p values become secret shares during the execution of the algorithm, comparison on these values is stored as a binary decision at *cmp*3 (see line 16, Algorithm 12). Later in the algorithm, we apply this binary decision value to all elements of a result set, thus it does not reveal any information about private values (see line 25).

Algorithm 11 **Full Privacy-Preserving Inexact Search: Bounded Traversal/Backtracking**

```
1:  procedure INEXACT SEARCH(INPUT: $[\![X]\!], [\![W]\!], z, [\![SA]\!], [\![B]\!]$; OUTPUT: $[\![I]\!]$)
      ▷ Pre: $[\![X]\!]$: reference string, $[\![W]\!]$: search term, $z$: upper mismatch bound, $[\![SA]\!]$: suffix array for $X$, $[\![B]\!] : BWT$ transform of $X$
      ▷ Post: $[\![I]\!]$: set of all $SA$ intervals corresponding to $W'$, such that $X = YW'Z$, $ED(W, W') \leq z$
      ▷ $C$ function as dictionary
2:      for each $ch \in \{1,2,3,4\}$ do: $[\![C_{ch}]\!] \leftarrow [0]$
3:      for each $char \in \{[\![X_0]\!], \dots, [\![X_{|X|-1}]\!]\}$ do
4:          for each $ch \in \{1,2,3,4\}$ do
5:              $[\![C_{ch}]\!] \leftarrow [\![C_{ch}]\!] + compare\_el(ch, [\![char]\!])$
6:      for each $ch \in \{1,2,3,4\}$ do
          ▷ $O$ function as dictionary, keys length of $B$
7:          $[\![O_{ch}]\!] \leftarrow [0], \dots, [0]$
8:          $count = 0$
9:          for each $j \in \{0, \dots, [\![len(B)]\!]\}$ do
10:             $[\![count]\!] \leftarrow [\![count]\!] + [\![compare\_eq([\![B]\!]_{[\![j]\!]}, [\![ch]\!])]\!]$
11:             $[\![O_{ch}]\!]_j \leftarrow [\![count]\!]$                    ▷ store value for $ch$
12:     $[\![I]\!] \leftarrow$ Full Inexact Recurrence($[\![X]\!]$, $[\![W]\!]$, $len(W)-1$, $z$, $[\![0]\!]$, $[\![len(X)]\!]-1$, $[\![SA]\!]$, $[\![C]\!]$,$[\![O]\!]$ )
13:     return $[\![I]\!]$
```

3.5 Correctness Proof and Adaptive Function Evaluation

The goal of the proof function is to evaluate the correctness of the private inexact search (Algorithm 4) and the full private inexact search algorithm (Algorithm 11) results. SODA deliverable D1.2 and [4] provide a method to provide such a proof for the *edit script*, which contains the full sequence of operations including matches and mismatches.

3.6 Discussion

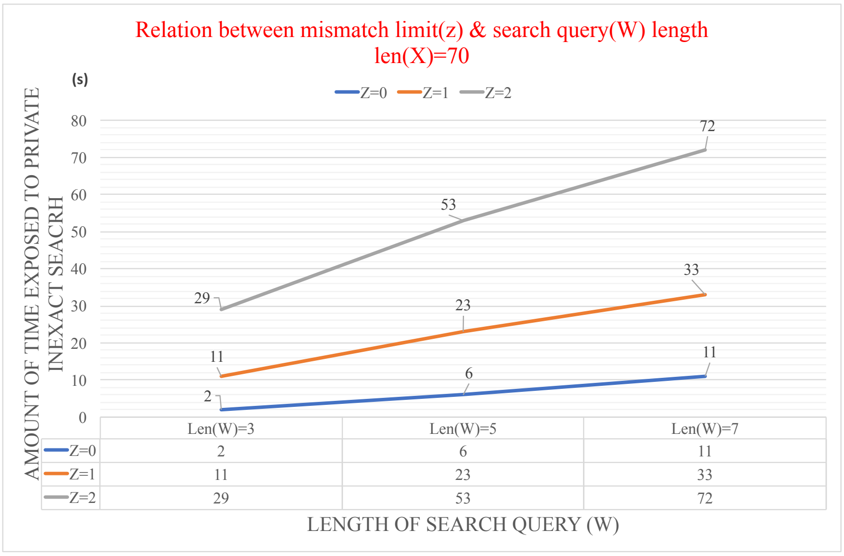
Here we sketch some of the performance results. Details are provided in [4] together with a discussion on further optimizations.

3.6.1 Performance Analysis

Figure 3 and Figure 4 show the performance difference between a VIFF implementation of the private inexact search algorithm for one-party (testing only) and multi-party computation. Figure 5 shows that the fully private MPC variant takes longer. A longer reference string is included in Figure 2 to represent a more realistic scenario, but which would take relatively long to compute in a MPC setting as shown in e.g. Figure 6. [4] has a more elaborate analysis. Yet, these experimental results show that MPC techniques are tractable on genomic sequences of up to hundred characters in length.

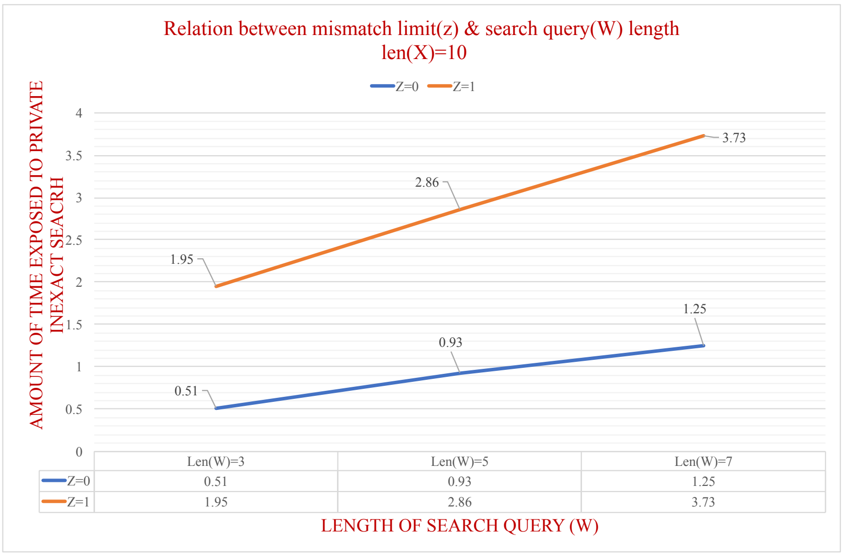
Algorithm 12 **Full Privacy-Preserving Inexact Search Recurrence**

```
1: procedure FULL INEXACT RECURRENCE(INPUT: $[\![X]\!], [\![W]\!]$, $i$, $z$, $[\![k]\!]$, $[\![l]\!]$, $[\![SA]\!]$, $[\![C]\!]$, $[\![O]\!]$; OUTPUT: $[\![I]\!]$)
   ▷ Pre: $[\![X]\!]$: reference string, $[\![W]\!]$: search term, $i$: index in $W$, $z$: upper mismatch bound, if $k < l$, then $[\![[k,l]]\!]$ : $SA$ interval, $[\![SA]\!]$: suffix array for $X$, $[\![C]\!], [\![O]\!]$: pre-computed $C$ and $O$ functions by Algorithm 11
   ▷ Post: : if $k < l$, let $R$ be the shortest string that corresponds to $SA$ interval of $[\![[k,l]]\!]$, $[\![I]\!]$: set of all $SA$ intervals corresponding to $W'$ such that $X = YW'RZ$, $ED(W[:i], W') \leq z$
2:     $[\![I]\!] \leftarrow$[0],...,[0]
3:     if $z < 0$ then return $[\![I]\!]$                                  ▷ no occurrences found
4:     if $i < 0$ then:
5:         for each $j \in \{[0], \ldots, [n]\}$ do
6:             $[\![cmp1]\!] \leftarrow \{j \geq [\![k]\!]\}$                  ▷ store binary decision 1
7:             $[\![cmp2]\!] \leftarrow \{j < [\![l]\!]\}$                    ▷ store binary decision 2
8:             $[\![I]\!]_j = [1] * ([\![cmp1]\!] * [\![cmp2]\!])$
9:         return $[\![I]\!]$                       ▷ returning interval of the actual match
       ▷ Insertion
10:    $[\![I]\!] \leftarrow$ Full Inexact Recurrence($[\![W]\!], i-1, z-1, [\![k]\!], [\![l]\!], [\![SA]\!], [\![C]\!], [\![O]\!]$)
11:    for each $ch \in \{A, C, G, T\}$ do
12:        $[\![res1]\!] = [\![O_{ch}]\!]_{[\![k]\!]-1}$
13:        $[\![res2]\!] = [\![O_{ch}]\!]_{[\![l]\!]-1}$
14:        $[\![k_p]\!] \leftarrow [\![C_{ch}]\!] + 1 + ([\![res1]\!]$ if $i != len(W) - 1$ else $0)$
15:        $[\![l_p]\!] \leftarrow [\![C_{ch}]\!] + [\![res2]\!]$
16:        $[\![cmp3]\!] = \{[\![k_p]\!] \leq [\![l_p]\!]\}$          ▷ store binary decision 3 for $k_p$ and $l_p$
       ▷ Deletion
17:        $[\![I_1]\!] \leftarrow$ Full Inexact Recurrence($[\![W]\!], i, z-1, [\![k_p]\!], [\![l_p]\!], [\![SA]\!], [\![C]\!], [\![O]\!]$)
18:        $[\![equality]\!] = (ch == [\![W]\!]_{[\![i]\!]})$
       ▷ Match
19:        $[\![I_2]\!] \leftarrow$ Full Inexact Recurrence($[\![W]\!], i-1, z, [\![k_p]\!], [\![l_p]\!], [\![SA]\!], [\![C]\!], [\![O]\!]$)
       ▷ Mismatch
20:        $[\![I_3]\!] \leftarrow$ Full Inexact Recurrence($[\![W]\!], i-1, z-1, [\![k_p]\!], [\![l_p]\!], [\![SA]\!], [\![C]\!], [\![O]\!]$)
21:        for each $j \in \{[0], \ldots, [n]\}$ do
22:            $[\![I_4]\!]_j = (1 - [\![equality]\!]) * [\![I_3]\!]_j + [\![equality]\!] * [\![I_2]\!]_j$
23:        $[\![I_5]\!] \leftarrow logicor([\![I_1]\!], [\![I_4]\!])$
24:        for each $j \in \{[0], \ldots, [n]\}$ do
25:            $[\![I_6]\!]_j = [\![cmp3]\!] * [\![I_5]\!]_j$
26:        $[\![I]\!] \leftarrow logicor([\![I]\!], [\![I_6]\!])$
27:    return $[\![I]\!]$
```



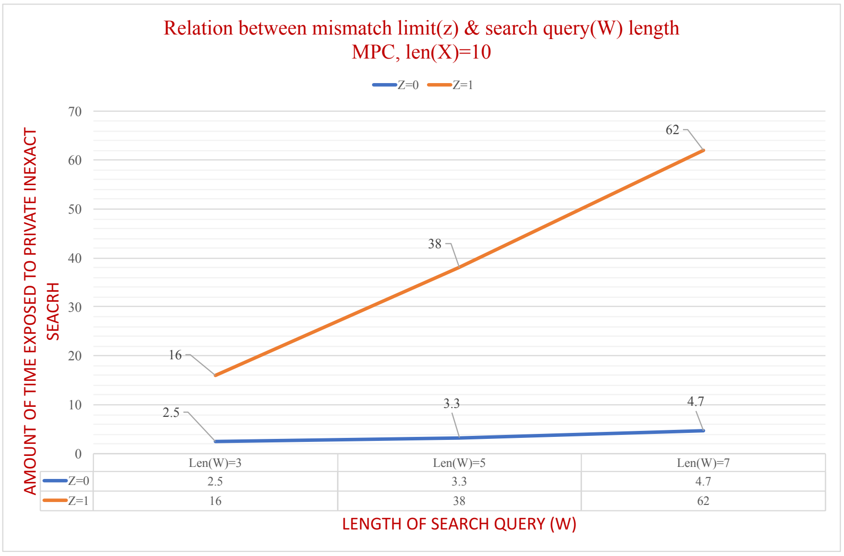
	Len(W)=3	Len(W)=5	Len(W)=7
Z=0	2	6	11
Z=1	11	23	33
Z=2	29	53	72

Figure 2: Time efficiency of Private Inexact Search Algorithm given reference string (length of 70) in one-party scheme



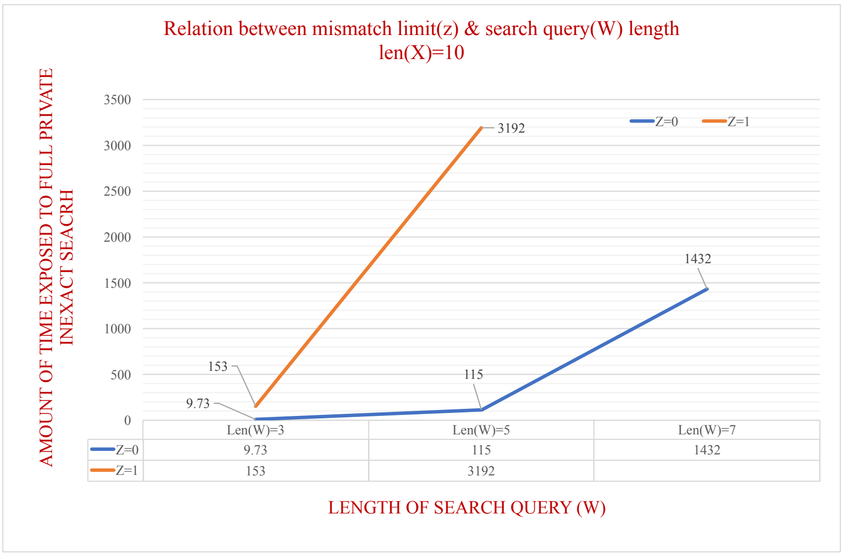
	Len(W)=3	Len(W)=5	Len(W)=7
Z=0	0.51	0.93	1.25
Z=1	1.95	2.86	3.73

Figure 3: Time efficiency of Private Inexact Search Algorithm given reference string (length of 10) in one-party scheme



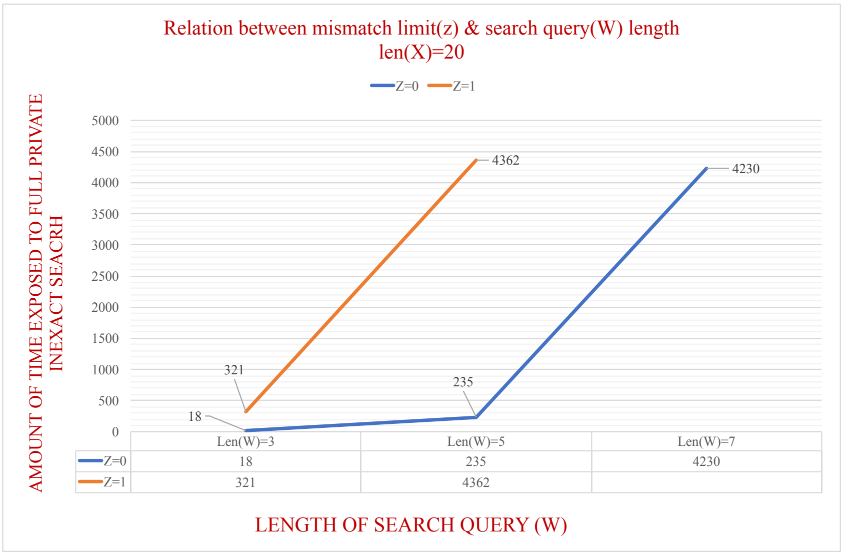
	Len(W)=3	Len(W)=5	Len(W)=7
Z=0	2.5	3.3	4.7
Z=1	16	38	62

Figure 4: Time efficiency of Private Inexact Search Algorithm given reference string (length of 10) in *MPC* scheme



	Len(W)=3	Len(W)=5	Len(W)=7
Z=0	9.73	115	1432
Z=1	153	3192	

Figure 5: Time efficiency of Full Private Inexact Search Algorithm given reference string (length of 10) in one-party scheme



	Len(W)=3	Len(W)=5	Len(W)=7
Z=0	18	235	4230
Z=1	321	4362	

Figure 6: Time efficiency of Full Private Inexact Search Algorithm given reference string (length of 20) in one-party scheme

4 Secure Convolutional Neural Networks

Convolutional neural networks (CNNs) have become a popular and effective class of machine learning algorithms. A well-known application of CNNs is the recognition of handwritten digits. The MNIST dataset of handwritten digits consists of a training set of 60,000 images and of a test set of 10,000 images.

We have investigated the following outsourcing scenario, using the MNIST dataset [18]. First, the training images are used in the clear to obtain a highly reliable CNN classifier. Next, the classifier is used on random test images keeping both the CNN parameters (neuron weights and bias for all layers) and the test image secret. This way the use of the CNN classifier can be safely outsourced, without running the risk of being copied by others. At the same time, the classifier can be run on inputs (handwritten digits) that will also remain hidden. This is useful, e.g., when the digits represent sensitive information, such as credit card numbers.

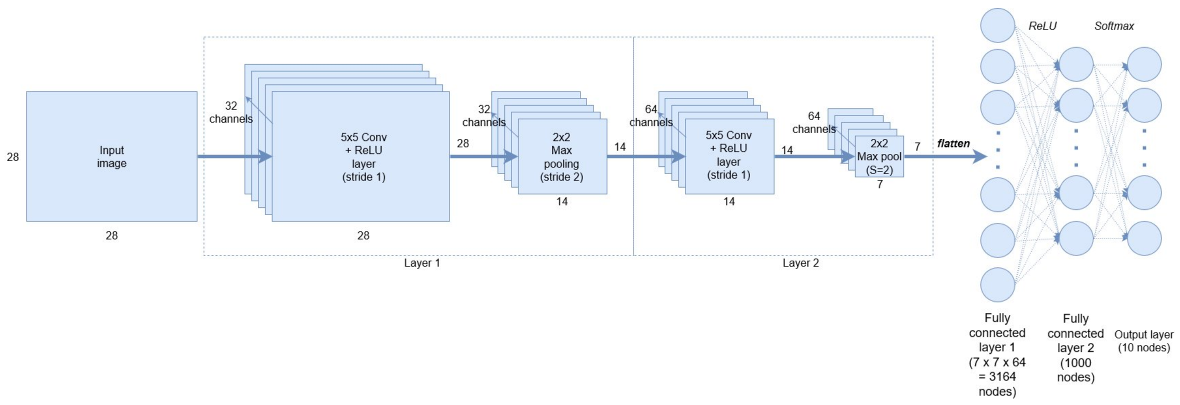

Input image
28
28
32 channels
5x5 Conv + ReLU layer (stride 1)
28
28
32 channels
2x2 Max pooling (stride 2)
14
14
Layer 1
64 channels
5x5 Conv + ReLU layer (stride 1)
14
14
64 channels
2x2 Max pool (S=2)
7
7
Layer 2
flatten
ReLU
Softmax
Fully connected layer 1 (7 x 7 x 64 = 3164 nodes)
Fully connected layer 2 (1000 nodes)
Output layer (10 nodes)

Figure 7: Convolutional Neural Network for MNIST digits (from adventuresinmachinelearning.com).

The structure of a CNN for recognizing handwritten digits is shown in Figure 7. There are two convolutional layers (layer 1 and layer 2) and two fully-connected layers (layer 3 and layer 4). The ReLU function, defined by $ReLU(x) = \max(x, 0)$, is used as activation function for the entire network. Both convolutional layers include a max-pooling stage.

The structure and the dimensions used for each layer is not considered confidential information for our purposes. Hiding the structure and potentially hiding the exact number of layers (of each type) would incur much higher cost. All other works for secure CNNs also reveal the structure of the network (see, e.g., [74]).

We have designed and implemented several alternative secure evaluation of CNNs, either using integer arithmetic only or using fixed-point arithmetic. Basic descriptions of CNNs such as shown in Figure 7 assume floating-point arithmetic. All arithmetic operations are assumed to be done with sufficient precision.

The evaluation of the CNN in Figure 7 can be done entirely using integral numbers only, provided a proper scaling is used. It turns out that a maximum bit length of 37 bits suffices for the entire computation. However, using the maximum bit length simply for all layers is overkill, as the numbers only reach their maximal size in the final layer. Instead, we let the maximum bit length grow as we move from one layer to the next layer. Concretely, we set the maximum bit length to $\ell = 16, 23, 30, 37$ for the four layers of the MNIST CNN. This leads to direct savings for the integer comparisons performed in a secure multiparty computation. In MPyC, we use a common protocol for secure comparison that requires $O(\ell)$ work and $O(\log \ell)$ rounds.

Alternatively, evaluation of the MNIST CNN can be done using fixed-point arithmetic. A main advantage of the use of fixed-point numbers is that we avoid the need for any (re)scaling during the evaluation. We only need to choose the proper size for the fixed-point numbers that are used throughout the entire evaluation (for all layers). We have found that a total bit length of 10 bits, with 4 fractional bits (hence the integer part consists of 6 bits), suffices for reliable results.

We have programmed all our solutions in MPyC. The code is available on GitHub, see

`github.com/lschoe/mpyc/demos/cnnmnist.py`.

The MPyC program `cnnmnist.py` combines the two approaches, using either secure integer arithmetic or secure fixed-point arithmetic. Concretely, we either use the MPyC secure integer type `SecInt(37)` or the MPyC secure fixed-point type `SecFxp(10,4)`. When using integer arithmetic, we set the maximum bit length for each layer by modifying the `bit_length` attribute of the secure integer type. Note that we have also rearranged the applications of the ReLU activation function and the max-pooling stage to minimize the total number of secure comparisons.

MPyC barriers are used to throttle the entire computation. By placing a barrier between two layers, we prevent that the MPyC runtime starts evaluating the next layer before the previous layer is finished. This way the memory usage of the MPyC runtime is limited. The number of barriers placed should be limited, however, to ensure that sufficiently large parts of the entire computation will be evaluated in parallel.

The performance for both solutions is quite similar. On a modern 4-core desktop PC (i7, 4th generation) a sample run with three parties of the integer version runs in 2 mins 51 secs, whereas the fixed-point version runs in 2 mins 56 secs. The maximum memory usage for the fixed-point version is slightly lower than for the integer version. Note that MPyC has not been optimized for performance, but rather for usability. The MPyC code is very similar to a Python program that evaluates MNIST CNNs in the clear.

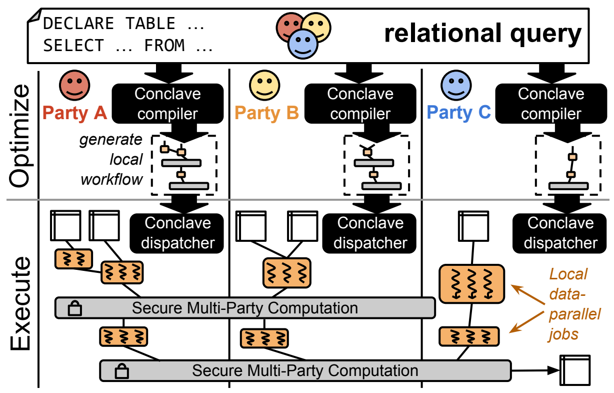


Figure 8: Multiple parties agree on a relational query that Conclave turns into a mix of local and secure MPC steps, generates the code for, and executes.

5 Conclave: Secure Multi-Party Computation on Big Data

5.1 Introduction

Deploying MPC to run joint analytics over large private data sets has a wide range of potential use cases: for example, drug companies, medical researchers, and hospitals can benefit from jointly measuring the incidence of illnesses without revealing private patient data [81, 7]; banks and financial regulators can assess systemic risk without revealing their private portfolios [86, 10]; and antitrust regulators can measure monopolies using companies' revenue data.

However, integrating MPC into the "big data" analytics context at present faces two roadblocks: (*i*) implementing MPC applications requires substantial domain-specific expertise, making it impractical for most data analysts, and (*ii*) existing algorithmic techniques and software frameworks for MPC still do not scale to large data sizes (§5.2) for typical big-data workloads.

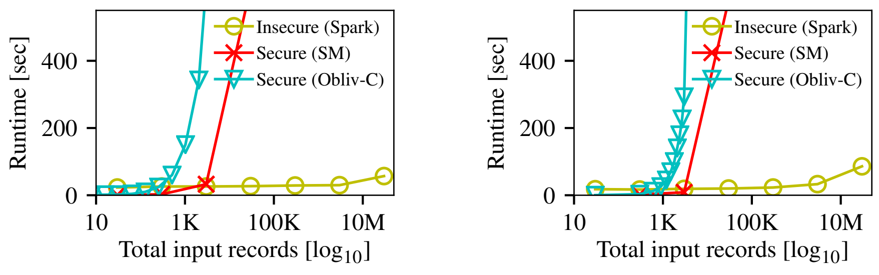
This work presents Conclave, an MPC-enabled query compiler that addresses these problems. Conclave's design focuses on making MPC accessible and efficient: data analysts write relational queries as if they had access to all parties' data in the clear, and the query compiler turns the queries into a combination of efficient local processing steps and secure MPC steps (Figure 8).

Two key ideas help Conclave scale to "big data". First, Conclave *combines* insecure, but fast and scalable, parallel data-processing systems (*e.g.*, Hadoop MapReduce, Spark [117], or Naiad [79]) with secure, but slow, cross-party MPC systems (*e.g.*, Sharemind [13], or Obliv-C [118]). Second, Conclave analyses the queries to apply transformations that optimize runtime without compromising security guarantees, and uses optional, coarse-grained annotations to gain further speedups.

Together, these ideas speed up query execution by processing data sets via high-throughput, data-parallel local computations whenever possible, while combining intermediate results under MPC where necessary. Conclave's automatic division into local and MPC steps relieves data analysts from the need to understand where to place the boundary between local and multi-party computation. Existing MPC frameworks, by contrast, either run the entire computation as an MPC, or require fine-grained annotations on all variables to decide which operations run using cryptographic MPC techniques [118, 13, 91]. Conclave, by contrast, has minimal annotation burden — none by default, and optional column-level annotations on input tables to improve performance.

This work makes four key contributions:

1. the Conclave approach of doing as little work as possible, but as much as necessary, in MPC



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

Figure 9: 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).

(§5.3);

2. automated analyses that derive which parts of a relational query must be executed under MPC using only coarse-grained annotations (§5.4);
3. new "hybrid" MPC–cleartext protocols that improve performance of MPC joins and aggregations using existing partial trust between parties (§5.5); and
4. our prototype implementation of Conclave, which applies these ideas to generate efficient code for execution using sequential Python, Spark [117], Obliv-C [118] and Sharemind [13].

We measured the performance of our prototype using microbenchmarks and end-to-end relational analytics queries (§5.6). Even with minimal annotations on input relations and basic optimizations, Conclave scales queries to input data orders of magnitude larger than existing MPC frameworks can support. Our new "hybrid" MPC–cleartext protocols speed up join and aggregation operators by 7× or more compared to execution in Sharemind [13], a fast, commercial MPC framework.

5.2 Motivation

MPC has served purposes from detecting VAT tax fraud by analyzing business transactions [11], to setting sugar beet prices via auction [14], and evaluating the gender pay gap across businesses [9]. Although these applications compute only over data of a few hundreds or thousands of records[5], some useful computations on large data would benefit from MPC, but are currently infeasible.

Further, we sketch two such example big-data applications of MPC.

Credit card regulation. A government regulator (in the U.S., the OCC [106]) 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 currently issued in the U.S. [107] and at least 167M credit cards [104].

[5]A notable exception is the deployment described in [12] where several millions of tax and education records were processed using Sharemind; the computation however took about two weeks to complete.

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 [105, §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 [82]. While the inputs to the HHI computation are small (a single number per company), computing them requires filtering and aggregating over millions of trip records that companies keep private.

Scalabale processing systems such as Apache Spark [117] allow a data analyst to implement the above applications in a high-level query language, for instance SQL or LINQ [77]. In the context of SQL (and other relational languages), data is represented as *relations*, *i.e.*, collections of rows where each row represents a data record (for instance information about a single airport transfer) and a column represents an attribute (for instance the time of a transfer). SQL queries consist of *relational operators* such as joins, aggregations, and projections which allow an analyst to manipulate relations.

To make MPC more accessible to data analysts without MPC expertise, it is apt to provide similar abstractions; this requires integration with a SQL-like front-end language and MPC implementations of common relational operators. Realizing these operators as MPC protocols however is not trivial — a join on private records for instance requires to *obliviously* link two relations on key attributes; an aggregation requires rows with equal attributes to be obliviously grouped together.

The requirement to keep the underlying algorithms data-oblivious, along with the additional performance limitations of current MPC schemes — *i.e.*, the communication overhead of secret-sharing based protocols and the state size explosion of garbled circuits — results in stark scalability limitations, as we show further.

Figure 9 compares insecure plaintext execution of two relational operators to execution in MPC frameworks using secret-sharing (Sharemind [13]) and garbled-circuits (Obliv-C [118]). 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. While Spark processes millions of records in seconds, neither Obliv-C, nor Sharemind MPC scale past a few thousand records for either operator.

These results are consistent with prior studies: Sharemind takes 200s to sort 16,000 elements [62], and DJoin takes an hour to join 15,000 records [81]. 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: over 60% of practical analytics queries use joins, and over 34% contain aggregations [61]. Apart from new cryptographic techniques with better scalability, the best way to run MPC on large data may therefore be to avoid using its cryptographic techniques unless absolutely necessary.

5.3 Conclave overview

The key insight behind Conclave is that the end-to-end security guarantees of MPC can often hold even if parts of a query run outside MPC. This insight allows to use cheaper algorithms, local computation, and scalable, data-parallel processing systems for parts of the query. This is crucial for scaling MPC to large data sets.

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.

```
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).

Intuitively, any operation computed using only a party's local inputs and publicly available data can run outside MPC, as can any operation that applies only reversible operations to reach a final, revealed result. Conclave detects such cases automatically, generates code for existing MPC and cleartext data analytics frameworks, and manages its execution. This frees the data analyst from picking which operations to run under MPC, and from manually orchestrating several systems.

Like many practical MPC systems, Conclave assumes an *honest-but-curious* adversary.

Conclave also provides performance optimizations that rely on an understanding of *de facto* trust relationships between parties to introduce harmless, deliberate "leakage" that speeds up MPC steps. These optimizations rely on a *semi-trusted party* (STP), who may learn parts of an operator's input data in the clear, and who assists the multi-party computation by performing otherwise expensive operations outside of MPC. For instance, the performance of both joins and aggregations can improve significantly if Conclave can leak their key columns (but no other columns) to an STP. Importantly, however, Conclave guarantees to apply value-leaking optimizations *only* if users supply explicit input annotations that permit deriving an authorization (§5.4.2).

Furthermore, Conclave makes the relaxing security assumption that the size of all input, intermediate, and output relations, *i.e.*, their row count, is public. In other words, Conclave does not hide the size of any relation during execution. Hiding the size of intermediate relations would require padding each relation to a fixed size that exceeds the size of the largest relation, which is prohibitively expensive.

5.4 Specifying Conclave queries

Conclave is a *query compiler* that transforms a relational query into a data processing *workflow*. It is similar to plaintext-only big data query compilers (*e.g.*, Hive [100], Pig [84], or Scope [19]) and

```
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 7–9) and the final result (line 25).

"workflow managers", like Apache Oozie [55] or Musketeer [42]. Like these systems, Conclave transforms the query into a directed acyclic graph (DAG) of relational operators, and executes this DAG on several *backend* systems. Unlike prior query compilers, however, Conclave rewrites the query to improve performance while taking care to preserve MPC's security guarantees.

Conclave assumes that analysts are comfortable writing relational queries using SQL or LINQ [77]; that they agree via out-of-band mechanisms on the query to run; and that all parties faithfully execute the protocol.

Parties locally store input data with the schema expected by the query. Each party runs (*i*) a local Conclave agent, which communicates with the other parties and manages local and MPC jobs, (*ii*) at least one local data processing system (*e.g.*, Hadoop MapReduce, Spark, or Naiad), and (*iii*) a local MPC endpoint (*e.g.*, an Obliv-C or Sharemind node), all on private infrastructure.

5.4.1 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 §5.2 in a DryadLINQ-like language [116].

Even though the input data to a Conclave query is distributed across multiple parties, Conclave largely abstracts this fact away from analysts. Concretely, the analysts specify the query's core as though it was a relational query on a single database stored at a trusted party (lines 14–22 of Listing 1 and lines 13–24 of Listing 2).

The only difference from a relational query is that Conclave also requires identifying at which input relations are located (lines 7–12 and 4–9). Each input relation has a specified "owner", *viz.*, the party storing it. This information helps Conclave (*i*) locate the data, and (*ii*) detect where operations combine data across parties. By combining per-party input relations using a duplicate-preserving set union operator (`concat`, lines 12 and 11), analysts can create compound relations across parties and use them in the query. In addition to inputs, analysts also annotate each output relation with one or more recipient parties. These parties end up storing the cleartext result of executing the query (lines 22 and 25).

Conclave's high-level, declarative query specification contrasts with existing MPC frameworks, which usually provide Turing complete DSLs (*e.g.*, SecreC [57] or Obliv-C [118]). While they are expressive, such interfaces are often unfamiliar to data analysts and require fine-grained security annotations of intermediate variables.

5.4.2 Optional trust annotations

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

The intuition behind trust annotations is that the sensitivity of data within a relation often varies by column. Consider a relation that holds information about a company's branches: it may have public `address` and `zip` columns (as this information is readily available from public sources), but privately-owned columns `isFranchise` or `workforceUnionized`. Other columns may be private, but the owning party might be happy to reveal them to a specific *semi-trusted party* (STP), such as a government regulator. For example, in the credit card regulation query (Listing 1), the government regulator already holds demographic information organized by SSN, and the credit card companies

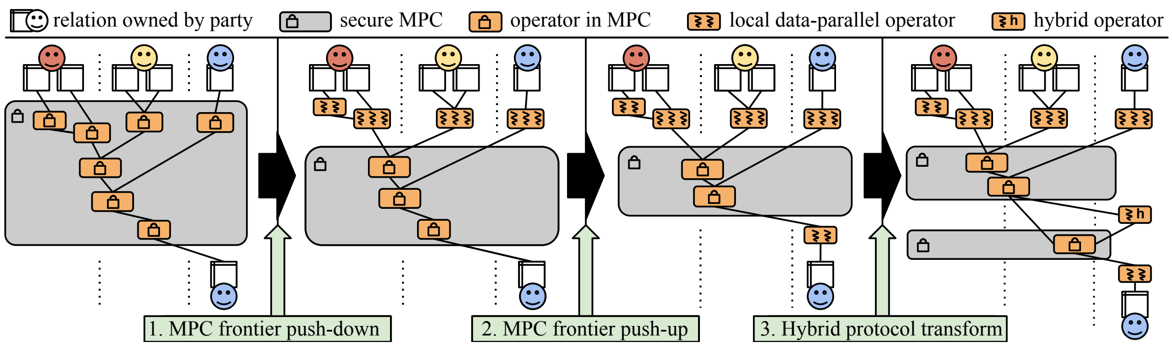


Figure 10: Conclave minimizes the work under MPC by: (*i*) pushing the MPC frontier down and locally preprocessing where possible; (*ii*) pushing the MPC frontier up from the outputs, processing reversible operators in the clear at the receiving party; and (*iii*) inserting special "hybrid" operators that implement efficient hybrid MPC-cleartext protocols.

may be willing to reveal the SSNs of their customers to the regulator (though not to the other parties, *i.e.*, their competitors). Hence, the parties agree to make the regulator a semi-trusted party for the `ssn` column of the credit card companies' customer relations (see line 8, Listing 1). Such selective revealing of columns where permissible can help Conclave avoid, or shrink, expensive MPC steps and substantially improves performance.

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

5.5 Query compilation

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

Conclave analyzes and optimizes a query in four stages (Figure 10); all parties run these deterministically.

1. Conclave starts with a query plan consisting of a single, large MPC. First, it propagates input relation locations to intermediate relations to determine where data crosses party boundaries (§5.5.1).
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 (§5.5.2).
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 MPC-cleartext operators in place of operators that can run partially outside MPC. Conclave then breaks these hybrid operators into cliques of local cleartext operators and secure MPC operators (§5.5.3).
5. 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.

5.5.1 Propagating annotations

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

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

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

In a **second pass**, Conclave propagates trust annotations from input relations in topological order, and combines them using similar rules. Specifically, unary operators propagate their input's trust annotation to all outputs, while multi-input operators intersect their input's trust annotations to determine their outputs. This captures the fact that Conclave can only use hybrid operators to run an MPC with the aid of a semi-trusted party if *all* parties supplying inputs to the MPC trust that semi-trusted party. However, this pass alone is insufficient and can introduce unauthorized leakage. The STP may infer additional information if the output of a hybrid operator (computed with the aid of an STP) is further combined, under MPC, with a relation owned by a party that does not trust the STP with her data. In particular, the STP can combine her knowledge of the plaintext values leaked via the hybrid operator with leaked sizes of intermediate relations. To prevent such leakage, Conclave — after the initial propagation — finds operators that combine relations whose trust sets differ by $D \neq \emptyset$. It traverses all paths from these relations back to inputs, purging any parties in D from trust sets along the paths. The final trust sets allow Conclave to determine when it can use hybrid operators.

5.5.2 Finding the MPC frontier

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

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

Queries often combine inputs from multiple parties into a single, partitioned relation via a `concat` operator. This creates a "virtual" input relation that contains data from all parties (*e.g.*, `scores` on line 12 of Listing 1, and `taxi_data` on line 11 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,

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

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

Other operators, however, do not trivially distribute over the inputs of a `concat`. For example, splitting an aggregation that groups a partitioned relation by key and applying it to the original singleton relations requires another, secondary aggregation to produce identical results. Conclave inserts such a secondary aggregations when possible.[6] While the secondary aggregation must remain under MPC, Conclave can pull the local pre-aggregations out of MPC and significantly reduce the MPC's input data size. Moreover, Conclave can push the operator clique of `concat` and the secondary aggregation (which now forms the MPC frontier) past other distributive operators. In the market concentration query, Conclave pushes the MPC frontier to right above the `market_size` relation, at which point the data amount to only few integers per party.

MPC frontier push-up. In certain cases, Conclave can also push the MPC boundary up from the output relations (*i.e.*, the DAG's leaves). Some relational operators are *reversible*, *i.e.*, given their output, it is possible to reconstruct the input without additional information. For example, multiplication of column values by a fixed scalar factor has this property (provided the factor is $\neq 0$): revealing the multiplied values also reveals the multiplication inputs. Conclave's MPC push-up pass starts at output relations and lifts the MPC frontier through reversible operators by the revealing the input to operator to the output party, thus enabling local computation.

5.5.3 Hybrid operators

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

[6]This transformation leaks the size of the local aggregation result; Conclave currently treats all intermediate relation sizes as public. We plan to investigate padding approaches to alleviate this leakage (at the cost of increased runtime).

In the context of hybrid operators, a party is either (*i*) the STP, or (*ii*) a regular, untrusted party. Conclave deliberately leaks plaintext *values* of some columns to the STP, but maintains the standard MPC privacy guarantees for these columns towards the untrusted parties, and towards all parties for all other columns.

Conclave currently supports two hybrid operators: a *hybrid join*, and a *hybrid aggregation*. In the context of this report we focus on the hybrid join, and summarize the intuition behind the 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 have an STP in common. This STP learns the key columns on *both* sides of the join and computes, in the clear, which keys in the key columns match. The STP hence determines which rows in the secret-shared relations are in the join result without learning any other column values.

More concretely, the STP can compute the *indexes* of the rows of each input relation that comprise the result of the join. Using these indexes, the STP and the untrusted parties can jointly compute the result of the join. To avoid leaking information about the key columns to the untrusted parties, the parties employ an *oblivious selection protocol* (akin to the one by Laud [70]). An oblivious selection allows the parties to use a set of secret indexes to select the corresponding rows from a relation without leaking the indexes.

Before presenting the protocol, we establish some notation. Given a relation R, consisting of c columns and n rows, we denote the i-th column of R as $R_{(i)}$. We index columns starting at 1, *i.e.*, the first column of R is $R_{(1)}$. We denote a secret-shared relation as $[\![R]\!]$.

An MPC join (and similarly a hybrid join), takes as input two relations $[\![L]\!]$ and $[\![R]\!]$ with l and r columns respectively, and two column indexes k_l and k_r designating the *key columns* of the join. The join produces as output relation $[\![J]\!]$ consisting of $l+r$ columns. $[\![J]\!]$ contains all combinations of the rows from $[\![L]\!]$ and the rows from $[\![R]\!]$ where the key column value of the row from $[\![L]\!]$ (the value at index k_l) equals the key column value of the row from $[\![R]\!]$ (the value at index k_r). The first l columns of $[\![J]\!]$ correspond to the columns taken from $[\![L]\!]$ and the remaining r columns correspond to the columns taken from $[\![R]\!]$.

We present the hybrid join protocol in Protocol 7.

Step 1 of the protocol reveals the values of the key columns to the STP, but this exposure is authorized by the input relations' trust annotations.

Step 5 leaks the size of the joined result to the untrusted parties. This does not violate Conclave's privacy requirements since intermediate relation sizes are considered public. The oblivious selection protocol in Step 6 prevents further leakage to the untrusted parties.

The oblivious selection requires $\mathcal{O}((n+m)\log(n+m))$ non-linear operations, where n is the input size and m the result size, in contrast to the standard MPC join protocol, which requires $\mathcal{O}(n^2 \log n)$ non-linear operations (but assumes no STP).

Hybrid aggregation. Conclave can transform an MPC aggregation into a *hybrid aggregation* if the trust set on the group-by column contains an STP, *i.e.*, there is an STP authorized to learn the group-by column values. Conclave's hybrid aggregation protocol adapts the sorting-based MPC protocol by Jónsson *et al.* [62]. Since the STP has cleartext access to the group-by column, Conclave can outsource several work-intensive steps of the original protocol to her, including the oblivious sorting step (which is the main bottle-neck of the original protocol). The hybrid aggregation improves asymptotically over the regular MPC protocol: the oblivious sorting step of the original protocol is based on a sorting network and requires $\mathcal{O}(n \log^2 n)$ oblivious comparisons; in contrast, the hybrid aggregation performs the sort in the clear and only needs an oblivious shuffle which can be realized with $\mathcal{O}(n \log n)$ multiplications [66].

```
Protocol 7 ($[\![J]\!]$) ← HybridJoin($[\![L]\!], [\![R]\!], k_l, k_r$),
$[\![L]\!]$, $[\![R]\!]$ are secret relations
$k_l$ is key-column index of $[\![L]\!]$, $k_r$ is key-column index of $[\![R]\!]$
1: The parties reveal to the STP the key columns of the input relations, $[\![L_{(k_l)}]\!]$, and $[\![R_{(k_r)}]\!]$.
2: Given open key-column relation $L_{(k_l)}$, the STP enumerates its rows, obtaining $EL$. Column $EL_{(0)}$
   contains the keys of the original relation, and column $EL_{(1)}$ contains the row indexes. The STP
   does the same for $R_{(k_r)}$, obtaining $ER$.
3: The STP performs a clear-text join on the key columns of $EL$ and $ER$, obtaining $EJ$. Each row in
   $EJ$ consists of four entries $\{k, i_l, k, i_r\}$, where $k$ is the join-key value, $i_l$ is the index of a row in
   $[\![L]\!]$, and $i_r$ is the index of a row in $[\![R]\!]$.
4: The STP computes $LI \leftarrow EJ_{(2)}$, which contains the indexes of the rows of the left input relation
   $[\![L]\!]$ comprising the result of the join, and $RI \leftarrow EJ_{(4)}$ which contains the indexes of the rows of
   the right input relation $[\![R]\!]$.
5: The STP secret-shares $[\![LI]\!] \leftarrow LI$ and $[\![RI]\!] \leftarrow RI$.
6: Under MPC, the parties perform an oblivious selection protocol (akin to the one by Laud [70])
   using the indexes $[\![LI]\!]$ to select the rows from $[\![L]\!]$ comprising the result of the join. We denote the
   result as $[\![JL]\!]$. The parties do the same for $[\![RI]\!]$, selecting the correct rows from $[\![R]\!]$. We denote
   the result as $[\![JR]\!]$.
7: The parties concatenate the columns of $[\![JL]\!]$ and $[\![JR]\!]$ to obtain the final result of the join $[\![J]\!]$.
```

5.6 Evaluation

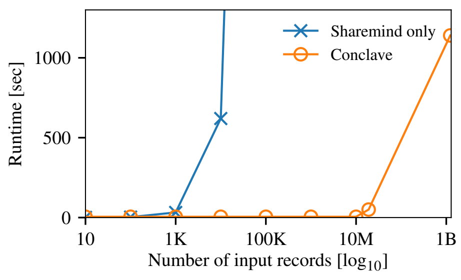
Further, we summarize a preliminary evaluation of Conclave's performance yields using our motivating queries (§5.2).

Setup. We run all 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 and 4 GB RAM, and run Ubuntu 14.04 with Spark 2.2 and Hadoop 2.6. The Sharemind VM has 4 vCPUs and 8 GB RAM, runs Debian Squeeze and Sharemind 2016.12. All machines have Intel Xeon E3 CPUs running at 2.4 GHz, connected by a 10G network.

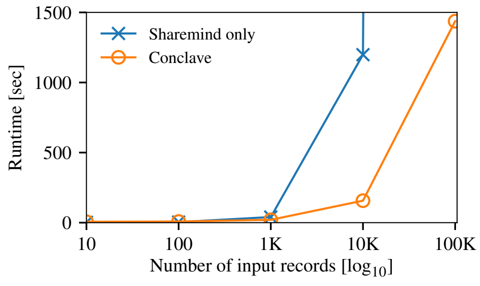
Metrics. All our graphs increase the data size on the *x*-axis by five to eight orders of magnitude, and plot query runtime on the *y*-axis. Less is better in all graphs, and we use a $\log_{10}$-scale *x*-axis to be able to show the scalability limits of different systems on the same graph, even though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

Market concentration query. The market concentration query computes the Herfindahl Hirschman Index (HHI) [50] 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 [94]. We randomly divide the trips across three imaginary VFH companies and filter out any trips with a zero fare. This results in a total of 1.3 billion trips across all parties. We subsample different numbers of rows from the input data sets, and measure the end-to-end query execution time for different input sizes.

Figure 11a shows the results across eight orders of magnitude in input size. It is evident that this query scales poorly when run entirely under MPC in Sharemind: at 100,000 input rows, Sharemind does not complete it within an hour. The query contains an aggregation, and the runtime of this expensive operator (cf. §5.2, Fig. 9) dominates all others. Its poor scalability leads to a rapid increase in execution time at 10,000 records. Conclave, by contrast, scales roughly linearly in the size of the input data, since it pushes the MPC frontier past aggregations for the per-party revenue. All data-intensive processing happens outside MPC in local Spark jobs, and only a handful of records enter the



(a) Market concentration query (§5.4, Listing 2).



(b) Credit card regulation query (§5.4, Listing 1).

Figure 11: Our preliminary performance results show that Conclave's optimizations yield order of magnitude improvements for realistic queries (§5.2).

final Sharemind computation, which consequently completes quickly. In particular, Conclave runs the market concentration query in <20 minutes for 1B input records.

Credit card regulation query. Conclave's hybrid operators can offer substantial benefits for the common case of queries whose performance is dominated by aggregations and joins. The credit card regulation query is an example: it first performs a join between the regulator's demographic information, and then computes an aggregate (*viz.*, the average score grouped by ZIP code). The credit card companies are happy to 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 (*e.g.*, to prevent credit card companies from targeted advertising to their competitors' customers). 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 the runtime as both the complex join and the aggregation work can now happen outside MPC.

Figure 11b confirms this. Running the query entirely under MPC in Sharemind fails to scale beyond 10,000 customers per company; at 100k customers, the query does not complete within two hours. With Conclave's hybrid operators, however, the query completes in about twenty minutes even at these large scales. The experiment also highlights that hybrid operators are crucial to obtaining good performance for this query: since the first operator is a join, Conclave cannot push the MPC frontier further down. Hence, without the hybrid operators, Conclave would have no choice but run the entire query under MPC.

5.7 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 §5.2. 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 [91] performs mixed mode computations that move between MPC and local work. Wysteria programs are written in a DSL that creates the programmer illusion of a single thread of control, but require familiarity with said DSL. Unlike Wysteria, Conclave programs are standard relational queries, and Conclave supports distributed backend systems for parallel processing: it connects with existing big data analytics frameworks like Spark.

Query rewriting. There are several efforts to optimize MPC via query rewriting, like Conclave does, at different levels of abstraction. Kerschbaum [67] 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 [7] and Opaque [119]. SMCQL, like Conclave, uses column-level annotations, but differentiates only between public and private columns. 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, but inside a protected Intel SGX enclave; its query rewriting rules focus on reducing the number of oblivious sorts required in distributed computation across multiple SGX machines.

Protected databases and scalability. The protected database community has produced decades of research on scaling secure query execution to the gigabyte-to-terabyte range [16, 46, 37]. This includes work on optimizations for boolean keyword search [87, 30], as well as large, general subsets of relational algebra [63]. 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 [54, 64], linear algebra [38], or matching [26].

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) [28] provably ensures that the output of an analysis reveals nothing about any individual input, but often uses a trusted curator to perform the analysis. Several prior systems have combined MPC and DP to avoid computing and output parties jointly reconstructing sensitive input data. DJoin [81] does so for SQL-style relational operations (with query rewriting, but without Conclave's automation and hybrid protocols), DStress [86] does so for graph analysis, and He et al. [48] do so for private record linkage. Conclave does not currently leverage DP in any way, but its query rewriting and code generation components are both sufficiently extensible to support addition of DP.

5.8 Conclusion

Conclave shows that secure MPC on "big data" is feasible by automatically rewriting queries to minimize expensive MPC work. Conclave can automatically run queries that would have either been impractical with previous MPC frameworks, or would have required substantial domain-specific knowledge to implement.

Conclave is open-source and available at:

`https://github.com/multiparty/conclave`.

6 Distributed RSA Key Generation

6.1 Introduction

RSA [92] is one of the oldest, publicly known, public key encryption schemes. This scheme allows a server to generate a public/private key pair, such that any client knowing the public key can use this to encrypt a message, which can only be decrypted using the private key. Thus a server can disclose its public key and keep the private key secret. This allows anyone to encrypt a message, which only the server itself can decrypt. Even though RSA has quite a few years on its back, it is still in wide use today such as in TLS, where it keeps web-browsing safe through HTTPS. Its technical backbone can also be used to realize digital signatures and as such is used in PGP.

In this work we consider the setting of distributed RSA key generation in the crucial two-party setting. Specifically this means that we consider two parties P_1 and P_2 whose goal is to generate an RSA modulus of a certain length, such that the knowledge of the private key is additively shared among them. Namely, the parties wish to compute the following:

Common input: A parameter ℓ describing the desired bits of the primes in an RSA modulus, and a public exponent e.

Common output: A modulus N of length 2ℓ bits.

Private outputs: P_1 learns outputs p_1, q_1, d_1, and P_2 learns outputs p_2, q_2, d_2, for which it holds that

- $(p_1 + p_2)$ and $(q_1 + q_2)$ are prime numbers of length ℓ bits.
- $N = (p_1 + p_2) \cdot (q_1 + q_2)$.
- $e \cdot (d_1 + d_2) = 1 \bmod \phi(N)$.
  (Namely, $(d_1 + d_2)$ is the RSA private key for (N, e).)

Furthermore, our ultimate goal is to have the functionality to work (or abort) even if one of the parties is not following the protocol. That is, in the *malicious* setting.

It turns out that all prior work follows a common structure for distributed RSA key generation. Basically, what is generally done is simply to pick random, odd numbers, and hope they are prime. However, the Prime Number Theorem tells us that many random prime candidates must be generated before success. Pairs of prime candidates must then be multiplied together to construct a modulus candidate. Depending on whether the tests of the prime candidates involve ensuring that a candidate is prime except with negligible probability, or only that it is somewhat likely to be prime, the modulus candidate must also be tested to ensure that it is the product of two primes. We briefly outline this general structure below:

Candidate Generation: The parties generate random additive shares of potential prime numbers.

Construct Modulus: Two candidates are multiplied together to construct a candidate modulus.

Verify Modulus: This involves ensuring that the public modulus is the product of two primes.

Construct Keys: Using the additive shares of the prime candidates, along with the modulus, the shared RSA key pair is generated.

With this overall structure in mind we consider the chronology of efficient distributed RSA key generation.

6.1.1 Related Work

Work on distributed RSA key generation started with the seminal result of Boneh and Franklin [15]. A key part of their result is an efficient algorithm for verifying biprimality of a modulus without knowledge of its factors. Unfortunately, their protocol is only secure in the semi-honest setting, against an honest majority. Several followup works handle both the malicious and/or dishonest majority setting [90, 34, 41, 2, 24, 47, 39]. First Frankel *et al.* [34] showed how to achieve malicious security against a dishonest minority. Poupard and Stern [90] strengthened this result to achieve security against a malicious majority (specifically the two-party setting) using 1-out-of-β OT, with some allowed leakage though. Later Gilboa [41] showed how to get semi-honest security in the dishonest majority (specifically two-party) setting. Both Algesheimer *et al.* [2] and Damgård and Mikkelsen [24] instead do a full primality test of the prime candidates individually, rather than a biprimality test of the modulus. Later Hazay *et al.* [47] introduced a practical protocol maliciously secure against a dishonest majority (in the two-party setting), which is leakage-free. More specifically their protocol is based on the homomorphic encryption approach from Gilboa's work [41], but adds zero-knowledge proofs on top of all the steps to ensure security against malicious parties. However, they conjectured that it would be sufficient to only prove correctness of a constructed modulus. This conjecture was confirmed correct by Gavin [39].

6.1.2 Contributions

In this work we present two new protocols for distributed RSA key generation. One for the semi-honest setting and one for the malicious setting. Neither of our protocols rely on any specific number theoretic assumptions, but instead are based on oblivious transfer (OT), which can be realized efficiently using an OT extension protocol [65, 85]. The malicious secure protocol also requires access to an IND-CPA encryption scheme, coin-tossing, zero-knowledge and secure two-party computation protocols. Using OT extension significantly reduces the amount of public key operations required by our protocols. This is also true for the maliciously secure protocol as secure two-party computation (and thus zero-knowledge) can be done black-box, based on OT.

We show that our maliciously secure protocol is more than an order of magnitude faster than its most efficient *semi-honest* competitor [47]. In particular, a four thread implementation takes on average 42 seconds to generate a maliciously secure 2048 bit key, whereas the protocol of Hazay *et al.* [47] on average required 15 minutes for a *semi-honestly* secure 2048 bit key.

More concretely this is done by introducing a new ideal functionality which gives the adversary slightly more (yet useless) power than normally allowed. This idea may be of independent interest as it is relevant for other schemes where many candidate values are constructed and potentially discarded throughout the protocol. We furthermore show how to eliminate much computation in the malicious setting by allowing a few bits of leakage on the honest party's prime shares. We carefully argue that this does not help an adversary in a non-negligible manner.

We also introduce a new and efficient approach to avoid selective failure attacks when using Gilboa's protocol [41] for multiplying two large integers together. We believe this approach may be of independent interest as well.

Finally we carefully construct the rest of out protocols in such a way that they take maximum advantage of oblivious transfer recent OT extension papers [65, 85] has all but reduced this to a symmetric primitive.

6.1.3 Applications

When considering RSA, and public key encryption in general, in the distributed setting it turns away from being simply a public key encryption scheme and into a primitive in itself used as a subcomponent in more advanced constructions. For example in the setting of distributed signature schemes [96], (homomorphic) threshold cryptosystems [47] and even general MPC [22].

There is also a case where distributed RSA key generation is an ends in itself, that is as a part of a cryptographic key management system in an enterprise setting without the use of a Hardware Security Module (HSM). HSMs are usually slow and expensive, and in general reflects a single point of failure. For this reason several companies, such as Unbound and Sepior have worked on realizing HSM functionality in a distributed manner, using MPC and secret-sharing. This removes the single point of failure, since computation and storage will be distributed between physically separated machines, running different operating systems and having different system administrators. Thus if one machine gets fully compromised by an adversary, the overall security of the generated keys will not be affected.

Moving further into the realm of big data and data-mining, another setting arrives where the usage of distributed public key generation really shines. This is a case of the client-server setting: Assume we have large amounts of data coming from many different clients, thus it is not desirable to have all the different input clients participate in an MPC execution. Ideally we would like these parties to be able to preprocess their own data and allow this to be obliviously given as input to an MPC computation carried out by a small number of servers. This is achievable if the clients can encrypt, and send to the servers, their preprocessed data using a public key encryption scheme; assuming servers have a secret sharing of the private key. Because the value held by the servers will be the same it makes it possible to avoid using MACs and simply use a protocol along the lines of Cramer *et al.* to do the MPC [22]. However, for popular schemes such as RSA or Paillier it is not easy to construct a public key with a secret shared private key, as this requires distributed random prime generation.

6.2 Preliminaries

We use κ to denote the computational security parameter and s the statistical security parameter. We use ℓ to denote the amount of bits in a prime factor of an RSA modulus. Thus $\ell \geq \kappa$. We use $[a]$ to denote the list of integers $1, 2, \ldots, a$. We will sometimes abuse notation and implicitly view bit strings as a non-negative integer.

Our functionality relies heavily on oblivious transfer. For our setting we require a random 1-out-of-β, meaning that one party, which we call the *sender* receives β random messages and the other party, which we call the *receiver*, gets to pick only one of the random messages to learn. The sender will not know which one the receiver picked and the receiver will not learn anything about the messages it did not pick. In some cases we need the sender to be able to specifically choose its messages. However, this is easily achieved by using the random OT-model as a black box and simply sending β extra strings repressing the shift between the random messages and the true message.

6.3 The Protocols

6.3.1 High level description

Keeping the general structure of distributed RSA key generation protocols of Section 6.1.1 in mind, we describe our maliciously secure protocol. Afterwards we will explain which steps we can do without if semi-honest security is all that is desired.

We start with a **Setup** phase. In this phase each party will *commit* to a random AES key K of its choosing, by sending $c = \mathsf{AES}_{\mathsf{AES}_r(K)}(0)$ for a random r (chosen by coin-tossing). This unusual *double encryption* ensures that c is not only *hiding* K, through the encryption, but also *binding* to K. The key K is then used to implement a committing functionality. This is done by using K as the key in an AES encryption, where the value we want to commit to is the message encrypted. However, for our proof to go through we require this commitment to be extractable. Fortunately this is easily achievable if the simulator knows K and to ensure this we do a zero-knowledge argument of knowledge that $c = \mathsf{AES}_{\mathsf{AES}_r(K)}(0)$. By executing this zero-knowledge argument the simulator can clearly extract K (assuming the zero-knowledge argument is an ideal functionality). This can be done highly efficiently using garbled circuits [59].

Afterwards the **Candidate Generation** is executed where the two parties choose random shares p_1 and p_2, respectively, with the hope that $p_1 + p_2$ is prime. They commit to these values by computing an AES encryption under their individual keys, picked in the setup phase, and exchanging the ciphertexts. The parties then run a secure protocol, based on 1-out-of-β OT, which rules out the possibility that $p_1 + p_2$ is divisible by any prime number smaller than some pre-agreed threshold B_1 [90]. We call this the first *trial division*. If $p_1 + p_2$ is not divisible by any such prime then it passed on to the next stage, otherwise it is discarded.

On shares p_1, p_2 passing this check parties execute the **Construct Modulus** phase by executing a secure protocol inspired by Gilboa [41]. This protocol is based on 1-out-of-2 OT and computes the candidate modulus $N = (p_1 + p_2)(q_1 + q_2)$ and returns this value to both parties. Gilboa's protocol allows two parties to compute an additive sharing of a product where each party holds a factor. For example $p_1 q_2 = a_1 + a_2$ where P_1 is the receiver and gets a_1 and P_2 is the sender and gets a_2. The idea of Gilboa's protocol is to execute a 1-out-of-2 OT for each bit of p_1. For each OT P_1 would input a random value for choice 0 and the same random value for choice 1, but added to q_2. The receiving party would input a bit of p_1 for each of the OTs. Based on the messages the receiver learns, and the randomness the sender picks, it is possible for the parties to compute an additive sharing of the product. With that at hand it is possible to compute an additive sharing of N by noticing that $N = (p_1 + p_2)(q_1 + q_2) = p_1 q_1 + p_2 q_2 + p_1 q_2 + p_2 q_1$ and thus having P_1's share be $p_1 q_1$ added with its share of $p_1 q_2$ and $p_2 q_1$, computing using Gilboa's protocol. Similarly for P_2 its share is $p_2 q_2$ added with its shares of $p_1 q_2$ and $p_2 q_1$, computing using Gilboa's protocol. Unfortunately Gilboa's protocol is only secure against semi-honest adversaries, even if a maliciously secure OT is used. The reason is, as is often the case when OT is involved, selective failure vulnerabilities [68, 78]. Specifically, what a malicious sender does is to guess that the receiver's choice bit is 0 (or 1) in a given OT. In this case, the sender inputs the correct message for choice 0, i.e. the random string. But for the message for a choice of 1, it inputs the 0-string. If the sender's guess was correct, then the protocol executes correctly. However, if its guess was wrong, then the result of the Gilboa protocol, i.e. the modulus, will be incorrect. If this happens then the protocol will abort during the proof of honesty. Thus, two distinct and observable things happen dependent on whether the sender's guess was correct or not and so the sender learns a bit of the receiver's input by observing what happens. In fact, the sender can repeat this as many times as it wants, each time succeeding with probability 1/2 (when the receiver's input is randomly sampled). This means that with probability 2^{-x} it can learn x of the receiver's secret input bits.

To prevent this attack we use the notion of *noisy encodings*. A noisy encoding is basically a linear encoding with some noise added such that decoding is *only* possible when using some auxiliary information related to the noise. We have the receiver noisily encode its true input to the Gilboa protocol. Because of the linearity it is possible to retrieve the true output in the last step of the Gilboa protocol (where the parties send their shares to each other in order to learn the result N) without leaking anything on the secret shares of the receiver, even in the presence of a selective failure attack.

In a bit more detail, we define a 2^{-s}-statistically hiding noisy encoding of a value $a \in \mathbb{Z}_{2^{\ell-1}}$ as follows:

- Let $\mathscr{P}$ be the smallest prime larger than $2^{2\ell}$.
- Pick random values $h_1, \ldots, h_{2\ell+3s}, g \in \mathbb{F}_{\mathscr{P}}$ and random bits $d_1, \ldots, d_{2\ell+3s}$ under the constraint that $g + \sum_{i \in [2\ell+3s]} h_i \cdot d_i \mod \mathscr{P} = a$.
- The noisy encoding is then $(h_1, \ldots, h_{2\ell+3s}, g)$ and the decoding info is $(d_1, \ldots, d_{2\ell+3s})$.

Assume without loss of generality that P_1 is acting as the receiver, then for each of its shares, p_1 and q_1, P_1 noisily encodes as described and sends the noisy encodings $(h_{p,1}, \ldots, h_{p,2\ell+3s}, g_p)$ and $(h_{q,1}, \ldots, h_{q,2\ell+3s}, g_q)$ to P_2. Next, when they execute the OT steps, P_1 uses the decoding info $(d_{p,1}, \ldots, d_{p,2\ell+3s})$ and $(d_{q,1}, \ldots, d_{q,2\ell+3s})$ of p_1 and q_1 respectively and uses this as input the OTs instead of the bits of p_1 and q_1. For each such bit of p_1, P_2 proceeds like in the regular Gilboa protocol and inputs to the OT a random value for choice 0 and the same random value plus q_2 for choice 1. It turns out that leaking at most s bits of $(d_{p,1}, \ldots, d_{p,2\ell+3s})$ and $(d_{q,1}, \ldots, d_{q,2\ell+3s})$ to P_2 does not give more than a 2^{-s} advantage in finding the value encoded. Thus, even if P_2 launches s selective failure attacks it gains no significant knowledge on P_1's shares.

After having completed the OTs, the parties compute their shares of the modulus N by using the linearity of the encodings.

The parties then execute the **Verify Modulus** phase by first doing a second *trial division*. This is done by locally checking that no primes smaller than a threshold B_2 ($B_1 < B_2$) are a factors of N. If N is divisible by such a number then N is definitely not a valid RSA modulus and is discarded. For an N not discarded, the parties run a secure **Biprimality Test** which verifies that N is the product of two primes. If it is not, it is discarded. The biprimality test is almost the same as the one presented by Boneh and Franklin [15].

For the first candidate passing these tests a **Proof of Honesty** is executed. This phase has three responsibilities: first, it is a maliciously secure execution of the full biprimality test of Boneh and Franklin [15]; second, it verifies that the modulus is constructed from the values committed to in the candidate generation phase. Finally it generates a random sharing of the private key. The proof of honesty is carried out twice. Once where party P_1 acts as the prover and P_2 the verifier, and once where P_2 acts the prover and P_1 the verifier. Thus each party gets convinced of the honesty of the other party and learns their respective shares of the private key.

To ensure a correctly executed biprimality test, a typical zero-knowledge technique is used, where coin-tossing is used to sample public randomness and the prover randomizes its witness along with the statement to prove. The verifier then gets the option to decide whether he wants to learn the value used for randomizing or the randomized witness. This ensures that the prover can only succeed with probability 1/2 in convincing the verifier if it does not know a witness.

To ensure that the modulus was constructed from the values committed to, a small secure two-party computation is executed which basically verifies that this is the case. Since the commitments are AES-based, this can be carried out in a very lightweight manner. Furthermore, to ensure that the values used in the maliciously secure biprimality test are also consistent with the shares committed to, we have the prover commit to the randomization values as well and verify these, along with their relation to the shares.

Finally, we let the proving party input some randomness which is used to randomize the verifying party's share of the private key.

The ideal functionality. The ideal functionality that our protocol implements works by sampling random primes congruent to 3 modulo 4 and secret key shares which are random and in the range between $-2^{2\ell+s}$ and $2^{2\ell+s}$. When a party is corrupted the adversary is allowed certain freedoms in its interaction with the ideal functionality. Specifically the adversary is given access to several commands, allowing it and the functionality to generate a shared RSA key through an interactive game. In particular the adversary is allowed to repeatedly query the functionality to sample primes, based on its choice of a prime share. The adversary can then let the ideal functionality use these primes to construct a modulus which it learns, along with its share of the secret key for this specific modulus. Finally, the adversary can then decide which modulus it wishes to use.

We furthermore note that the functionality allows the adversary a few bits of leakage of the honest party's prime share. To see that this is not a problem assume that learning some extra bits on the honest party's prime shares would give the adversary a non-negligible advantage in finding the primes of the modulus. This would then mean that there exists a polynomial time algorithm breaking the security of RSA with non-negligible probability by simply exhaustively guessing what the leaked bits are and then running the adversary algorithm on each of the guesses. Thus if the amount of leaked bits is at most polylog in κ, then such an algorithm would also be polynomial time, and cannot exist if RSA is secure.

6.3.2 Concrete Protocol

Below we describe the concrete protocol, which realizes this functionality, in more details:

Setup. The parties use a coin-tossing functionality to pick random values $r_1, r_2 \in \{0,1\}^\kappa$. Both parties pick a uniformly random key; K_1 for party P_1 and K_2 for P_2. For $\iota \in \{1,2\}$ party P_ι computes and sends $\mathsf{AES}_{\mathsf{AES}_{r_\iota}(K_\iota)}(0) = c_\iota$ to $P_{3-\iota}$. Each party then proves towards the other that it knows K_ι such that $\mathsf{AES}_{\mathsf{AES}_{r_\iota}(K_\iota)}(0) = c_\iota$.

Candidate Generation. The parties P_1 and P_2 choose private random strings p_1 and p_2, respectively, of length $\ell - 1$ bits, subject to the constraint that the two least significant bits of p_1 are 11, and the two least significant bits of p_2 are 0 (this ensures that the sum of the two shares is equal to 3 modulo 4). For $\iota \in \{1,2\}$ party P_ι sends $\mathsf{AES}_{K_\iota}(p_\iota) = H_{p_\iota}$ to $P_{3-\iota}$. The parties now check, for each prime number $3 \leq \beta \leq B_1$, that $(p_1 + p_2) \neq 0 \bmod \beta$. We describe this in procedure $\mathsf{Div\text{-}OT}$ in Protocol 8. The parties run this test for each prime $3 \leq \beta \leq B_1$ in increasing order (where B_1 is the pre-agreed threshold).

Construct Modulus. Once two numbers pass the previous test, the parties have shares of two candidate primes p_1, p_2 and q_1, q_2. They compute the candidate modulus

$$N = (p_1 + p_2)(q_1 + q_2) = p_1q_1 + p_2q_2 + p_1q_2 + p_2q_1.$$

Let $\mathscr{P}$ be the smallest prime larger than $2^{2\ell}$. For each $\alpha \in \{p,q\}$ party P_1 picks a list of values $h_{\alpha,1}, \ldots, h_{\alpha,2\ell+3s}, g_\alpha \in \mathbb{Z}_\mathscr{P}$ and a list of bits $d_{\alpha,1}, \ldots, d_{\alpha,2\ell+3s} \in \{0,1\}$ uniformly at random under the constraint that $g_\alpha + \sum_{i\in[2\ell+3s]} h_{\alpha,i} \cdot d_{\alpha,i} \mod \mathscr{P} = \alpha_1$. The parties execute the following steps for each $\alpha \in \{p,q\}$ and $i \in [2\ell+3s]$:

1. P_2 chooses a uniformly random value $r_{\alpha,i} \in \mathbb{Z}_\mathscr{P}$ and sets

$$c_{0,\alpha,i} = r_{\alpha,i} \quad \text{and} \quad c_{1,\alpha,i} = \begin{cases} r_{\alpha,i} + q_2 \mod \mathscr{P} & \text{if } \alpha = p \\ r_{\alpha,i} + p_2 \mod \mathscr{P} & \text{if } \alpha = q \end{cases}.$$

Protocol 8 Div-OT, 1-out-of-β OT based trial division procedure

The parties have common input $\beta \in \mathbb{N}$ and P_1 has $p_1 \in \mathbb{N}$ and P_2 has $p_2 \in \mathbb{N}$. The procedure returns $\perp$ iff $\beta | (p_1 + p_2)$, otherwise it returns $\top$.

1. P_2 uses the 1-out-of-β OT as the sender to learn random messages $\{m_i\}_{i \in [\beta]}$.
2. P_1 computes $a_1 = p_1 \mod \beta$ and gives a_1 as input to the 1-out-of-β OT as the receiver and thus learns m_{a_1}.
3. P_2 lets $a_2 = -p_2 \mod \beta$ and sends m_{a_2} to P_1.
4. P_1 checks whether $m_{a_1} = m_{a_2}$ and outputs $\perp$ and sends it to P_2 if this is the case, otherwise it outputs $\top$ and sends this to P_2.

2. P_2 inputs $c_{0,\alpha,i}, c_{1,\alpha,i}$ as the sender into a 1-out-of-2 OT.
3. P_1 inputs $d_{\alpha,i}$ as its choice-bit to the OT thus receives the message $c_{d_{\alpha,i},i}$.

P_1 sends the values $h_{\alpha,1}, \ldots, h_{\alpha,2\ell+3s}, g_\alpha$ to P_2 for $\alpha \in \{p,q\}$. P_1 then computes $z_1^\alpha = \sum_{i \in [2\ell+3s]} c_{d_{\alpha,i},i} \cdot h_{\alpha,i} \mod \mathscr{P}$ and P_2 computes $z_2^\alpha = -\sum_{i \in [2\ell+3s]} r_{\alpha,i} \cdot h_{\alpha,i} \mod \mathscr{P}$. Afterwards P_2 computes $a_2 = p_2 q_2 + z_2^p + g_p \cdot q_2 + z_2^q + g_q \cdot p_2 \mod \mathscr{P}$ and sends this to P_1. Now P_1 computes $a_1 = p_1 q_1 + z_1^p + z_1^q \mod \mathscr{P}$ and sends this to P_2. Both parties then compute $N = (a_1 + a_2 \mod \mathscr{P}) \mod 2^{2\ell}$.

Some auxiliary info needs to be computed in order to construct shared keys for this modulus and to ensure that $\gcd(e, \Phi(N)) = 1$. This consists of P_1 computing $w_1 = N + 1 - p_1 - q_1 \mod e$ and similarly P_2 computes $w_2 = p_2 + q_2 \mod e$. The parties then execute a random 1-out-of-β OT where P_2 acts as the sender and thus learns the random messages $r_0, \ldots, r_{\beta-1} \in \{0,1\}^\kappa$. P_1 acts as the receiver and inputs w_1 and thus learns r_{w_1}. P_2 then sends r_{w_2} to P_1. If $r_{w_1} = r_{w_2}$ then P_1 informs P_2 of this and they both discard the candidate N and its associated shares p_1, q_1, p_2, q_2.

Verify Modulus. The parties start with the second *trial division* which consists of one party trying to divide N by all primes numbers in the range $B_1 < \beta \leq B_2$. If one is a factor then the parties discard the candidate. The parties then carry out the *biprimality test* on N of Protocol 9, which is inspired by [15].

Proof of Honesty. The parties proceed as follows, once where P_1 takes the role of a *prover* and P_2 takes the role of a *verifier*, and once where P_2 takes the role of a *prover* and P_1 takes the role of a *verifier*. We use subscript P to denote the number of the proving party, similarly we use subscript V to denote the number of the verifying party e.g. when P_1 is the prover $p_P := p_1$, $q_P = q_1, K_P := K_1$ and when P_2 the verifier then $p_V := p_2$, $q_V = q_2, K_V := K_2$:

1. The parties execute step 1 of the biprimality test of Protocol 9 again, but this time using a coin-tossing functionality to sample the values γ (we denote the γ used in the i'th iteration by γ_i and $\gamma_{i,P}$ to denote the value γ_P sent in the i'th iteration, for $i \in [s]$).
2. For $j \in [s]$ the *prover* picks a random value $t_j \in \{0,1\}^{\ell-2+s}$. It then computes $\mathsf{AES}_{K_P}(t_j) = H_{t_j}$ and sends this to the verifier.
3. For each $i, j \in [s]$, the *prover* then sends the values $\bar{\gamma}_{i,j} = \gamma_i^{t_j} \mod N$ to the *verifier*.

Protocol 9 Biprimality Test

The parties have common input $N \in \mathbb{N}$. The procedure returns $\top$ if N is a biprime, otherwise it returns $\perp$ with overwhelming probability.

1. The parties execute following test s times.
   (a) P_1 samples a random value $\gamma \in \mathbb{Z}_N^\times$ with Jacobi symbol 1 over N.
   (b) P_1 sends γ to P_2.
   (c) P_1 computes $\gamma_1 = \gamma^{N+1-p_1-q_1} \mod N$ and sends this value to P_2.
   (d) P_2 checks if $\gamma_1 \cdot \gamma^{-p_2-q_2} \mod N \neq \pm 1$. In this case P_2 sends $\perp$ to P_1 and the parties break the loop and discard the candidate N.
2. The parties verify that $\gcd(N, p+q-1) = 1$.
   (a) P_1 chooses a random number $\bar{r}_1 \in \mathbb{Z}_{2^{\ell+2s}}$ and P_2 chooses a random $\bar{r}_2 \in \mathbb{Z}_{2^{\ell+2s}}$. P_1 computes $\mathsf{AES}_{K_1}(\bar{r}_1) = H_{\bar{r}_1}$ and sends this to P_2 and P_2 computes $\mathsf{AES}_{K_2}(\bar{r}_2) = H_{\bar{r}_2}$ and sends this to P_1. (The parties will verify that $(\bar{r}_1 + \bar{r}_2)(p+q-1) \mod 2^{2\ell+2s+2} = 1$.)
   (b) The parties run a multiplication protocol (similar to that run in the "Construct Modulus" phase) where they compute shares α_1, α_2 (known to P_1, P_2 respectively) of $\bar{r}_1 \cdot (p_2 + q_2 - 1) \mod 2^{2\ell+2s+2}$, and shares β_1, β_2 of $\bar{r}_2 \cdot (p_1 + q_1) \mod 2^{2\ell+2s+2}$.
   (c) P_1 sends to P_2 the value $s_1 = \bar{r}_1(p_1 + q_1) + \alpha_1 + \beta_1 \mod 2^{2\ell+2s+2}$.
   (d) P_2 computes $s_2 = \bar{r}_2(p_2 + q_2 - 1) + \alpha_2 + \beta_2 \mod 2^{2\ell+2s+2}$, and verifies that $\gcd(s_1 + s_2 \mod 2^{2\ell+s+2}, N) = 1$. If this is not the case then it sends $\perp$ to P_1 and discard the candidate N.

4. The parties then use a coin-tossing functionality to sample uniformly random bits $b_1, \dots, b_s \in \{0,1\}$. For each $j \in [s]$, the *prover* then sends $v_j = b_j \cdot (-p_P - q_P) + t_j$ to the *verifier*.
5. For each $i, j \in [s]$ the *verifier* checks that

$$\gamma_i^{v_j} \mod N =^? \bar{\gamma}_{i,j} \cdot \gamma_{i,P}^{b_j} \cdot \gamma_i^{-b_j \cdot (N+1)} \mod N$$

   If this is not the case then the parties abort.
6. The prover picks a uniformly random value $\rho_P \in \{0,1\}^{2\ell+s}$ and both parties then execute a secure two-party computation which takes as common input the values sent between the two parties throughout the protocol for this specific modulus N. The private input is $(p_P, q_P, K_P, \bar{r}_P, \rho_P)$

and $(p_V, q_V, K_V, \bar{r}_V)$. The computation is as follows:

$$
\begin{aligned}
& w := N+1-(p_P+q_P+p_V+q_V) \mod e\ , \\
& \chi := (H_{p_P} \overset{?}{=} \mathsf{AES}_{K_P}(p_P)) \wedge (H_{q_P} \overset{?}{=} \mathsf{AES}_{K_P}(q_P)) \quad \wedge \quad (c_P \overset{?}{=} \mathsf{AES}_{\mathsf{AES}_{r_P}(K_1)}(0)) \\
& \qquad \wedge (H_{p_V} \overset{?}{=} \mathsf{AES}_{K_V}(p_V)) \wedge (H_{q_V} \overset{?}{=} \mathsf{AES}_{K_V}(q_V)) \quad \wedge \quad (c_V \overset{?}{=} \mathsf{AES}_{\mathsf{AES}_{r_V}(K_V)}(0)) \\
& \qquad \wedge (\forall j \in [s] : \mathsf{AES}_{K_P}(v_j + b_j \cdot (p_P+q_P)) \overset{?}{=} H_{t_j}) \\
& \qquad \wedge (N \overset{?}{=} (p_P+p_V)\cdot(q_P+q_V)) \\
& \qquad \wedge w \neq 0 \\
& \quad \text{if } P_1 \text{ is verifier} : \tilde{d}_V := \left\lfloor \frac{-(w^{-1} \mod e)\cdot(N+1-(p_1+q_1))+1}{e} \right\rfloor \\
& \quad \text{else} : \qquad \tilde{d}_V := \left\lfloor \frac{-(w^{-1} \mod e)\cdot(-p_2-q_2)}{e} \right\rfloor \\
& \text{if } \chi = 1 : \bar{d}_V := \tilde{d}_V - \rho_P, \psi := \gcd(N, p_P+p_V+q_P+q_V-1) \overset{?}{=} 1 \\
& \quad \text{else } : \bar{d}_V = \perp, \psi = \perp \\
& \qquad \text{Output } (\chi, \psi, \bar{d}_V) \text{ to the verifier and } (\perp) \text{ to the prover}
\end{aligned}
$$

The parties abort if $\chi = 0$.

Construct Keys. P_1 computes and outputs $d_1 = \bar{d}_1 + \rho_1$. If $e | -p_2 - q_2$ then P_2 computes and outputs $d_2 = \bar{d}_2 + \rho_2$, else it computes and outputs $d_2 = \bar{d}_2 + \rho_2 + 1$.

6.3.3 Semi-honest Protocol

We note that if only semi-honest security is desired it is possible to execute almost the same protocol, except for the following differences:

- The underlying OT functionality only needs to be semi-honestly secure.
- All the AES encryptions can be removed.
- The *setup* and *proof of honest* phases can be skipped.
- The noisy encoding used in the multiplication is not needed as there is no need to prevent selective failures in a semi-honest execution.

6.4 Efficiency

We here try to compare the efficiency of our protocol with previous work. This is done in Table 5.

With regards to more concrete efficiency we recall that both our malicious and semi-honest protocols, along with previous work have the same type of phases, working on randomly sampled candidates in a pipelined manner. Because of this feature, all protocols limit the amount of unsuitable candidates passing through to the expensive phases, by employing trial division. This leads to fewer executions of expensive phases and thus to greater concrete efficiency. In some protocols this filtering is applied both to individual prime candidates *and* to candidate moduli, leading to minimal executions of the expensive phases. Unfortunately this is not possible in all protocols. For this reason we

Scheme	Assumptions	Dishonest majority	Malicious secure	Prime trial division	Modulus trial division	Rounds	Amount of candidates	Candidate generation	Construct modulus	(Bi)primality test	Leakage
Our result*	IND-CPA, OT, CT	✓	✓	✓	✓	**$O(1)$**	$O(\ell^2/\log^2(\ell))$	**$O(\ell)$**	**$O(\ell^2)$**	$O(s \cdot \ell^3)$	$\tau+2$
[15]	**None**	✗	✗	✓	✓	**$O(1)$**	$O(\ell^2/\log^2(\ell))$	**$O(\ell)$**	**$O(\ell^2)$**	$O(s \cdot \ell^3)$	**2**
[34]	DL	✗	✓	✗	✓	**$O(1)$**	$O(\ell^2/\log^2(\ell))$	$O(\ell^3)$	$O(\ell^3)$	$O(s^2 \cdot \ell^3)$	**2**
[90]†	OT	✓	✓	✓	✓	**$O(1)$**	$O(\ell^2/\log^2(\ell))$	**$O(\ell)$**	**$O(\ell^2)$**	?	$\tau+2$
[41]	PRG, OT	✓	✗	✗	✓	**$O(1)$**	$O(\ell^2/\log^2(\ell))$	**$O(\ell)$**	**$O(\ell^2)$**	$O(s \cdot \ell^3)$	**2**
[2]	**None**	✗	✗	✓	✗	$O(\ell)$	**$O(\ell/\log(\ell))$**	**$O(\ell)$**	**$O(\ell^2)$**	$O(s \cdot \ell^3)$	**2**
[24]	CRS, Strong RSA	✗	✓	✓	✓	**$O(1)$**	$O(\ell^2/\log^2(\ell))$	$O(\ell^3)$	$O(\ell^3)$	**$o(s \cdot \ell^3)$**	**2**
[47]	DCR, DDH	✓	✓	✓	✓	$O(1)$‡	$O(\ell^2/\log^2 \ell)$	$O(\ell^3)$	$O(\ell^3)$	$O(s \cdot \ell^3)$	**2**

Table 5: Comparison of the different protocols for distributed RSA key generation. The best possible values are highlighted in bold. All values assume a constant, and minimal, amount of participating parties; i.e. 2 or 3. The column *Amount of candidates* expresses the expected amount of random candidates that must be generated before finding a suitable modulus. The column *Candidate generation* expresses the computational bit complexity required to construct a *single* candidate prime. The column *Construct modulus* expresses the computational bit complexity required to construct a *single* potential modulus, based on two prime candidates. The column *(Bi)primality test* expresses the computational bit complexity required to verify that a *single* prime candidate is prime except with negligible probability or (depending on the protocol) to verify that a *single* modulus is the product of two primes except with negligible probability. The column *Leakage* expresses how many bits of information of the honest party's shares of the primes that is leaked to the adversary. Here τ means that $\sum_{\beta \in B_1} \log\left(\frac{\beta}{\beta-1}\right)$ bits can be leaked to a malicious adversary. Furthermore, the adversary is allowed to pick a probability x with which it learns $(1+\varepsilon)x$ extra bits. However, if the adversary does not learn the extra bits then the honest party learns that the adversary has acted maliciously.

*: For the malicious protocol $O(s^2 \cdot \ell^3)$ operations are executed *once* per successful key pair generation.

†: The authors do not describe how to ensure biprimality in case of a malicious adversary.

‡: Constant round on average.

also show in Table 5 which protocols manage to improve the expected execution time by doing trial division of the prime candidates, respectively the moduli.

We see that the only real competition lies in the work of Poupard and Stern [90]. However, we note that they don't provide a full maliciously secure protocol. In particular they do not describe how to do a biprimality test secure against a malicious and dishonest majority. Thus the only other protocol considering the same setting as us is the one by Hazay *et al.*. This is also the newest of the schemes and is considered the current state-of-the-art in this setting. However, this protocol requires asymptotically more operations for candidate generation, construction of modulus and the biprimality test.

6.4.1 Implementation.

Below we outline the concrete implementation choices we made. We implement AES in counter mode, using AES-NI, with $\kappa = 128$ bit keys. For 1-out-of-2 OT (needed during the *Construct Modulus* phase) we use the maliciously secure OT extension of Keller *et al.* [65]. For the base OTs we use the protocol of Peikert *et al.* [89] and for the internal PRG we use AES-NI with the seed as key, in counter mode. For the random 1-out-of-β OT we use protocol of Naor and Pinkas [80]. For the coin-tossing we use the standard "commit to randomness and then open" approach.

We did not yet implement the zero-knowledge argument or the two-party computation since they can be efficiently realized using existing implementations of garbled circuits (such as JustGarble [8] or TinyGarble [97]) by using the protocol of Jawurek *et al.* [59] for zero-knowledge and the dual-execution approach [78] for the two-party computation. These protocols are only executed *once* in our scheme and thus, as is described later in this section, we can safely estimate that the effect on the total run time is marginal.

6.4.2 Experiments.

We implemented our maliciously secure protocol and ran experiments on Azure, using Intel Xeon E5-2673 v.4 - 2.3Ghz machines with 64Gb RAM, connected by a 40.0 Gbps network.

We used the code to run 50 computations of a shared 2048 bit modulus, and computed the average run time. The results are as follows:

- With a single threaded execution, the average run time was 134 seconds.
- With four threads, the average run time was 39.1 seconds.
- With eight threads, the average run time was 35 seconds.

The run times showed a high variance (similar to the results of the implementation reported by Hazay *et al.* [47] for their protocol). For the single thread execution, the average run time was 134 seconds while the median run time was 84.9 seconds (the fastest execution took 8.2 sec and the slowest execution took 542 sec).

Focusing on the single thread execution, we measured the time consumed by different major parts of the protocol. The preparation of the OT extension tables took on average 12% of the run time, the multiplication protocol computing N took 66%, and the biprimality test took 7%. (These percentages were quite stable across all executions and showed little variance.) Overall these parts took 85% of the total run time. The bulk of the time was consumed by the secure multiplication protocol. In that protocol, most time was spent on computing the values z_1^α, z_2^α (as part of the **Construct Modulus** phase). This is not surprising since each of these computations computes $2\ell + 3s = 2168$ bignum multiplications.

Note that these numbers exclude the time required to do the zero-knowledge argument of knowledge in *Setup* and the two-party computation in *Proof of Honesty*. The zero-knowledge argument of knowledge requires about 12,000 AND gates (for two AES computations), and our analysis shows that the number of AND gates that need to be evaluated in the circuits of the honesty proof is at most 22 million. We also measured a throughput of computing about 3.2 million AND gates in Yao's protocol on the machines that we were using. Therefore we estimate that implementing these computations using garbled circuits will contribute about 7 seconds to the total time.

Comparing to previous work, the only other competitive protocol (for 2048 bit keys) with implementation work is the one by Hazay *et al.* [47]. Unfortunately their implementation is not publicly available and thus we are not able to make a comparison on the same hardware. However, we do not that the fastest time they report is 15 min on a 2.3 GHz dual-core Intel desktop, for their *semi-honestly secure* protocol.

References

[1] M.J. Atallah, F. Kerschbaum, W. Du. *Secure and private sequence comparisons*. Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society, WPES '03, ACM, New York, NY, USA, pp. 39–44, 2003.

[2] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 417–432. Springer, 2002.

[3] N. C. Ankeny. The least quadratic non residue. *Annals of Mathematics*, 55(1):65–72, 1952.

[4] S. Asadova. Privacy-Preserving DNA Sequence Alignment. Master's thesis, Eindhoven University of Technology, 2017.

[5] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In Piotr Rudnicki, editor, *8th ACM Symposium Annual on Principles of Distributed Computing*, pages 201–209, Edmonton, Alberta, Canada, August 14–16, 1989. Association for Computing Machinery.

[6] Erwin H. Bareiss. Sylvester's identity and multistep integer-preserving gaussian elimination. *Mathematics of Computation*, 22(103):565–578, 1968.

[7] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel Kho, and Jennie Rogers. Smcql: Secure querying for federated databases. *Proceedings of the VLDB Endowment*, 10(6):673–684, February 2017.

[8] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society, 2013.

[9] Azer Bestavros, Andrei Lapets, and Mayank Varia. User-centric distributed solutions for privacy-preserving analytics. *Commun. ACM*, 60(2):37–39, 2017.

[10] Dimitrios Bisias, Mark Flood, Andrew W. Lo, and Stavros Valavanis. A survey of systemic risk analytics. *Annual Review of Financial Economics*, 4(1):255–296, 2012.

[11] 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 Rainer Böhme and Tatsuaki Okamoto, editors, *FC 2015: 19th International Conference on Financial Cryptography and Data Security*, volume 8975 of *Lecture Notes in Computer Science*, pages 227–234, San Juan, Puerto Rico, January 26–30, 2015. Springer, Heidelberg, Germany.

[12] Dan Bogdanov, Liina Kamm, Baldur Kubo, Reimo Rebane, Ville Sokk, and Riivo Talviste. Students and taxes: a privacy-preserving social study using secure computation. Cryptology ePrint Archive, Report 2015/1159, 2015. http://eprint.iacr.org/2015/1159.

[13] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: a framework for fast privacy-preserving computations. Cryptology ePrint Archive, Report 2008/289, 2008. http://eprint.iacr.org/2008/289.

[14] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *FC 2009: 13th International Conference on Financial Cryptography and Data Security*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343, Accra Beach, Barbados, February 23–26, 2009. Springer, Heidelberg, Germany.

[15] Dan Boneh and Matthew K. Franklin. Efficient generation of shared RSA keys. *J. ACM*, 48(4):702–722, 2001.

[16] Christoph Bösch, Pieter H. Hartel, Willem Jonker, and Andreas Peter. A survey of provably secure searchable encryption. *ACM Comput. Surv.*, 47(2):18:1–18:51, 2014.

[17] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital Systems Research Center, Palo Alto, CA, 1994.

[18] H. Campmans. Optimizing convolutional neural networks in multiparty computation. Master's thesis, Dept of Mathematics and Computer Science, TU Eindhoven, The Netherlands, 2018.

[19] 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. *Proceedings of the VLDB Endowment*, 1(2):1265–1276, August 2008.

[20] Li Chen, Wayne Eberly, Erich Kaltofen, B. David Saunders, William J. Turner, and Gilles Villard. Efficient matrix preconditioners for black box linear algebra. *Linear Algebra and its Applications*, 343-344(Supplement C):119 – 146, 2002. Special Issue on Structured and Infinite Systems of Linear equations.

[21] Ronald Cramer and Ivan Damgård. Secure distributed linear algebra in a constant number of rounds. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 119–136, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[22] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–299. Springer, 2001.

[23] Ronald Cramer, Eike Kiltz, and Carles Padró. A note on secure computation of the Moore–Penrose pseudoinverse and its application to secure linear algebra. In *Proc. CRYPTO 2007, Santa Barbara, USA*, pages 613–630. Springer, 2007.

[24] Ivan Damgård and Gert Læssøe Mikkelsen. Efficient, robust and constant-round distributed RSA key generation. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2010.

[25] John D. Dixon. Exact solution of linear equations using p-adic expansions. *Numer. Math.*, 40(1):137–141, February 1982.

[26] Jack Doerner, David Evans, and Abhi Shelat. Secure stable matching at scale. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16: 23rd Conference on Computer and Communications Security*, pages 1602–1613, Vienna, Austria, October 24–28, 2016. ACM Press.

[27] Dugan, Tamara, and Xukai Zou. *A Survey of Secure Multiparty Computation Protocols for Privacy Preserving Genetic Tests* . Connected Health: Applications, Systems and Engineering Technologies (CHASE), 2016 IEEE First International Conference on, pp. 173-182., 2016.

[28] Cynthia Dwork. Differential privacy: A survey of results. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation*, TAMC'08, pages 1–19, Berlin, Heidelberg, 2008. Springer-Verlag.

[29] F. Kerschbaum, M. Beck, D. Schönfeld. *Inference control for privacy-preserving genome matching*. CoRR abs/1405.0205, 2014.

[30] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel Rosu, and Michael Steiner. Rich queries on encrypted data: Beyond exact matches. Cryptology ePrint Archive, Report 2015/927, 2015. http://eprint.iacr.org/2015/927.

[31] Uri Feige, Joe Killian, and Moni Naor. A minimal model for secure computation (extended abstract). In *Proc. STOC '94*, pages 554–563, 1994.

[32] Joan Feigenbaum and Lance Fortnow. Random-self-reducibility of complete sets. *SIAM Journal on Computing*, 22(5):994–1005, 1993.

[33] Pierre-Alain Fouque, Jacques Stern, and Geert-Jan Wackers. Cryptocomputing with rationals. In Matt Blaze, editor, *Financial Cryptography 2002, Southampton, Bermuda*, pages 136–146. Springer, 2003.

[34] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed rsa-key generation. In J. Vitter, editor, *STOC*, pages 663–672. ACM, 1998.

[35] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 331–361. Springer, 2018.

[36] V. R. Fridlender. On the least n-th power non-residue. *Dokl. Akad. Nauk. SSSR*, 66:351—352, 1949.

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

[38] 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*, pages 345–364, 10 2017.

[39] Gérald Gavin. RSA modulus generation in the two-party case. *IACR Cryptology ePrint Archive*, 2012:336, 2012.

[40] Irene Giacomelli, Somesh Jha, Marc Joye, C. David Page, and Kyonghwan Yoon. Privacy-preserving ridge regression over distributed data from LHE. Cryptology ePrint Archive, Report 2017/979, 2017.

[41] Niv Gilboa. Two party RSA key generation. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 116–129. Springer, 1999.

[42] 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)*, April 2015.

[43] Michael T. Goodrich. The mastermind attack on genomic data. In *2009 IEEE Symposium on Security and Privacy*, pages 204–218, Oakland, CA, USA, May 17–20, 2009. IEEE Computer Society Press.

[44] S. W. Graham and C. J. Ringrose. Lower bounds for least quadratic non-residues. In B. C. Berndt et al., editors, *Analytic Number Theory: Proc. of a Conf. in Honor of Paul T. Bateman*, pages 269–309, Boston, MA, 1990.

[45] Joseph Halpern and Vanessa Teague. Rational secret sharing and multiparty computation: Extended abstract. In *Proc. STOC 2004*, STOC ’04, pages 623–632, NY, USA, 2004. ACM.

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

[47] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient RSA key generation and threshold paillier in the two-party setting. In Orr Dunkelman, editor, *CT-RSA*, volume 7178 of *Lecture Notes in Computer Science*, pages 313–331. Springer, 2012.

[48] Xi He, Ashwin Machanavajjhala, Cheryl J. Flynn, and Divesh Srivastava. Scaling private record linkage using output constrained differential privacy. *CoRR*, abs/1702.00535, 2017.

[49] Adolf Hildebrand. On the least pair of consecutive quadratic nonresidues. *Michigan Math. J.*, 34(1):57–62, 1987.

[50] Albert O. Hirschman. The paternity of an index. *The American Economic Review*, 54(5):761–762, 1964.

[51] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, NY, USA, 2nd edition, 2012.

[52] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *J Mach Learn Res.*, 18:187:1–187:30, 2017.

[53] R. H. Hudson. The least pair of consecutive character non-residues. *Journal für die reine und angewandte Mathematik*, pages 219–220, 1974.

[54] Mihaela Ion, Ben Kreuter, Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. Private intersection-sum protocol with applications to attributing aggregate ad conversions. Cryptology ePrint Archive, Report 2017/738, 2017. http://eprint.iacr.org/2017/738.

[55] Mohammad Islam, Angelo K. Huang, Mohamed Battisha, Michelle Chiang, Santhosh Srinivasan, Craig Peters, Andreas Neumann, and Alejandro Abdelnur. 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)*, pages 4:1–4:10, 2012.

[56] Ernst Jacobsthal. *Anwendungen einer Formel aus der Theorie der quadratischen Reste*. PhD thesis, Friedrich Wilhelms Universität, Berlin, 1906.

[57] Roman Jagomägis. Secrec: a privacy-aware programming language with applications in data mining. *Master's thesis, University of Tartu*, 2010.

[58] Angela Jäschke and Frederik Armknecht. Accelerating homomorphic computations on rational numbers. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *Proc. 14th Int. Conf. ACNS 2016, Guildford, UK*, pages 405–423. Springer, 2016.

[59] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM SIGSAC*, pages 955–966. ACM, 2013.

[60] Somesh Jha, Louis Kruger, and Vitaly Shmatikov. Towards practical privacy for genomic computation. In *2008 IEEE Symposium on Security and Privacy*, pages 216–230, Oakland, CA, USA, May 18–21, 2008. IEEE Computer Society Press.

[61] Noah M. Johnson, Joseph P. Near, and Dawn Song. Practical differential privacy for SQL queries using elastic sensitivity. *CoRR*, abs/1706.09479, 2017.

[62] Kristján Valur Jónsson, Gunnar Kreitz, and Misbah Uddin. Secure multi-party sorting and applications. Cryptology ePrint Archive, Report 2011/122, 2011. http://eprint.iacr.org/2011/122.

[63] Seny Kamara and Tarik Moataz. SQL on structurally-encrypted databases. Cryptology ePrint Archive, Report 2016/453, 2016. http://eprint.iacr.org/2016/453.

[64] Seny Kamara, Payman Mohassel, Mariana Raykova, and Seyed Saeed Sadeghian. Scaling private set intersection to billion-element sets. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014: 18th International Conference on Financial Cryptography and Data Security*, volume 8437 of *Lecture Notes in Computer Science*, pages 195–215, Christ Church, Barbados, March 3–7, 2014. Springer, Heidelberg, Germany.

[65] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO*, volume 9215 of *Lecture Notes in Computer Science*, pages 724–741. Springer, 2015.

[66] Marcel Keller and Peter Scholl. Efficient, oblivious data structures for MPC. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 506–525, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.

[67] Florian Kerschbaum. Expression rewriting for optimizing secure computation. In *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 49–58, 2013.

[68] Mehmet S. Kiraz and Berry Schoenmakers. A protocol issue for the malicious case of Yao's garbled circuit construction. In *In Proceedings of 27th Symposium on Information Theory in the Benelux*, pages 283–290, 2006.

[69] Youness Lamzouri, Xiannan Li, and Kannan Soundararajan. Conditional bounds for the least quadratic non-residue and related problems. *Math. Comput.*, 84(295):2391–2412, 2015.

[70] Peeter Laud. Parallel oblivious array access for secure multiparty computation and privacy-preserving minimum spanning trees. *PoPETs*, 2015(2):188–205, 2015.

[71] Li, H., & Durbin, R. *Fast and accurate short read alignment with Burrows-Wheeler transform* . Bioinformatics, 25(14), 1754-1760., 2009.

[72] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

[73] U. V. Linnik. On the least prime in an arithmetic progression. i. the basic theorem. *Rec. Math. [Mat. Sbornik] N.S.*, 15(57):139–178, 1944.

[74] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17: 24th Conference on Computer and Communications Security*, pages 619–631, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[75] Richard F. Lukes, C. D. Patterson, and Hugh C. Williams. Some results on pseudosquares. *Math. Comput.*, 65(213):361–372, 1996.

[76] Manber, U., & Myers, G. *Suffix arrays: a new method for on-line string searches* . SIAM Journal on Computing, 22(5), 935-948., 1993.

[77] 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)*, pages 706–706, 2006.

[78] Payman Mohassel and Matthew K. Franklin. Efficiency tradeoffs for malicious two-party computation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC*, volume 3958 of *Lecture Notes in Computer Science*, pages 458–473. Springer, 2006.

[79] Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: A Timely Dataflow System. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP)*, pages 439–455, November 2013.

[80] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In Jeffrey Scott Vitter, Lawrence L. Larmore, and Frank Thomson Leighton, editors, *STOC*, pages 245–254. ACM, 1999.

[81] Arjun Narayan and Andreas Haeberlen. Djoin: Differentially private join queries over distributed databases. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, pages 149–162, October 2012.

[82] New York City Taxi & Limousine Commission. Taxicab factbook, 2014.

[83] O. Goldreich. *Foundations of Cryptography: Volume II Basic Applications*. Cambridge University Press, 2004.

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

[85] Michele Orrù, Emmanuela Orsini, and Peter Scholl. Actively secure 1-out-of-n OT extension with application to private set intersection. In Helena Handschuh, editor, *CT-RSA*, volume 10159 of *Lecture Notes in Computer Science*, pages 381–396. Springer, 2017.

[86] Antonis Papadimitriou, Arjun Narayan, and Andreas Haeberlen. Dstress: Efficient differentially private computations on distributed data. In *Proceedings of the 12th European Conference on Computer Systems (EuroSys)*, pages 560–574, 2017.

[87] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos D. Keromytis, and Steve Bellovin. Blind seer: A scalable private DBMS. In *2014 IEEE Symposium on Security and Privacy*, pages 359–374, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press.

[88] Patil, Manish, Xuanting Cai, Sharma V. Thankachan, Rahul Shah, Seung-Jong Park, and David Foltz. *Approximate string matching by position restricted alignment* . Proceedings of the Joint EDBT/ICDT 2013 Workshops, pp. 384-391. ACM, 2013.

[89] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David A. Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2008.

[90] Guillaume Poupard and Jacques Stern. Generation of shared RSA keys by two parties. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT*, volume 1514 of *Lecture Notes in Computer Science*, pages 11–24. Springer, 1998.

[91] Aseem Rastogi, Matthew A. Hammer, and Michael Hicks. Wysteria: A programming language for generic, mixed-mode multiparty computations. In *2014 IEEE Symposium on Security and Privacy*, pages 655–670, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press.

[92] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[93] Hans Salié. Über den kleinsten positiven quadratischen nichtrest nach einer primzahl. *Mathematische Nachrichten*, 3(1):7–8, 1949.

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

[95] Claude Elwood Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 7 1948.

[96] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220. Springer, 2000.

[97] Ebrahim M. Songhori, Siam U. Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *IEEE Symposium on Security and Privacy*, pages 411–428. IEEE Computer Society, 2015.

[98] Zhi-Hong Sun. Consecutive numbers with the same Legendre symbol. *Proc. of the American Math. Soc.*, 130(9):2503–2507, 2002.

[99] Doug Szajda, Michael Pohl, Jason Owen, and Barry G. Lawson. Toward a practical data privacy scheme for a distributed implementation of the Smith-Waterman genome sequence comparison algorithm. In *ISOC Network and Distributed System Security Symposium – NDSS 2006*, San Diego, CA, USA, February 2–3, 2006. The Internet Society.

[100] A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive – A Warehousing Solution over a Map-Reduce Framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[101] Tomas Toft. *Primitives and Applications for Multi-party Computation*. PhD thesis, 2007.

[102] Tomas Toft. Solving linear programs using multiparty computation. In Roger Dingledine and Philippe Golle, editors, *FC 2009: 13th International Conference on Financial Cryptography and Data Security*, volume 5628 of *Lecture Notes in Computer Science*, pages 90–107, Accra Beach, Barbados, February 23–26, 2009. Springer, Heidelberg, Germany.

[103] Enrique Treviño. The least k-th power non-residue. *Journal of Number Theory*, 149:201 – 224, 2015.

[104] U.S. Census Bureau. *Table 1188 – Credit Cards-Holders, Number, Spending, Debt, and Projections*, chapter 25: Banking, Finance, and Insurance. 131 edition, August 2011.

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

[106] U.S. Office of the Comptroller of the Currency.

[107] U.S. Social Security Administration. Social security faqs. Q20.

[108] Brigitte Vallée. Gauss' algorithm revisited. *Journal of Algorithms*, 12(4):556 – 572, 1991.

[109] Meilof Veeningen. Pinocchio-based adaptive zk-SNARKs and secure/correct adaptive function evaluation. In Marc Joye and Abderrahmane Nitaj, editors, *AFRICACRYPT 17: 9th International Conference on Cryptology in Africa*, volume 10239 of *Lecture Notes in Computer Science*, pages 21–39, Dakar, Senegal, May 24–26, 2017. Springer, Heidelberg, Germany.

[110] P. S. Wang. A p-adic algorithm for univariate partial fractions. In *Proceedings of the Fourth ACM Symposium on Symbolic and Algebraic Computation*, SYMSAC '81, pages 212–217, NY, USA, 1981. ACM.

[111] P. S. Wang, M. J. T. Guy, and J. H. Davenport. P-adic reconstruction of rational numbers. *SIGSAM Bull.*, 16(2):2–3, May 1982.

[112] Triantafyllos Xylouris. *Über die Nullstellen der Dirichletschen L-Funktionen und die kleinste Primzahl in einer arithmetischen Progression*. PhD thesis, Rheinischen Friedrich-Wilhelms-Universität Bonn, 2011.

[113] A. C. Yao. Protocols for secure computations. In *23rd Annual Symp. FOCS*, volume 00, pages 160–164, 11 1982.

[114] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

[115] Ching-Hua Yu. Sign modules in secure arithmetic circuits. Cryptology ePrint Archive, Report 2011/539, 2011. http://eprint.iacr.org/2011/539.

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

[117] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 15–28, April 2012.

[118] Samee Zahur and David Evans. Obliv-C: A language for extensible data-oblivious computation. Cryptology ePrint Archive, Report 2015/1153, 2015. http://eprint.iacr.org/2015/1153.

[119] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 283–298, Boston, Massachusetts, USA, 2017.