HOLMES: Efficient Distribution Testing for Secure Collaborative Learning

Ian Chang
UC Berkeley

Katerina Sotiraki
UC Berkeley

Weikeng Chen
UC Berkeley & DZK Labs

Murat Kantarcioglu
University of Texas at Dallas & UC Berkeley

Raluca Ada Popa
UC Berkeley

Abstract

Using *secure multiparty computation (MPC)*, organizations which own sensitive data (e.g., in healthcare, finance or law enforcement) can train machine learning models over their joint dataset without revealing their data to each other. At the same time, secure computation restricts operations on the joint dataset, which impedes computation to assess its quality. Without such an assessment, deploying a jointly trained model is potentially illegal. Regulations, such as the European Union's General Data Protection Regulation (GDPR), require organizations to be legally responsible for the errors, bias, or discrimination caused by their machine learning models. Hence, testing data quality emerges as an *indispensable* step in secure collaborative learning. However, performing distribution testing is prohibitively expensive using current techniques, as shown in our experiments.

We present HOLMES, a protocol for performing distribution testing *efficiently*. In our experiments, compared with three non-trivial baselines, HOLMES achieves a speedup of more than $10\times$ for classical distribution tests and up to $10^4\times$ for multidimensional tests. The core of HOLMES is a hybrid protocol that integrates MPC with zero-knowledge proofs and a new ZK-friendly and naturally oblivious sketching algorithm for multidimensional tests, both with *significantly lower* computational complexity and concrete execution costs.

1 Introduction

The MIT Technology Review article, "AI is sending people to jail–and getting it wrong" [1] is a reminder that machine learning models are determining people's fate. The article explains how COMPAS[1], a system that rates people's risk of future crime and decides if one should be held in jail before trial, has been shown to disproportionately target low-income populations and minorities [1]. Other examples where AI has infiltrated our everyday life include models to detect traditionally unrecognized anxiety and depression from speech [2, 3] or diagnose attention deficiency [4]. In these cases, accuracy is crucial for people to receive the right treatment.

However, if the training data is inaccurate, skewed, or affected by systemic biases, without any special attention to this issue, the trained model will also be biased [5]. There are many approaches (e.g., [6–9]) to guarantee fairness, starting from detecting and removing imbalances from the training data. For example, if a dataset has a large number of negative records (e.g., low credit scores) toward a certain group, one can reduce the imbalance by subsampling.

The situation in *secure collaborative learning*, where the model is trained in a way that none of the parties has access to the whole dataset, is vastly different. If the quality of the joint dataset is good, we expect the model trained on the joint dataset to be superior to models trained on individual datasets [10–14]. However, due to privacy considerations (e.g., due to GDPR [15]) organizations cannot know if the data is indeed of high quality. GDPR also requires organizations using such models to *prevent* errors, bias, and discrimination and *take liability* of the model [16]. Hence, organizations face the following conundrum. How can an organization take the (unknown) risk for another organization's *untested* data?

Organizations will use collaborative secure computation only if there are *data quality* guarantees about the joint dataset. Hence, the ability to check data quality in secure computation is as important as data confidentiality and integrity. The first step toward this direction is to perform *distribution testing* over the joint dataset to examine:

- **one-dimensional properties,** such as the histogram of values of a specific attribute (e.g., income), or as basic as checking the range of each entry (e.g., age should be ≤ 200).
- **multidimensional properties,** such as whether the distribution of several attributes (e.g., age, gender, and income) *fits* into a desired distribution (e.g., represents a balanced demographic composition).

Distribution testing is a prominent method in statistical analysis. For instance, in clinical trials, the NIH Collaboratory [17] recommends comparing the distribution of different

[1]Correctional Offender Management Profiling for Alternative Sanctions.

datasets to detect data discrepancies. However, this useful tool is missing in prior works of secure collaborative learning (e.g., [18–21]), often left as an open problem. This is likely due to its *extremely high* overhead in MPC.

We present HOLMES, a protocol that performs such distribution testing efficiently, often at only a small fraction of the cost of secure collaborative learning. In our experiments, compared to three non-trivial baselines that we describe below, HOLMES achieves a speedup of more than $10\times$ for classical one-dimensional tests and up to $10^4\times$ for multidimensional tests. The core of HOLMES is a hybrid protocol that integrates MPC with zero-knowledge proofs and a new ZK-friendly, naturally oblivious sketching algorithm for multidimensional tests. HOLMES is already open-sourced in GitHub (anonymously): https://github.com/holmes-inputcheck/

1.1 Utilising IZK

Intuitively, to guarantee privacy no data-dependent computation should be performed outside the MPC protocol used in secure collaborative learning. So, bypassing the inefficiency of MPC seems impossible. Our insight is that the computation in distribution testing can be divided into parts that involve a single party's dataset. Hence, distribution testing is mostly a *verification* task, instead of a direct computation. This brings us to the non-trivial first baseline described below.

First Baseline: Each party provides some auxiliary information based on their individual dataset, called *witness*, for the distribution tests. All parties verify each party's computation using the witnesses and proceed to compute the distribution tests in MPC. Verifying a computation can be significantly faster than directly computing. For example, verifying that a value x is in the range $[a,b]$ typically involves computing the bit decomposition of $(x-a)$ and $(b-x)$. Hence, it becomes easier when each party provides this information as a witness.

However, this solution is still costly to implement in secure collaborative systems, when we require malicious security with a dishonest majority. SPDZ [22] and SCALE-MAMBA [23] are the fastest well-known MPC protocols for generic arithmetic computations with a dishonest and malicious majority, and are commonly used in secure collaborative learning (e.g., [19]). Since efficient distribution testing algorithms are based primarily on linear arithmetic operations, SPDZ-type protocols have lower computational overhead than other general MPC approaches, such as garbled circuits.

However, the cost of performing distribution tests still grows rapidly in this setting. Assume that t parties, each with the same amount of data, want to check the data quality of each individual dataset using distribution tests. We specifically use the standardized measure "wall-clock time" to describe the computational overhead, which refers to the amount of time **each** party takes until the MPC protocol finishes.

- In the best case, all t parties agree on the same set of distribution tests for all individual datasets. If the computation of the tests on a single dataset requires C multiplications, the computation for t datasets has cost $C \cdot t$ multiplications, and the online phase of SCALE-MAMBA has computational overhead $O(C \cdot t)$ field elements [2]. Moreover, the offline computational overhead of SCALE-MAMBA is proportional to the product of number of parties and the size of the online computation, i.e., $O(C \cdot t^2)$, since for each multiplication we need to produce t pieces of information, one for each party. Thus, running the distribution tests on t datasets in this t-party MPC leads to a wall-clock time of $O(C \cdot t^2)$.
- In the worst case, each party provides a different set of distribution tests for each individual dataset. Assuming that each set of tests has computational overhead $O(C)$ in SCALE-MAMBA, the online computational overhead using a t-party protocol is $O(C \cdot t^2)$; this is because the online cost for all sets of tests on each individual dataset is $O(C \cdot t)$. Taking into account the offline phase, running the distribution tests on all t datasets in this t-party MPC leads to a wall-clock time of $O(C \cdot t^3)$.

We present a solution that reduces the wall-clock time to $O(C \cdot t)$ in all cases.

HOLMES: efficiency via IZK. Zero-knowledge proofs are protocols that allow verifying a computation on a single party's dataset, without revealing any other information. Thus, they are more suited for efficient distribution testing. Specifically, using interactive zero-knowledge (IZK) proofs, which are 2-party protocols involving a prover and a verifier, each party proves to $(t-1)$ parties the distribution tests for its individual dataset, and verifies the tests for the other $(t-1)$ datasets. In contrast to the first baseline, now the pairwise IZK protocols can be run concurrently, i.e., while $\mathcal{P}_1$ runs an IZK with $\mathcal{P}_2$, the parties $\mathcal{P}_3$ and $\mathcal{P}_4$ can also run an IZK etc. Hence, if the computational overhead in IZK for the distribution tests on an individual dataset is $O(C)$, the wall-clock time for the datasets in IZK is $O(C \cdot t)$: each party proves $t-1$, potentially different, statements and verifies $t-1$ statements.

IZK reduces the computational overhead as performing distribution tests on an individual dataset, in essence, is not a computation that requires input from t parties; thus, it does not actually need t-party MPC. This new computational model leaves two challenges:

1. How can we ensure that the data in the MPC for collaborative learning is the same data used in IZK?
2. Are IZK-based solutions concretely efficient?

We answer the first question in §3.1 by proposing a lightweight consistency check. Additionally, our evaluation in §4 confirms that the answer to the second question is yes.

Remark: choice of IZK protocol. A generic approach for IZK is to use pairwise secure two-party computation (2PC) protocols. Namely, each pair of parties performs a 2PC protocol to show that a set of tests on its individual dataset passes

[2] Additions do not require communication in SCALE-MAMBA.

and to verify a (potentially different) set of tests on the other party's dataset. However, there are also specialized IZK protocols that focus on the functionality of proving statements instead of a general two-party computation. In §4.4.1, we provide a comparison between HOLMES and other baselines. In HOLMES, we use QuickSilver [24] as the underlying IZK protocol. We show that the baseline of pairwise generic 2PC protocols, e.g., using SCALE-MAMBA, is up to 46× slower. Additionally, we investigate baselines based on non-interactive ZK (NIZK) protocols, e.g., Spartan [25], which may asymptotically reduce the communication of each party to $O(C)$, as in this case each party only needs to produce a single proof for the tests on its individual dataset and broadcast it to the other $t-1$ parties. However, NIZK protocols typically suffer from large proving times. Indeed, we find that producing an NIZK proof (i.e., using Spartan) is slower than performing $t-1$ IZK protocols (i.e., using QuickSilver) for various dataset sizes and up to $t = 10$ parties. Since we are interested in the most efficient solution, including the computational overhead of each party, we use QuickSilver. We leave as an open problem to find an NIZK that is concretely more efficient in our setting.

1.2 Testing multidimensional properties

We now turn our focus to the task of *efficiently* testing multidimensional properties of distributions. For example, in a financial dataset, instead of focusing on a single attribute (i.e., one dimension) such as "debt", we want to understand the properties of the joint distribution of several attributes (i.e., multiple dimensions), for example gender, age, race, income level, and debt. A property we might want to ensure is that the histogram of these attributes is not far from a balanced distribution, in which different genders, ages, races, and income levels are fairly represented.

In plaintext systems, this can be done by directly computing the histogram; namely, putting the data into buckets, where each bucket represents a different combination of values for the attributes gender, age, race, income level, and debt. Then, the distribution test is done by performing a Pearson's χ^2-test. In secure computation, the bucketing must be *oblivious*, in that it cannot reveal to which bucket a sample belongs. This either requires a linear scan of the buckets for each sample, as shown in Fig. 1, or oblivious RAM (ORAM) [26–33] in MPC, which theoretically has better complexity but in practice is concretely expensive.

The curse of oblivious bucketing. As shown in Fig. 1, in linear scan besides the real update (indicated by a black line), there are many dummy updates (indicated by red lines). In particular, the total number of updates is equal to the number of buckets. If the i-th attribute takes D_i distinct values, then the number of buckets is $\prod_i D_i = D_1 \cdot D_2 \cdot \ldots \cdot D_d$, which quickly becomes prohibitively large. Therefore, the cost of oblivious bucketing for N data points is $O(N \cdot \prod_i D_i)$. Our experiments

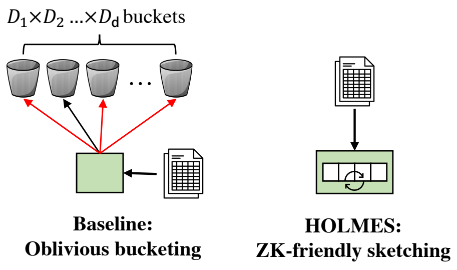


Figure 1: Methods for multidimensional distribution testing.

in §4 show that such oblivious bucketing for 5 attributes with values ranging from 1 to 10 takes 10^5 seconds.

Streaming and sketching to the rescue. We avoid oblivious bucketing by utilizing two concepts from algorithm design: *streaming* and *sketching*.

- **Streaming:** an algorithm that takes as input a sequence and only needs access to *limited* memory.
- **Sketching:** an algorithm that (approximately) performs the computation, using a *compressed* representation of the data.

We present a *sketching* algorithm that, given pseudorandomness, compresses the histogram of a dataset while preserving the necessary information for applying Pearson's χ^2-test. We also show how to compute this compressed representation in a *streaming* fashion, i.e., by accessing sequentially each entry in the dataset. This algorithm applies a *random linear projection* that approximately preserves the ℓ_2-norm, according to the *Johnson-Lindenstrauss lemma* [34].

A challenge that arises in our algorithm is how to efficiently obtain pseudorandomness in IZK, as running classical pseudorandom functions, such as SHA-256, is impractical. Instead, we strive to find a *tailored* way to obtain pseudorandomness for our algorithm. We call this construction "ZK-friendly sketching".

Finding pseudorandomness for random projection. Our random projection requires r pseudorandom maps with an one-bit output $b \in \{-1, 1\}$. Although we can use any pseudorandom function and extract one bit from it through bit decomposition, we find that many ZK-friendly hash functions [35, 36] are still costly. Instead, we discover that Legendre PRF, which has been studied recently [37–40] and is conjectured to be a universal one-way hash function (UOWHF) [41, 42] with a one-bit extractor, is a natural fit, and is extremely cheap–it only requires 8 input or multiplication gates in IZK.

With these techniques, HOLMES's multidimensional distribution testing only requires $O(r \cdot N)$ computation, where r is the output size of the random linear projection. In our experiments, we show that this solution is up to $10^4\times$ faster compared to linear scan.

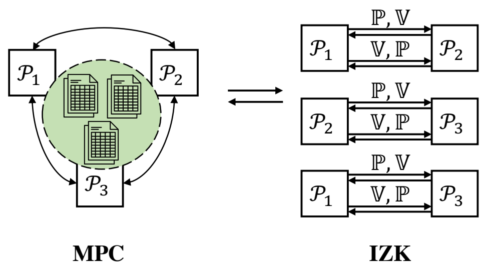


Figure 2: System model of HOLMES.

1.3 Summary of contributions

In summary, HOLMES's contributions are:

- a new hybrid protocol that integrates MPC, IZK, and a lightweight consistency check for distribution testing, which has lower complexity and is concretely much more efficient (at least $10\times$ in our experiments) compared to three non-trivial baselines;
- a new efficient multidimensional distribution testing procedure via ZK-friendly sketching, which has lower complexity and is concretely much more efficient (up to $10^4\times$ in our experiment) compared to the naive multidimensional distribution testing.

Using HOLMES, we create a library of well-known statistical tests, such as z-test, t-test, F-test, and χ^2-test. We also perform extensive experimental evaluation of HOLMES, including evaluation using real-world datasets from bank marketing [43, 44], healthcare [45, 46], and online auctions [47].

2 Protocol Overview

We assume that t parties want to participate in secure collaborative learning based on a t-party MPC protocol (e.g., SCALE-MAMBA). In HOLMES, the parties perform two types of computation: MPC and IZK, as shown in Fig. 2. During the distribution testing, parties participate in a t-party MPC, which is also used for the collaborative learning, and run IZK in a *pairwise* and *bidirectional* manner, where each party communicates with every other party, and both parties take turns as a prover and verifier to perform distribution tests on each other's individual dataset. We require that MPC and IZK allow parties to load their datasets before learning the distribution tests, which prevents adaptively changing their input. We formalize this property in §3.1.

Tests. A distribution test is a predicate over an individual or joint (i.e., from multiple parties) dataset. Examples include well-known statistical tests, such as mean equality z-test (when the variance of the dataset is known) and t-test (when the variance is unknown), variance equality F-test, and Pearson's χ^2-test. These tests check a property between two populations, or between a population and a public distribution. Each population can be an individual or a joint dataset and can contain data points with multiple attributes (e.g., age, gender, income, etc.). HOLMES implements these tests and various distribution test gadgets, as shown in Tab. 7.

Workflow. HOLMES runs as a subroutine in the early stages of secure collaborative learning when all parties have loaded their datasets in MPC. HOLMES is invoked to perform distribution tests before MPC starts to run the actual training algorithm, as follows.

1. **Input loading in IZK:** Each party loads its dataset in IZK. This prevents the party from changing the input adaptively after seeing the revealed distribution tests.
2. **Revealing the distribution tests:** The parties reveal the distribution tests they want to perform.
3. **Consistency check:** Parties perform a consistency check (e.g., as in §3.1) to verify that the input loaded in IZK and MPC is the same. Parties reject if the check fails.
4. **IZK verification:** Each pair of parties acts as the prover and the verifier in IZK. The prover proves the correct calculation of some specified statistics about their dataset. The verifier verifies the proof and rejects if IZK fails.
5. **MPC finishing touches:** For distribution tests over joint datasets (e.g., z, t, F-tests), parties decide whether the data passes the test in MPC. Here, MPC only performs a small computation over *statistics* verified in IZK; thus we call it the "finishing touches".

If all parties accept the distribution tests in HOLMES, the secure collaborative learning continues.

2.1 Protocol Components

HOLMES combines three underlying protocols:

1. an MPC protocol Π_{mpc}, (Definition 3)which performs computations involving two or more parties;
2. an IZK protocol Π_{izk}, (Definition 4)which verifies computation performed by a single party;
3. a consistency check (CC) protocol Π_{cc} (§3.1), which ensures that a party loads the same inputs in the MPC and IZK.

The underlying MPC and IZK protocols have to additionally satisfy the following properties.

Definition 1. *An MPC protocol* Π *as in Definition 3enables* input-loading *if in the beginning of the protocol each party* $\mathcal{P}_i$ *with input* $\mathbf{x}_i$*, without access to the function* $\mathcal{F}$*, computes and sends* $\mathsf{load}_{\text{mpc}}(j, \mathbf{x}_i)$ *to party* $\mathcal{P}_j$*. Then, the parties receive* $\mathcal{F}$ *and proceed to the protocol with inputs* $\mathsf{load}_{\text{mpc}}(j, \mathbf{x}_i)$*, without further access to the inputs* $\mathbf{x}_i$*.*

Definition 2. *An IZK protocol* Π *for the language* $\mathcal{L} = \{\mathbb{x} : \exists \mathbb{w} \text{ s.t. } (\mathbb{w}, \mathbb{x}) \in \mathcal{R}\}$ *as in Definition 4enables* input-loading *if in the beginning of the protocol the prover sends* $l :=$

$\mathsf{load}_{\mathrm{izk}}(\mathbb{w})$, *where* $\mathsf{load}_{\mathrm{izk}}$ *is a cryptographic commitment (Definition 6)and during the protocol the prover proves that* $\mathbb{x} \in \mathcal{L}$ *and* l *is a commitment to* $\mathbb{w}$*, i.e., runs an IZK for the language* $\mathcal{L}' = \{(\mathbb{x}, l) : \exists \mathbb{w}$ *s.t.* $(\mathbb{w}, \mathbb{x}) \in \mathcal{R}$ *and* $l = \mathsf{Com}(\mathbb{w})\}$.

We remark that the majority of MPC and IZK protocols are input-loading. For MPC, $\mathsf{load}_{\mathrm{mpc}}$ typically corresponds to secret sharing (e.g. [23, 48, 49]).

The protocol of HOLMES is described in Fig. 3. We define the security of HOLMES in the real/ideal-world paradigm using the standard definition for (standalone) malicious security [50] as in Definition 3. In malicious security, up to $t-1$ of the parties can collude (statically) and arbitrarily deviate from the protocol. The ideal functionality $\mathcal{F}_{HOLMES}$ (Fig. 3) takes as input the list of distribution tests and the datasets; it outputs whether the datasets pass the tests or fail. Finally, in Appendix E we provide the security proof of the following theorem.

Theorem 1. *If* Π_{mpc} *is a secure MPC protocol that enables input-loading,* Π_{izk} *is an IZK protocol that enables input-loading, and* Π_{cc} *is a consistency check protocol (Definition 7), then HOLMES securely computes* $\mathcal{F}_{HOLMES}$.

Choice of protocols. HOLMES can be instantiated with a variety of MPC and IZK protocols, as long as they satisfy Definition 1 and Definition 2. In §3.1, we present a lightweight consistency check for MPC and IZK with additional, yet typical, properties, and in §4, we discuss how different protocol choices affect the efficiency of HOLMES.

Leakage from distribution tests. Note that any distribution testing leaks one-bit information – whether the test passed or failed. Hence, it is important that a party does not participate in distribution tests that may leak sensitive information. We leave as an open direction quantifying the privacy loss due to distribution tests, as discussed in §6.

Input-loading as a safeguard. Informally, the input-loading phase impedes parties from trying to tune their input data to pass the distribution tests. Even if parties abort after learning the distribution tests (e.g., pretending that the network is down) and request to redo the distribution testing, input-loading enforces that parties in the second execution must use the initial data.

3 Our Design

3.1 Consistency Check

HOLMES ensures that data used in MPC and IZK is the same via a *consistency check*, which is a protocol between a prover and $t-1$ verifiers. We define this functionality formally in Appendix B. We now describe our lightweight protocol for specific types of MPC and IZK, which we use in our implementation of HOLMES. The proof of Lemma 1 appears in Appendix B.

Lemma 1. *The construction below is a consistency check.*

Construction. Let Π_{mpc} be a secure MPC protocol over field $\mathbb{F}_p$ that enables input-loading through additive secret sharing(Definition 5); namely, for an input $\mathbf{x} = (x_1, \ldots, x_N) \in \mathbb{F}_p^N$, $(\mathsf{load}_{\mathrm{mpc}}(j, \mathbf{x}))_{j \in [t]} = (\mathsf{share}_j(x_1), \ldots, \mathsf{share}_j(x_N))_{j \in [t]}$.

Let Π_{izk} be an IZK protocol over field $\mathbb{F}_p$ that enables input-loading through homomorphic commitments(Definition 6); namely for an input $\mathbf{x} = (x_1, \ldots, x_N) \in \mathbb{F}_p^N$, $(\mathsf{load}_{\mathrm{izk}}(j, \mathbf{x}))_{j \in [t]} = (\mathsf{Com}_{\mathsf{ck}_j}(x_1), \ldots, \mathsf{Com}_{\mathsf{ck}_j}(x_N))_{j \in [t]}$. The protocol Π_{cc} works as follows.

1. Prover $\mathbb{P}$ samples a random number $r \leftarrow\$ \mathbb{F}_p$ and sends $(\mathsf{load}_{\mathrm{mpc}}(j, r), \mathsf{load}_{\mathrm{izk}}(j, r)) := (\mathsf{share}_j(r), \mathsf{Com}_{\mathsf{ck}_j}(r))$ to each verifier $\mathbb{V}_j$.
2. Parties run a coin toss protocol [51] to obtain a random challenge $\beta \leftarrow\$ \mathbb{F}_p$.
3. Parties run an MPC protocol to compute $\rho := r + \sum_{k=1}^{N} x_k \cdot \beta^k$. Note that each $\mathbb{V}_j$ knows $(\mathsf{share}_j(x_1), \ldots, \mathsf{share}_j(x_N), \mathsf{share}_j(r))$.
4. Prover $\mathbb{P}$ runs an IZK proof with each verifier $\mathbb{V}_j$ for proving that $\mathsf{Com}_{\mathsf{ck}_j}(\rho) - \mathsf{Com}_{\mathsf{ck}_j}(r) - \sum_{k=1}^{N} \mathsf{Com}_{\mathsf{ck}_j}(x_k) \cdot \beta^k = \mathsf{Com}_{\mathsf{ck}_j}(0)$.
5. Verifiers accept if all IZK proofs are valid. Otherwise, they reject.

Cost analysis. Computing ρ and $\mathsf{Com}_{\mathsf{ck}_j}(\rho) - \mathsf{Com}_{\mathsf{ck}_j}(r) - \sum_{k=1}^{N} \mathsf{Com}_{\mathsf{ck}_j}(x_k) \cdot \beta^k$ requires $O(N)$ local operations since the MPC uses additive secret sharing and the commitment scheme is homomorphic. Hence, the main cost is due to the coin toss protocol and the IZK for proving that a value is a commitment to 0. Both of these functionalities are very efficient in state-of-the-art systems.

3.2 One-Dimensional Distribution Tests

We now provide more details about how IZK and MPC work together in each distribution test. We describe one-dimensional tests in this section, and present multidimensional tests in §3.3. We split the distribution tests into two steps: (1) compute statistical properties of an individual dataset, which is done in IZK, as described in §3.2.1; and (2) perform distribution tests using the computed statistical properties. If the distribution test is over an individual dataset, the second step is performed in IZK, but if it is over a joint dataset, the second step is done in MPC, as described in §3.2.2. We discuss the cost of these distribution tests in Appendix C.

3.2.1 IZK: Verifying basic statistics

HOLMES verifies basic statistical properties in IZK over an individual dataset: range, histogram, mean, variance, and trimmed mean.

Range. To prove that $a \leq x \leq b$ where $x \in \mathbb{F}_p$, the prover $\mathbb{P}$ shows that $x - a \geq 0$ and $b - x \geq 0$. We describe in detail the range check for a single element $x \in \mathbb{F}_p$ in Appendix F. The

HOLMES protocol

For t parties $\mathcal{P}_1, \cdots, \mathcal{P}_t$ with inputs $(\mathbf{x}_1, \ldots, \mathbf{x}_t)$ respectively, the protocol proceeds as follows:

1. **MPC input-loading:** Each party $\mathcal{P}_i$ sends $\ell_{\text{mpc}}[i,j] := \mathsf{load}_{\text{mpc}}(j, \mathbf{x}_i)$ to party $\mathcal{P}_j$ for $j \in [t] \setminus \{i\}$.
2. **IZK input-loading:** Each party $\mathcal{P}_i$ initiates $t-1$ IZK protocols as a prover $\mathbb{P}$ with $\mathbb{w} = \mathbf{x}$ with parties $\mathbb{V} = \mathcal{P}_j$ for $j \in [t] \setminus \{i\}$. $\mathcal{P}_i$ sends $\ell_{\text{izk}}[i,j] := \mathsf{load}_{\text{izk}}(j, \mathbf{x}_i)$ to party $\mathcal{P}_j$ for $j \in [t] \setminus \{i\}$, where $\mathsf{load}_{\text{izk}}(j, \cdot)$ is the $\mathsf{load}_{\text{izk}}$ function of the IZK with $\mathbb{P} = \mathcal{P}_i$ and $\mathbb{V} = \mathcal{P}_j$.
3. **Distribution tests:** Each party $\mathcal{P}_i$ sends a set of distribution tests tests_i to each party $\mathcal{P}_j$ for $j \in [t] \setminus \{i\}$.
4. **Consistency check:** Each party $\mathcal{P}_i$ performs Π_{cc} with the other $t-1$ parties to show that $\ell_{\text{mpc}}[i,j] = \mathsf{load}_{\text{mpc}}(j, \mathbf{x}_i)$ and $\ell_{\text{izk}}[i,j] = \mathsf{load}_{\text{izk}}(j, \mathbf{x}_i)$.
5. **IZK protocol:** Each party $\mathcal{P}_i$ performs Π_{izk} with $\mathcal{P}_j$ for each test in tests_j as described in §3.2 and §3.3.
6. **MPC protocol:** Parties perform Π_{mpc} for each test in tests_i for $i \in [t]$ as described in §3.2 and §3.3. For each test, the parties either output pass or fail.

Ideal functionality $\mathcal{F}_{HOLMES}$

For t parties $\mathcal{P}_1, \cdots, \mathcal{P}_t$ and a simulator Sim, $\mathcal{F}_{HOLMES}$ proceeds as follows: Upon receiving a message $(\mathcal{P}_i, \mathbf{x}_i, \mathsf{tests}_i)$ from each of the t parties (or from Sim if that party is corrupted),

1. **Abort:** $\mathcal{F}_{HOLMES}$ awaits a message $\mathsf{deliver}$ or abort from Sim to decide whether the computation should move forward. $\mathcal{F}_{HOLMES}$ proceeds to the next step if the message is $\mathsf{deliver}$. Otherwise, $\mathcal{F}_{HOLMES}$ sends abort to each $\mathcal{P}_i$ and Sim, and halts.
2. **Distribution tests:** $\mathcal{F}_{HOLMES}$ runs the distribution tests specified in $(\mathsf{tests}_1, \ldots, \mathsf{tests}_t)$ with respect to the inputs $(\mathbf{x}_1, \ldots, \mathbf{x}_t)$. For each distribution test, $\mathcal{F}_{HOLMES}$ sends either pass or fail to each $\mathcal{P}_i$ and Sim, and halts.

Figure 3: HOLMES protocol and ideal functionality $\mathcal{F}_{HOLMES}$.

gadget $\mathsf{range}(\langle S \rangle, \mathsf{attr}, [a,b])$ performs a range check on each data point of the population S with respect to attribute attr.

Histogram. The histogram of a population S for an attribute attr with values in $\mathbb{F}_p$ counts the data points in a set of non-overlapping buckets. Each bucket might correspond to a single value or a range $[a,b]$. For instance, for the attribute "marital status", we have single-value buckets (e.g., single, married, etc.), whereas for the attribute "age" we might be interested in range-buckets (e.g., 0-10, 11-20, etc.). In our setting, all values are elements in $\mathbb{F}_p$, so a single value a can be described by the range $[a,a]$. Hence, we focus on the case of range-buckets.

The prover $\mathbb{P}$ first computes a one-hot encoding (OHE) $\overrightarrow{\sigma} = (\sigma_k)_{k=1}^{D} = (0,0,...,1,...,0,0)$ for each entry such that if σ_k is 1, the entry belongs to the k-th bucket. In Appendix F, we describe how the prover $\mathbb{P}$ proves that $\overrightarrow{\sigma}$ is a valid one-hot encoding and how to perform the histogram check. We also discuss the extension to the multidimensional case, where each data point has d attributes (i.e., is in $\mathbb{F}_p^{\mathsf{d}}$) and each bucket corresponds to d ranges $([a_i,b_i])_{i=1}^{\mathsf{d}}$, one for each attribute. The gadget $\mathsf{histogram}(\langle S \rangle, (\mathsf{attr}_1, ..., \mathsf{attr}_{\mathsf{d}}), (\mathfrak{b}_1, \ldots, \mathfrak{b}_D)) \to \overrightarrow{\mathsf{count}}$ performs a histogram check on attributes $\mathsf{attr}_1, ..., \mathsf{attr}_{\mathsf{d}}$ for the population S with respect to buckets $\mathfrak{b}_1, \ldots, \mathfrak{b}_D$.

Mean and variance. Mean and variance are essential in many tests, such as z-tests and t-tests. To prove that $\bar{x}$ is the mean of values $(x_j)_{j=1}^{N}$, the prover $\mathbb{P}$ shows that $N \cdot \bar{x} \approx \sum_{j=1}^{N} x_j$. In practice, we want to keep a few decimal places for $\bar{x}$ (e.g., $\bar{x} = 12.34$ with two decimal places). This is done by defining $\bar{x}' = 1234$, a fixed-point representation of $\bar{x}$, and asking $\mathbb{P}$ to show that $N \cdot \bar{x}' \leq 100 \cdot \sum_{j=1}^{N} x_j < N \cdot (\bar{x}' + 1)$ using (Algorithm 1). To prove the correct calculation of the variance s^2, $\mathbb{P}$ first proves the calculation of the mean $\bar{x}$ and of the mean of the square of each value, $\bar{y}$. The variance can be verified by checking that $s^2 \approx \frac{N}{N-1}(\bar{y} - \bar{x}^2)$.[3] We provide the two gadgets $\mathsf{mean}(\langle S \rangle, \mathsf{attr}) \to \bar{x}$ and $\mathsf{variance}(\langle S \rangle, \mathsf{attr}) \to s^2$.

Trimmed mean. Trimmed mean is similar to mean, but it only considers entries with values within a certain range $[0, \theta]$. This statistic is useful as it can remove extreme values before computing the mean. This check combines range checks and a mean check as shown in Appendix F. HOLMES implements trimmed mean in the gadget $\mathsf{trimmedMean}(\langle S \rangle, \mathsf{attr}, \theta) \to \tilde{x}$.

3.2.2 MPC: Finishing touches

Distribution tests use the basic statistics of §3.2.1. If the tests involve an individual dataset, they are computed in IZK. If they involve dataset from multiple parties, the final computation is done in MPC. Since basic statistics are verified in IZK, only the "finishing touches" (i.e., a small computation) are performed in MPC. We discuss how to perform well-known statistical tests: z-test, t-test, F-test, and Pearson's χ^2-test. Be-

[3]The term $N/(N-1)$ corrects the bias of the variance because $\bar{x}$ is computed from the data [52].

low we denote by yellow the basic statistics that depend on private datasets and have been verified in IZK.

***z*-test.** This distribution test checks whether the means of two populations S_1 and S_2 (of size N_1, N_2) for the attribute $\mathtt{attr}$ are equal, assuming known variances. The parties provide the means $\bar{x}_1$ and $\bar{x}_2$, and the test passes if:

$$(\bar{x}_1 - \bar{x}_2) / \sqrt{\sigma_1^2/N_1 + \sigma_2^2/N_2} \overset{?}{\leq} T_{\alpha,N_1,N_2},$$

where T_{α,N_1,N_2} is the critical value determined by the significance level α, N_1, and N_2 and is computed outside of MPC.

***t*-test.** This distribution test checks whether the means of two populations S_1 and S_2 for the attribute $\mathtt{attr}$ are equal, when the variances are not known. Parties provide the means $\bar{x}_1$ and $\bar{x}_2$, and the variances s_1^2 and s_2^2. The test passes if:

$$\bar{x}_1 - \bar{x}_2 / \sqrt{s_1^2/N_1 + s_2^2/N_2} \overset{?}{\leq} T_{\alpha,\mathrm{df}}$$

where $T_{\alpha,\mathrm{df}}$ is the critical value determined by the significance level α and the degrees of freedom, which are defined as follows.

$$\mathrm{df} = \frac{\left(s_1^2/N_1 + s_2^2/N_2\right)^2}{\left(s_1^2/N_1\right)^2/(N_1-1) + \left(s_2^2/N_2\right)^2/(N_2-1)},$$

The value $T_{\alpha,\mathrm{df}}$ is computed in MPC using a lookup table for df. In our implementation, the lookup table for df ranges from 1 to 100. When $\mathrm{df} > 100$, $T_{\alpha,\mathrm{df}}$ is approximated by 1.645.

***F*-test.** This distribution test checks whether the variances of populations S_1 and S_2 for the attribute $\mathtt{attr}$ are equal. Parties provide the variance s_1^2 and s_2^2, and the test passes if:

$$s_1^2 / s_2^2 \overset{?}{\leq} T_{\alpha,N_1,N_2}$$

where T_{α,N_1,N_2} is determined by the significance level α, N_1, and N_2 and is computed outside of MPC.

(One-dimensional) Pearson's χ^2-test. This distribution test checks whether the attribute $\mathtt{attr}$ of population S (which can be a joint dataset) follows a public distribution. Parties provide the histogram $\overrightarrow{\mathsf{count}}$ of their joint dataset over the attribute $\mathtt{attr}$ which has D buckets, and the test passes if:

$$\textstyle\sum_{j=1}^{D}(\mathsf{count}[j] - N\mathsf{p}_j)^2/(N\mathsf{p}_j) \overset{?}{\leq} T_{\alpha,D},$$

where N is the number of entries and p_j is the probability mass for the j-th bucket of the public distribution $\overrightarrow{\mathsf{p}} = (\mathsf{p}_1, \ldots, \mathsf{p}_D)$. The critical value $T_{\alpha,D}$ is determined by the significance level α and D and is computed outside of MPC.

3.2.3 Subsampling with malicious security

HOLMES allows distribution tests to be performed on a *random* subset of the dataset, which is decided after the data has been loaded to IZK and MPC. The random subset is chosen using a pseudorandom function, with a seed that comes from a coin toss protocol among the t parties [51]. Though this might sacrifice accuracy, it boosts efficiency and allows more tests to be performed with a given computational budget. We leave as an open direction identifying applications where the subsampling is beneficial, as discussed in §6.

3.3 Multidimensional Distribution Tests

We now discuss the setting where we want to test the distribution over multiple attributes (i.e., dimensions). Particularly, we want to test if the distribution of a dataset is close to a public distribution (e.g., a balanced distribution where different groups are represented appropriately) using Pearson's χ^2-test. Note that in this case, the number of buckets is $\prod_{j=1}^{\mathsf{d}} D_i$ where D_i is the number of buckets of the i-th attribute.

Baseline: multidimensional bucketing. We can naturally extend the one-dimensional Pearson's χ^2-test by creating multidimensional buckets. In particular, given the histogram $\overrightarrow{\mathsf{count}}$ over the attributes $(\mathtt{attr}_1, \cdots, \mathtt{attr}_{\mathsf{d}})$, the test checks if:

$$\textstyle\sum_{j=1}^{D_1 \cdot D_2 \cdot \ldots \cdot D_{\mathsf{d}}}(\mathsf{count}[j] - N\mathsf{p}_j)^2/(N\mathsf{p}_j) \overset{?}{\leq} T_{\alpha,D},$$

where N is size of population S, D_j is the number of distinct buckets of the j-th attribute $\mathtt{attr}_j$, $D = \prod_{i=1}^{\mathsf{d}} D_i$, and p_j is the probability mass for the j-th bucket of the public distribution $\overrightarrow{\mathsf{p}} = (\mathsf{p}_1, \ldots, \mathsf{p}_D)$. The critical value $T_{\alpha,D}$ is determined outside of MPC by the significance level α and the number of buckets D. We illustrate this test in Fig. 4.

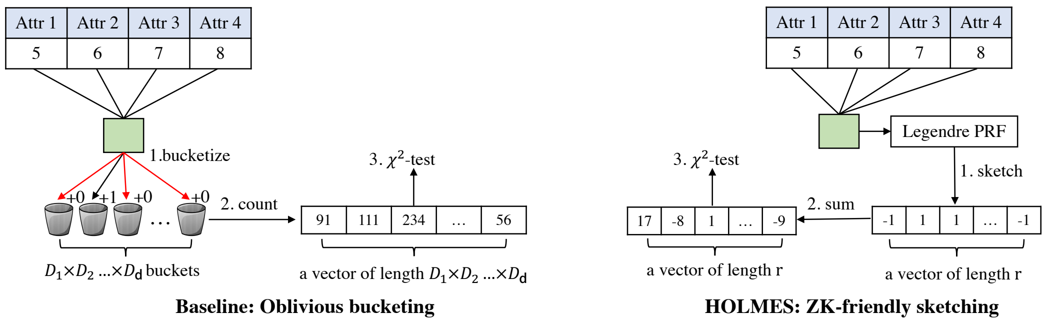
Cost analysis of the baseline. This baseline becomes impractical when the number of buckets D is high.

- **IZK cost:** A multidimensional histogram with D buckets is computed obliviously as in Appendix F. This requires an arithmetic circuit of size $O(N \cdot (D + \mathsf{d}\ell))$, where each bucket contains ranges of size at most 2^ℓ, and becomes impractical when D is large.
- **MPC cost:** The MPC performs the final computation for Pearson's χ^2-test, which involves the histogram $\overrightarrow{\mathsf{count}}$ of length D, which has been verified in IZK. The cost in MPC is $O(D \cdot \mathsf{cost}_{\div})$ operations and one comparison, where $\mathsf{cost}_{\div}$ is the cost of division in MPC.

The linear growth with respect to D is discouraging. In our experiments in §4.4.3, performing distribution testing over four attributes—age, jobs, marital status, and education—results in $D = 37{,}500$ and takes 10^5 seconds to compute.

New test: unnormalized χ^2-test. HOLMES uses another test for goodness-of-fit, called unnormalized χ^2-test inspired by the work of Arias-Castro, Pelletier, and Saligrama [53]. This test has a more complicated critical value, but it requires no divisions. The test checks if:

$$\textstyle\sum_{j=1}^{D_1 \cdot D_2 \cdot \ldots \cdot D_{\mathsf{d}}}(\mathsf{count}[j] - N\mathsf{p}_j)^2 \overset{?}{\leq} T_{\alpha,N,\mathsf{p}_1,\mathsf{p}_2,\ldots,\mathsf{p}_D},$$



Attr 1
Attr 2
Attr 3
Attr 4
5
6
7
8
1.bucketize
+0
+1
+0
+0
2. count
3. χ^2-test
91
111
234
...
56
$D_1 \times D_2 \ldots \times D_d$ buckets
a vector of length $D_1 \times D_2 \ldots \times D_d$
Baseline: Oblivious bucketing
Legendre PRF
1. sketch
2. sum
17
-8
1
-9
-1
a vector of length r
HOLMES: ZK-friendly sketching

Figure 4: Two methods of multidimensional distribution testing.

where the critical value $T_{\alpha,N,\mathsf{p}_1,\mathsf{p}_2,\ldots,\mathsf{p}_D}$ is computed from a variant of the generalized χ^2 distribution with parameters $(\mathsf{p}_1,\ldots,\mathsf{p}_D)$ [54–56]. We provide the details of the statistical test in Appendix G. Since the parameters are public, $T_{\alpha,N,\mathsf{p}_1,\mathsf{p}_2,\ldots,\mathsf{p}_D}$ is computed outside MPC.

Workflow in HOLMES. Naively computing the unnormalized χ^2-test has the same IZK cost as the baseline, and hence it is still prohibitively expensive for large D. We use sketching and pseudorandom functions (PRFs) to reduce this cost. The overall workflow for computing the unnormalized χ^2-test is also illustrated in Fig. 4. We show why our approach approximates well the unnormalized χ^2-test in §3.3.1.

- **PRF keys:** After the input-loading in IZK and MPC, parties run a coin toss protocol to sample r keys for the Legendre PRF, denoted by $\mathsf{k}_1,\mathsf{k}_2,\ldots,\mathsf{k}_r$. These PRF keys will be used to produce a pseudorandom matrix consisting of values in $\{-1,1\}$ for the random linear projection. We expand upon our sketching using random linear projections in §3.3.1. Details about the Legendre PRF can be found in Appendix H.
- **IZK sketching:** Each party $\mathcal{P}_i$ whose dataset is involved in the computation proves the following in IZK:
  1. computation of the linear index o (i.e., the index of value 1 in the one-hot encoding) of each data point for the attributes $(\mathsf{attr}_1,\ldots,\mathsf{attr}_\mathsf{d})$as described in Appendix F.4;
  2. computation of $\mathrm{PRF}_{\mathsf{k}_j}(o)$ for each $j\in[r]$ and the linear index o of each data point. For the k-th data point, this result in a r-vector $\vec{u}_k$ consisting of elements in $\{-1,1\}$;
  3. computation of $\overrightarrow{\mathsf{sum}}_i$ as $\sum_{k=1}^{N_i}\vec{u}_k$, where N_i is the dataset size of $\mathcal{P}_i$.
- **MPC finishing touches:** Similarly to the other tests, the computation in MPC is lightweight:
  1. compute $\overrightarrow{\mathsf{sum}}=\sum_{i=1}^{t}\overrightarrow{\mathsf{sum}}_i$, where t is the number of parties whose dataset is involved in the test;
  2. check if $\sum_{v=1}^{r}(\mathsf{sum}[v]-\mathsf{q}_v)^2\overset{?}{\leq} r\cdot T_{\alpha,N,\mathsf{p}_1,\mathsf{p}_2,\ldots,\mathsf{p}_D}$.

  The vector $\vec{\mathsf{q}}$ is computed outside of MPC as follows: Let $\mathbf{R}\in\mathbb{F}_p^{r\times D}$ be a matrix such that $\mathbf{R}[i][j]=\mathrm{PRF}_{\mathsf{k}_i}(j)$, then $\vec{\mathsf{q}}=N\cdot\mathbf{R}\cdot\vec{\mathsf{p}}$.

Cost analysis of HOLMES's approach. The IZK requires an arithmetic circuit of size $O(N\cdot r\cdot\mathsf{cost}_{\mathrm{PRF}})$ for a joint dataset of size N, where $\mathsf{cost}_{\mathrm{PRF}}$ is the cost of a PRF evaluation in IZK. The cost in MPC is $O(t+r)$ operations in $\mathbb{F}_p$. The cost of computing $\vec{\mathsf{q}}$ is $O(D\cdot r\cdot\mathsf{cost}_{\mathrm{PRF}})$ local operations. However, this computation is public, so its cost is negligible. In our experiments §4.4.2, HOLMES's approach can be 10^4 times faster than the baseline.

3.3.1 Why sketching works?

We now explain why HOLMES's approach approximates well the unnormalized χ^2-test.

Approximating the unnormalized χ^2-test. Observe that the unnormalized χ^2-test compares the Euclidean distance of $\overrightarrow{\mathsf{count}}$ and $N\vec{\mathsf{p}}$, $\mathsf{dist}(\overrightarrow{\mathsf{count}},N\vec{\mathsf{p}})$, to the value $T_{\alpha,N,\mathsf{p}_1,\mathsf{p}_2,\ldots,\mathsf{p}_D}$. From well-known results in statistics [34, 57] which we review in Appendix D, the Euclidean distance of two D-dimensional vectors $\vec{x}$ and $\vec{y}$, $\mathsf{dist}(\vec{x},\vec{y})$, can be approximated by $\mathsf{dist}(\mathbf{R}\cdot\vec{x},\mathbf{R}\cdot\vec{y})/r$ where $\mathbf{R}\cdot\vec{x}$ and $\mathbf{R}$ is a $r\times D$ matrix with entries (pseudo-)randomly chosen from $\{-1,1\}$ and r is sufficiently large, but independent of D. Hence, for suitable r

$$\mathsf{dist}(\overrightarrow{\mathsf{count}},N\vec{\mathsf{p}})\approx\mathsf{dist}(\mathbf{R}\cdot\overrightarrow{\mathsf{count}},\vec{\mathsf{q}})/r\ .$$

Proving that $\overrightarrow{\mathsf{sum}}=\mathbf{R}\cdot\overrightarrow{\mathsf{count}}$. The histogram $\overrightarrow{\mathsf{count}}$ stores the number of elements in each bucket. Hence, if $\vec{\sigma}_k$ is the one-hot encoding of the k-th value in the dataset, then

$$\overrightarrow{\mathsf{count}}=\textstyle\sum_{k=1}^{N}\vec{\sigma}_k\ .$$

Using this equality, it follows that

$$\mathbf{R}\cdot\overrightarrow{\mathsf{count}}=\mathbf{R}\cdot\textstyle\sum_{k=1}^{N}\vec{\sigma}_k=\sum_{k=1}^{N}\mathbf{R}\cdot\vec{\sigma}_k\ .$$

Since $\vec{\sigma}_k$ is a one-hot encoding, it contains a single element of value 1. Let o_k be the index of the element of value 1; we call o_k the linear index of $\vec{\sigma}_k$. Then,

$$\mathbf{R}\cdot\vec{\sigma}_k=\mathbf{R}[o_k],$$

where $\mathbf{R}[o_k]$ is the o_k-th column of the matrix $\mathbf{R}$. In our sketching, the matrix $\mathbf{R}$ is pseudorandom, and each element $\mathbf{R}[i][j]$

is equal to $\text{PRF}_{\mathsf{k}_i}(j)$, i.e., the evaluation on input j of a pseudorandom function with key k_i. Hence,

$$\mathbf{R}[o_k] = (\text{PRF}_{\mathsf{k}_1}(o_k), \ldots, \text{PRF}_{\mathsf{k}_r}(o_k)) \; ,$$

which is by definition equal to $\vec{u}_k$. Overall, since $\overrightarrow{\mathsf{sum}} = \sum_{k=1}^{N} \vec{u}_k$, it holds that

$$\overrightarrow{\mathsf{sum}} = \mathbf{R} \cdot \overrightarrow{\mathsf{count}} \; .$$

Choice of parameter *r*. In our implementation, we choose $r = 200$, which empirically results in 1.1 approximation factor. We discuss this choice in Appendix D.

3.3.2 Choosing an IZK-friendly PRF

Why use a PRF? The naive solution for our sketching is to sample $\mathbf{R}$ with random elements, without the special structure related to PRF evaluations. In this case, for the k-th data point, party $\mathcal{P}_i$ provides the computation $\mathbf{R}[o_k]$ obliviously, i.e., performs a lookup on the D columns of the *random matrix* $\mathbf{R}$. Using a linear scan, this requires an arithmetic circuit of size $O(N \cdot r \cdot D)$ in IZK. In our IZK sketching, the matrix $\mathbf{R}$ is produced from pseudorandom functions and this allows computing $\mathbf{R}[o_k]$ directly, without a linear scan. Hence, for a dataset to size N, $\mathcal{P}_i$ produces a proof that $\overrightarrow{\mathsf{sum}}_i$ was computed correctly with $N \cdot r$ PRF evaluations. When the PRF evaluation cost in IZK is small, our solution is more efficient.

Our choice: Legendre PRF. A concern with PRF evaluations in IZK is that the cost for common PRFs (e.g., SHA-256) is prohibitive. Thus, new ZK-friendly PRFs have been developed, e.g., Rescue [36] and Poseidon [35]. In our ZK-friendly sketching, we identify Legendre PRF [37–40] as the most suitable choice. Recall that in the sketching algorithm, the output of each PRF evaluation is an element in $\{-1, 1\}$. Legendre PRF, whose output is the Legendre symbol of a value modulo a prime, already has this property. In contrast, for other PRFs we need to extract these bits from a longer output, which incurs extra cost. We provide details about the Legendre PRF and compare its cost with other PRFs in Appendix H.

4 Implementation and Evaluation

In this section, we present and discuss the evaluation results of HOLMES, which answer the following questions:

- How well do HOLMES's distribution tests hold up against corruptions to both simulated and real-world data? (§4.3)
- How does HOLMES compare to the baselines, as well as alternative efficient system implementations? (§4.4)
- What is the overhead of HOLMES on real-world datasets? What contributes to this overhead? (§4.4.3)

4.1 Setup

We run our experiments on AWS c5.9xlarge instances, each with 36 cores and 72 GB memory. Each party has its own c5.9xlarge instance. We limit each instance's bandwidth to 2 Gbps and add a round-trip latency of 20 ms. We standardize data inputs across all protocols as field elements in $\mathbb{F}_p$ where $p = 2^{62} - 2^{16} + 1$. Text labels of an attribute are mapped and converted to field elements in $\mathbb{F}_p$. A d-dimensional input is formed as a vector of d field elements in $\mathbb{F}_p$, where the k-th vector entry represents the range bucket that the data point falls into for the k-th attribute. Decimals arising from divisions are stored in fixed-point representation (§3.2.1), where we multiply the operand by 10^2 to achieve a precision up to two decimal places by default. This fixed-point accuracy can be easily changed anytime by the parties for their use case.

General parameters for all setups include the statistical security parameter $\lambda = 30$, computational security parameter $\kappa = 128$, the input size, and the fixed-point accuracy.

HOLMES and the baselines are implemented using state-of-the-art cryptographic libraries, as follows.

HOLMES. We use QuickSilver [24] due to the lower prover overhead for IZK. The version of QuickSilver we use has integrated the latest techniques in Silver [58]. The number of concurrent threads run in a single prover-verifier protocol is defaulted to 32 to maximize multithreading. The input form that we use is the `IntFp` datatype, which represents a field element in $\mathbb{F}_p$.

We use SCALE-MAMBA [23] and MP-SPDZ [22, 59] for MPC, where the Low Gear protocol in MP-SPDZ is used for the offline phase of MPC and SCALE-MAMBA is used for the online phase. They are the state-of-the-art MPC protocols for arithmetic computations over large prime fields with a dishonest and malicious majority. For MP-SPDZ and SCALE-MAMBA, we use the Full Threshold Linear Secret Sharing Scheme with prime set to $p = 2^{62} - 2^{16} + 1$. Furthermore, we compile our circuits with the `sint` bit length limited to 32 bits, statistical security parameter $\lambda = 30$, and prime modulus size limited to 64 bits.

We compare HOLMES with three baselines–generic MPC, pairwise generic 2PC, NIZK/SNARK–to quantify the efficiency advantages. We now describe the setup of these systems.

Baseline 1: Generic MPC. The baseline runs HOLMES's MPC setup in entirety, using SCALE-MAMBA and MP-SPDZ. The data are only loaded once, and since there is no other auxiliary protocol there is no need for the consistency check.

Baseline 2: Pairwise 2PC. Each pair of parties runs a 2PC with the same setup as the generic MPC baseline using SCALE-MAMBA and MP-SPDZ. We instantiate a single party to host and execute 2PC protocols to all other parties on concurrent threads. Each individual 2PC protocol is run on a separate network port.

Baseline 3: NIZK & SNARK. We use a state-of-the-art NIZK and SNARK system, Spartan [25], with low communication overhead and small verification time. To utilize data-parallel circuit uniformity in Spartan, we copy the subcircuits of the distribution tests for each data entry. Furthermore, we parallelize verifications of NIZK/SNARK proofs on concurrent threads. We adapt the 62-bit field with modulus $p = 2^{62} - 2^{16} + 1$.

4.2 Artifacts

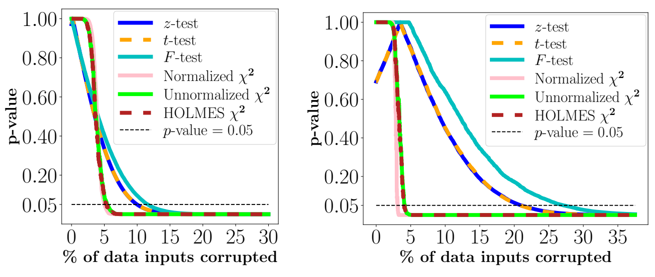
We released HOLMES in an open-sourced anonymous repository in GitHub. The implementation consists of three parts:

- **Compute engine:** The original QuickSilver is not compatible with many efficient MPC protocols because it works on a special prime field.[4] We perform an extensive search for a prime with low Hamming weight that is compatible with such MPC preprocessing protocols, and we settle to $p = 2^{62} - 2^{16} + 1$. We contribute a fork of EMP-ZK, called EMP-ZK-HOLMES[5], which includes a highly tuned, specialized implementation for modular reduction and learning-parity-with-noise (LPN) map for this prime.
- **Distribution tests:** We implement distribution tests for range, histogram, mean, variance, trimmed mean, z-test, t-test, F-test, and χ^2-test, including both oblivious bucketing and our ZK-friendly sketching. The codebase also includes integration tests, unit tests, and individual benchmarks.[6]
- **Examples and benchmarks:** We assemble distribution tests for the baselines and three real-world datasets (described in §4.4.3) and benchmark their performance. We also include accuracy evaluations for HOLMES' statistical tests against corruptions to simulated and real-world data. Finally, we provide a QuickSilver-to-SCALE-MAMBA source-to-source compiler and an online-only SCALE-MAMBA for ease of benchmarking with the baselines.

4.3 Accuracy evaluation

We show how HOLMES performs in face of specific corruption scenarios. Our goal is to better understand the fraction of data that needs to be corrupted for a distribution test to fail. Our statistical testing suite for the accuracy evaluation includes the z-test, t-test, F-test, naive normalized χ^2-test, naive unnormalized χ^2-test, and HOLMES's ZK-friendly sketching χ^2-test. We evaluate the accuracy on both simulated and real datasets.

General setup. We start with a dataset of positive integer values. We randomly divide the dataset into two equal parts; one part—which we call the "corrupted dataset"—is the dataset controlled by the adversary, and the other is an honest dataset. Our corruption model is as follows: at each iteration, we randomly select a data point from the corrupted dataset; we then modify, depending on the test, its value by 1, assuming that it remains within the acceptable bounds. For t and z-tests, we add 1 to the value, for F and χ^2-test we choose to either increase or decrease by 1. Note that this is a minimal amount of corruption per iteration. For instance, for the χ^2-tests, this corresponds to moving the data point to the next or previous bucket. The two populations in each test refer to the corrupted and the honest dataset [7].



(a) Simulated dataset (b) Real (marketing) dataset

Figure 5: Accuracy of distribution tests after corrupting the dataset

We compute the test statistic (i.e., the value before the final comparison with T_α) via the formulas of §3.2.2 and §3.3. We then calculate the p-value using inverse cumulative probability functions[8]. Typically, a test fails if the p-value is less than 0.05, which corresponds to a significance level $\alpha = 0.95$.

Simulated dataset. We simulate a dataset of randomly sampled values from $\mathcal{N}(\mu, \sigma^2)$, the normal distribution with mean μ and variance σ^2. We sample 40000 entries with $\mu = 17$ and $\sigma = 10$ to prevent field integer overflow and underflow while ensuring a diverse set of values in the dataset. For our χ^2-tests, we establish a histogram with 35 buckets $([0,1],[1,2],\ldots,[35,\infty))$ to avoid excessive outliers at the boundary buckets (i.e. $[0,1]$ and $[35,\infty)$), and ensure there are no empty buckets. For the other tests, we use the parameters $\mu = 17$ and $\sigma = 10$ and we calculate $\bar{x}_2$ and s_2^2 using the honest dataset. We graph the statistical p-values after corruptions in Fig. 5(a).

Real dataset. We use a bank marketing dataset [43, 44], which we describe further and time in our evaluation of HOLMES on real-world datasets in §4.4.3. We perform the z-test, t-test, and F-test over the telemarketing call duration attribute. We perform the χ^2-tests (normalized, unnormalized

[4] QuickSilver is restricted to a Mersenne prime $p = 2^{61} - 1$. However, this prime is not compatible with MPC preprocessing protocols based on ring learning-with-error (LWE) and would force the MPC to choose preprocessing protocols based on oblivious transfer, which are slower.

[5] https://github.com/holmes-inputcheck/emp-zk

[6] https://github.com/holmes-inputcheck/holmes-library

[7] In χ^2-test, where there is a single population, we use the entire dataset to compute the public distribution $\vec{p}$ and we ignore the honest dataset.

[8] For the unnormalized χ^2-test with or without ZK-friendly sketching, we compute a lookup table for the inverse cumulative probability functions as outlined in Appendix G For the lookup table, we use 5000 random samples.

with and without our sketching) over the attributes age, job, educational level, and marital status. We graph the statistical p-values after corruptions in Fig. 5(b).

Results. For the simulated dataset, the z-test and the t-test, both of which test the mean between two populations, fail at around 10% corruptions and they follow the same trend as a function of the fraction of corruptions. The normalized, unnormalized, and HOLMES χ^2-tests follow the same trend, which confirms the accuracy of HOLMES χ^2-test. The χ^2-tests fail when approximately 5% of the dataset is corrupted, much faster than all other statistical tests. Finally, the F-test is the most robust to corruptions and fails at about 13% corruptions.

For the marketing dataset, the z-test and the t-test have the same trend. However, now there might be an initial increase in p-values, depending on the random split of the dataset into honest and corrupted, since we use the honest dataset to compute the underlying properties. For instance, if the honest dataset has initially larger mean than the corrupted, increasing the values of data points initially leads to an increase of the p-value. Hence, the p-values drop at an offsetted percentage of corrupted data entries for the z-test, t-test, and F-test. Similarly to the simulated dataset, the χ^2-tests drop off at the same rate and lead to test failures at approximately the same point.

4.4 Evaluation discussion

We evaluate the overheads for our distribution tests. We show the efficiency for the histogram, mean, variance, and trimmed mean of HOLMES and its comparison with the generic MPC baseline with $t = 2$ parties in Fig. 7.Our results are as follows: the histogram in HOLMES is about 5–11× more efficient than generic MPC; the mean and variance are 2–3× more efficient; and the trimmed mean is about 5–10× more efficient.

Next, in §4.4.1 we compare the overhead of range checks and ZK-friendly sketching with alternative applicable systems; these are the two most expensive gadgets supported in HOLMES. Range check is the main source of overhead for many HOLMES gadgets (§3.2.1, Algorithm 3, Algorithm 4), while ZK-friendly sketching is the bottleneck for HOLMES's χ^2-test. Later on, in §4.4.2 we depict the drastic overhead reduction of HOLMES's χ^2-test over the naive χ^2-test. Finally, in §4.4.3 we show that HOLMES performs efficiently in practical settings compared to our generic MPC baseline by modeling distribution test workflows on real-world datasets.

4.4.1 Comparison of HOLMES with the Baselines

We evaluate range checks and ZK-friendly sketching for number of parties $t = 2, 6, 10$ and input size per party $N_{\mathsf{ind}} = 100$k, 200k, 500k.

Range and HOLMES's χ^2-test Setup. For the range check, we vary the sizes of the range $[a,b]$ (i.e., $b-a$) as 2^ℓ for $\ell \in \{8,12,16,20,24\}$. We run the range check algorithm (Algorithm 1)with these inputs and parameters. Our results are listed in Tab. 1. In the ZK-friendly sketching, for simplicity, we assume that all attributes take the same number of distinct values. We consider the number of attributes $\mathsf{d} \in \{2,4\}$, and vary the buckets per attribute as $D_0 = \cdots = D_{\mathsf{d}} \in \{5,10,50\}$. We perform the IZK check described in §3.3, i.e, for each data point, we compute its linear index (Appendix F.4) and feed it into $r = 200$ Legendre PRFs with polynomial degree 3, and quadratic nonresidue $7 \in \mathbb{F}_p$; the $r = 200$ unique keys are generated from a random oracle based on SHA-256. We run the Legendre PRF algorithm (Algorithm 6)with these inputs and parameters. Our results are listed in Tab. 3.

Baseline 1: Generic MPC . The overhead grows quadratically in the number of parties and linearly in the input size; hence, generic MPC is the slowest baseline in our comparison. The baseline is 10–256× and 35–198× slower than QuickSilver for the range check and the ZK-friendly sketching, respectively.

Baseline 2: Pairwise 2PC . Pairwise 2PC provides higher throughput than generic MPC due to the parallelization of the offline phases and online phases. It also has lower latency than generic MPC, since 2PC reduces the creation of authenticated shares from the entire t-party combined dataset of size $t \cdot N_{\mathsf{ind}}$ to the two-party combined dataset of size $2 \cdot N_{\mathsf{ind}}$. MP-SPDZ still needs to preprocess $t-1$ different inputs of size $2 \cdot N_{\mathsf{ind}}$, and as a result, the preprocessing phase contributes to most of the overhead. In sum, the overhead grows linearly to the number of parties and linearly to the input size.

Pairwise 2PC is faster than generic MPC, but slower than HOLMES. Pairwise 2PC is 4–32× slower for the range check and 13–36× slower for the ZK-friendly sketching than QuickSilver. For 10 parties, 2PC is 18× and 13× faster than generic MPC for the range check and the ZK-friendly sketching, respectively. For 6 parties, these numbers become 10× and 8×, depicting a speedup factor of around $O(t)$ over generic MPC.

Baseline 3: NIZK & SNARK . Spartan$_{\mathrm{NIZK}}$ is the second fastest system behind HOLMES. NIZK and SNARK systems scale well for a large number of parties, since each party only generates a single proof for its dataset, and parties can concurrently verify other parties' proofs on multiple cores. Thus, this approach has overhead that remains relatively constant to the number of parties (up to the number of threads in our machine). However, the overhead to prove dense circuits with lots of constraints is large relative to arithmetic-based IZKs, and still grows linearly to the input size; for instance, in the two-party case, Spartan$_{\mathrm{NIZK}}$ is 2.4–16× slower for the range check and 4–45× slower for the ZK-friendly sketching than QuickSilver. Hence, for extremely large and dense circuits (e.g. ZK-friendly sketching), we extrapolate our small input size experiments to larger input sizes.

For a small number of parties, Spartan$_{\mathrm{NIZK}}$ is quite inefficient and has speeds comparable to pairwise 2PC. However, for larger numbers of parties, e.g., 10, it is approximately 4–5× and 3–4× faster for the range check and the ZK-friendly

Table 1: Overhead of range checking on different protocols with varying input sizes. For a range $[a,b]$, the range size is $b-a$. $\text{Spartan}_{\text{SNARK}}$ is significantly slower than $\text{Spartan}_{\text{NIZK}}$, so we omit benchmarks for input sizes 200k and 500k.

		Number of parties = 2					**Number of parties = 6**					**Number of parties = 10**				
N_{ind}	**Range Size**	$\mathbf{2^{8}}$	$\mathbf{2^{12}}$	$\mathbf{2^{16}}$	$\mathbf{2^{20}}$	$\mathbf{2^{24}}$	$\mathbf{2^{8}}$	$\mathbf{2^{12}}$	$\mathbf{2^{16}}$	$\mathbf{2^{20}}$	$\mathbf{2^{24}}$	$\mathbf{2^{8}}$	$\mathbf{2^{12}}$	$\mathbf{2^{16}}$	$\mathbf{2^{20}}$	$\mathbf{2^{24}}$
100k	**QuickSilver**	**3.4s**	**3.7s**	**4.2s**	**4.7s**	**4.9s**	**17.0s**	**18.6s**	**21.0s**	**23.4s**	**24.5s**	**30.6s**	**33.4s**	**37.8s**	**42.1s**	**44.1s**
	Paired 2PC	35.0s	51.9s	73.2s	85.9s	103.4s	88.5s	133.1s	173.3s	216.4s	267.9s	145.3s	217.0s	288.7s	359.3s	432.3s
	MPC	35.0s	51.9s	73.2s	85.9s	103.4s	894.7s	1309s	1739s	2134s	2568s	2566s	3795s	5024s	6241s	7603s
	Spartan$_{\text{NIZK}}$	30.4s	48.5s	59.9s	77.1s	108.0s	30.8s	49.0s	60.3s	77.6s	108.5s	31.5s	49.2s	60.6s	77.9s	109.8s
	Spartan$_{\text{SNARK}}$	275.5s	526.5s	526.9s	528.0s	1070s	276.7s	528.7s	529.2s	530.2s	1073s	278.4s	530.9s	531.4s	532.4s	1076s
200k	**QuickSilver**	**4.7s**	**5.6s**	**6.2s**	**7.2s**	**7.8s**	**23.7s**	**27.8s**	**30.9s**	**36.1s**	**38.9s**	**42.6s**	**50.1s**	**55.6s**	**65.1s**	**70.1s**
	Paired 2PC	71.0s	104.5s	139.4s	173.7s	207.9s	178.9s	265.9s	345.2s	439.4s	523.3s	293.4s	433.3s	578.9s	716.8s	864.4s
	MPC	71.0s	104.5s	139.4s	173.7s	207.9s	1757s	2682s	3423s	4278s	5303s	5092s	7576s	10142s	12691s	15064s
	Spartan$_{\text{NIZK}}$	66.2s	108.8s	130.9s	153.6s	208.2s	66.1s	108.6s	131.0s	154.2s	208.5s	66.4s	107.5s	132.1s	152.5s	210s
500k	**QuickSilver**	**9.3s**	**10.3s**	**12.9s**	**14.8s**	**16.5s**	**46.7s**	**51.3s**	**64.4s**	**74.3s**	**82.4s**	**84.1s**	**92.4s**	**115.9s**	**133.7s**	**148.3s**
	Paired 2PC	177.3s	263.7s	361.1s	441.1s	530.9s	443.3s	663.7s	872.9s	1081.6s	1294.4s	729.9s	1092s	1446s	1797s	2142s
	MPC	177.3s	263.7s	361.1s	441.1s	530.9s	4370s	6543s	8799s	10707s	13405s	12792s	19411s	25299s	31333s	38063s
	Spartan$_{\text{NIZK}}$	155.4s	244.3s	299.7s	356.5s	483.0s	155.5s	244.7s	300.1s	356.8s	483.3s	158.1s	242.6s	305.8s	357.3s	487.0s

sketching, respectively, than pairwise 2PC. $\text{Spartan}_{\text{NIZK}}$ is 1–3$\times$ slower for the range check and 4–5$\times$ slower for the ZK-friendly sketching than QuickSilver.

$\text{Spartan}_{\text{SNARK}}$ has succinct proof size and verification time but with massive prover overhead. Even for the most complicated benchmark, the ZK-friendly sketching with $N_{\text{ind}} = 100\text{k}$, verification time is $\sim$1–2s. However, the computational overhead for the same circuit in $\text{Spartan}_{\text{SNARK}}$ is around 9$\times$ larger than $\text{Spartan}_{\text{NIZK}}$ and 2$\times$ slower than generic MPC.

4.4.2 Cost of multidimensional tests

We now measure the efficiency of HOLMES's multidimensional χ^2-test. Namely, we test HOLMES's ZK-friendly sketching against the oblivious bucketing approach, which we call the "naive χ^2-test". For both HOLMES χ^2-test and naive χ^2-test, we run the benchmarks entirely in our choice of IZK: QuickSilver. We vary the number of attributes as $\mathsf{d} \in \{2,3,4,5\}$, and the buckets per attributes as $D_0 = \cdots = D_{\mathsf{d}} \in \{5,10,15,20,25\}$. Since the naive approach is extremely expensive, we were able to run the experiment only on a small scale, so extrapolate our small scale experiments to a larger scale. As shown Fig. 7(g) and Fig. 7(h),HOLMES and the naive χ^2-test have drastically different growth patterns. The overhead of the naive χ^2-test grows exponentially in the number of buckets per attribute. The overhead of HOLMES multidimensional χ^2-test is no longer dominated by the number of multidimensional buckets as in the naive χ^2-tests; instead, the new overhead of our sketching approach, $O(N \cdot r \cdot \mathsf{cost}_{\mathsf{PRF}})$, is now dominated by the input size and the number of PRF keys. Based on our experiments, we find that HOLMES's sketching approach greatly improves the efficiency of multidimensional tests when the number of attributes and distinct values per attribute are large. For instance, when $\mathsf{d} = 4$ and $D_i = 25$, we observe an efficiency increase of around $10^4\times$.

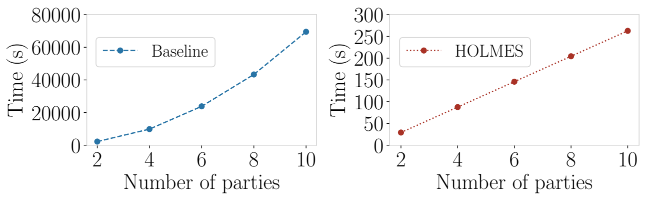


Figure 6: Marketing dataset overhead in generic MPC and HOLMES.

Table 2: Cost breakdown for the marketing dataset (two-party).

Number of entries (41188×2)	82376
Number of attributes	21
Total time	29.19 s
Average time per entry	0.35 ms
Loading the data to IZK	0.27 s
Range tests for all attributes	5.40 s
Histogram and χ^2 test on *age*	0.76 s
Multidimensional χ^2 test on *age, job, marital status, education*	22.3 s
Mean, variance, and t test on *call duration*	0.12 s
Consistency check	< 0.01 s

4.4.3 Evaluation on real-world dataset

We apply HOLMES and the generic MPC baseline to a real-world bank marketing dataset and study the overhead. We include two additional real-world examples of dataset testing workflows and study their overheads in Appendix I.

In our experiment, we vary the number of parties from 2 to 10 to see how HOLMES and the generic MPC baseline scale with more parties. In secure collaborative learning with more parties we expect to have access to more data, so we assume that each party provides the same amount of data N_{ind}; when there are t parties, there are $t \cdot N_{\text{ind}}$ data. For example, for our marketing data, we assume that each party provides $N_{\text{ind}} = 41188$ entries of data. When there are 10 parties, the entire distribution testing would be over $N_{\text{ind}} \cdot t = 411880$ entries.

Table 3: Overhead of the ZK-friendly sketching with varying parameters. For simplicity, we assume that we have the same number of buckets for each attribute. The dimension setup refers to [number of attributes, buckets of each attribute]. Spartan$_{\text{SNARK}}$ is significantly slower than Spartan$_{\text{NIZK}}$, so we omit benchmarks for input sizes 200k and 500k.

		Number of parties = 2			**Number of parties = 6**			**Number of parties = 10**		
N_{ind}	**Dimension setup**	**[1,10]**	**[4,10]**	**[4,50]**	**[1,10]**	**[4,10]**	**[4,50]**	**[1,10]**	**[4,10]**	**[4,50]**
100k	**QuickSilver**	**57.3s**	**57.9s**	**58.5s**	**286.8s**	**289.9s**	**292.6s**	**516.3s**	**521.9s**	**526.7s**
	Paired 2PC	2064s	2074s	2074s	4340s	4385s	4354s	6927s	6955s	6966s
	MPC	2064s	2074s	2074s	36333s	36330s	36522s	91387s	90982s	90648s
	Spartan$_{\text{NIZK}}$	2602s	2610s	2613s	2521s	2559s	2579s	2667s	2651s	2689s
	Spartan$_{\text{SNARK}}$	20752s	20985s	20212s	20754s	20988s	20214s	20757s	20990s	20217s
200k	**QuickSilver**	**114.1s**	**111.8s**	**111.9s**	**570.6s**	**558.8s**	**559.6s**	**1027s**	**1006s**	**1007s**
	Paired 2PC	4086s	4098s	4106s	8734s	8737s	8708s	13915s	13788s	13935s
	MPC	4086s	4098s	4106s	72600s	72934s	72863s	182917s	184209s	183890s
	Spartan$_{\text{NIZK}}$	5065s	5087s	5090s	5152s	5146s	5127s	5135s	5140s	5153s
500k	**QuickSilver**	**280.5s**	**277.8s**	**276.9s**	**1402s**	**1389s**	**1385s**	**2524s**	**2500s**	**2492s**
	Paired 2PC	10136s	10163s	10201s	21914s	21794s	21773s	34820s	34941s	34840s
	MPC	10136s	10163s	10201s	181400s	182747s	181889s	456853s	456366s	463618s
	Spartan$_{\text{NIZK}}$	10002s	9989s	10152s	10501s	10388s	10441s	10549s	10417s	10521s

Marketing dataset workflow. The dataset [43, 44] consists of telemarketing records for financial products. It includes client profile and call records. We choose a distribution test workflow that fits the common use case of untrusted banks who wish to jointly train a model to predict the success of the campaign. Before training they want to ensure that the dataset has a balanced number of customers from different backgrounds. Therefore, banks may consider the following tests:

- Pearson's (histogram) χ^2-test over age grouped into the buckets 10–19, 20–29, $\cdots$, 90–99 to ensure that the dataset distribution is similar to the national census age distribution,
- Pearson's (multidimensional) χ^2-test over age, job, educational level, and marital status to check if the dataset is balanced across customers with different backgrounds,
- t-test over the telemarketing call duration to check whether their telemarketing records are similar enough to train a model together, and
- range checks for all attributes.

Results. We present our results in Fig. 6(a) and Fig. 6(b). HOLMES's approach outperforms the generic MPC baseline by 77–264×. The overhead gap widens as the number of parties increases. As expected, HOLMES's overhead grows linearly to t, while the baseline overhead grows quadratically to t. We also present the cost breakdown in Tab. 2. We see that the range and multidimensional tests contribute to a large portion of the overhead compared to all other tests. The consistency check between IZK and MPC has a small overhead.

5 Related Work

We summarize related works and explain their connection to HOLMES.

Secure multiparty computation frameworks. A rich body of works propose MPC protocols [49, 60, 61] for malicious adversaries and dishonest majority, with SPDZ [62–64] and authenticated garbling [65–68] being the state-of-the-art. HOLMES uses SPDZ since it is more suitable for arithmetic computation that is used for secure collaborative learning.

Zero-knowledge proofs. Zero-knowledge proofs [69] enable a party to prove a statement without leaking any information. Constructing practical ZK has gained much attention, especially since succinct non-interactive proofs [25, 70–73] have been used in blockchains. New protocols for interactive zero-knowledge proofs based on silent OT [24, 74–79] are currently being studied for their efficiency. Although not currently ready for implementation, a subset of these protocols known as line-point zero-knowledge (LPZK) [80, 81] promise greater flexibility with primes and smaller prover and verification overhead than QuickSilver. HOLMES in the future can be extended to these newer protocols.

Statistics and range checks. There have been works [82, 83] whose goal is to perform statistical tests privately using MPC. In contrast to our protocol, they mostly focus on the two-party case and consider different threat models. Also, range checks [71, 84, 85] are frequently used to limit the effect of misreported values in secure computation. As an example, Prio [84] (or Prio+ [85]) is a system that aggregates statistics over multiple clients who wish to preserve the confidentiality of their individual data but relies on the existence of non-colluding semi-honest servers. HOLMES offers security guarantees even with a dishonest majority.

Secure collaborative computation systems. Multiple works build systems for data analytics and machine learning against malicious adversaries [18, 19, 86–99], but they do not address the issue of corrupted input datasets or group fairness, which is often left as an open question. We envision an integration of HOLMES to secure collaborative computation systems as an efficient method for distribution testing.

6 Conclusion

We first discuss some challenges that HOLMES does not solve and we identify several exciting directions for future work. Finally, we conclude with a summary of our contributions.

Identifying necessary tests. HOLMES enables parties to perform distribution tests tailored to their use case. It does not, however, decide what the necessary tests are. The parties have to specify tests depending on their application. We are not aware of a systematic approach that identifies the necessary tests for a specific application, such as measuring the data quality or identifying bias in clinical trials. Since this is relevant even without any privacy considerations, this question is orthogonal to the goal of HOLMES: privately computing distribution tests. A compelling future direction is to combine HOLMES's rich class of distribution tests with a systematic approach to identify necessary tests in practical applications.

Privacy leakage from distribution tests. Any distribution test leaks one-bit information – whether the test passed or failed – which leads to potential attacks. For instance, assume that an organization wants to check that the mean value of another organization's dataset is a close to a specific value, e.g., in a medical study we might want to prove that the mean efficacy of a drug is 0.9. A malicious organization, who is not supposed to know this mean can recover it by requesting multiple distributions tests. For example, in the medical study example, the adversary can ask whether the mean is 0.01, 0.02, etc., until the distribution test succeeds.

Potential mitigations for this problem include having a curated list of allowed distribution tests (e.g., proving that the mean of the age of a population is below 150 does not leak any sensitive information), or enforce a rate limit on the tests. An interesting direction for future research is to devise attacks that exploit this leakage and identify mitigations in specific applications.

Improving HOLMES's efficiency. Even though HOLMES outperforms the baselines based on state-of-the-art systems in our benchmarks, there are specific cases that other systems have a small efficiency advantage. For example, in the range check test with $\ell = 8$ and $N_{\mathsf{ind}} = 100\text{k}$, at 11 parties or more we expect $\text{Spartan}_{\text{NIZK}}$ (31.5s) to be faster than QuickSilver (34s) since Spartan's overhead remains relatively constant to a growing number of parties. We leave as an open question how to build a system that is more efficient in all settings.

New protocols for interactive zero-knowledge proofs based on silent OT [24, 74–79] are currently being studied for their efficiency. Although not ready for implementation, a subset of these protocols known as line-point zero-knowledge (LPZK) [80, 81] promises greater flexibility with primes and smaller prover overhead than QuickSilver. HOLMES in the future can be extended to these newer protocols.

Additionally, HOLMES supports distribution tests performed on a random subset of the dataset to boost efficiency. When the datasets are sufficiently large, intuitively subsampling should not affect accuracy. However, we are not aware of specific applications where this feature can be tested.

Adversarial machine learning. HOLMES is a useful tool for identifying bias in datasets used in machine learning training without compromising their privacy. Even though we have experimented with the accuracy of HOLMES in specific adversarial scenarios, our protocol does not offer any formal guarantees. In the realm of adversarial machine learning, data poisoning shows that it is possible to corrupt a machine learning model by using datasets practically indistinguishable from the honest ones. As a mitigation, robust statistics [100–102] focuses on statistics that are resilient to any corrupted input distribution. A fascinating future direction is to augment HOLMES with robust statistics that not only detect bias, but can reduce its effect in the final machine learning application.

In conclusion, we present HOLMES, a protocol for performing distribution testing in secure collaborative learning efficiently. The core of HOLMES consists of two contributions:

- a new hybrid protocol that integrates MPC, IZK, and a lightweight consistency check for distribution testing, which is concretely more efficient than non-trivial baselines, and
- a novel, efficient multidimensional distribution testing procedure that utilizes sketching and pseudorandom functions to avoid the severe penalty of oblivious computation.

These two tools significantly improve the performance of distribution testing. Efficient support for distribution testing can be seen as the first step towards detecting different types of incorrect (or even malicious) inputs for secure computation in general, which is an essential for practical secure collaborative learning. HOLMES is open-sourced in GitHub: https://github.com/holmes-inputcheck/.

Acknowledgements

We thank the anonymous reviewers and our shepherd for their helpful feedback. This work is supported by NSF CISE Expeditions Award CCF-1730628, NSF CAREER 1943347, and gifts from the Alibaba, Amazon Web Services, Ant Group, Astronomer, Ericsson, Facebook, Futurewei, Google, IBM, Intel, Lacework, Microsoft, Nexla, Nvidia, Samsung, Scotiabank, Splunk, and VMware.

References

[1] Karen Hao. *AI is sending people to jail—and getting it wrong*. https://www.technologyreview.com/2019/01/21/137783/algorithms-criminal-justice-ai/.

[2] Ganes Kesari. *AI Can Now Detect Depression From Your Voice, And It's Twice As Accurate As Human Practitioners*. https://bit.ly/32HdcUQ.

[3] *Ellipsis Health*. https://www.ellipsishealth.com/.

[4] *Qbtech*. https://www.qbtech.com/.

[5] McKinsey Global Institute. *Tackling bias in artificial intelligence (and in humans)*. https://mck.co/3Ge65B8.

[6] Zhe Yu, Joymallya Chakraborty, and Tim Menzies. "FairBalance: Improving Machine Learning Fairness on Multiple Sensitive Attributes With Data Balancing". In: *Arxiv:2107.08310*.

[7] Angelina Wang, Arvind Narayanan, and Olga Russakovsky. "REVISE: A tool for measuring and mitigating bias in visual datasets". In: *ECCV '20*.

[8] Faisal Kamiran and Toon Calders. "Data preprocessing techniques for classification without discrimination". In: *Knowledge and Information Systems* 33.1 (2012), pp. 1–33.

[9] Thomas Davidson, Debasmita Bhattacharya, and Ingmar Weber. "Racial Bias in Hate Speech and Abusive Language Detection Datasets". In: *Workshop on Abusive Language Online '19*.

[10] Nature Communications Editorial. "Data sharing and the future of science". In: *Nature Communications '18*.

[11] Heather A. Piwowar and Todd J. Vision. "Data reuse and the open data citation advantage". In: *PeerJ '13*.

[12] Milton Packer. "Data sharing in medical research". In: *British Medical Journal '18*.

[13] Liina Kamm, Dan Bogdanov, Sven Laur, and Jaak Vilo. "A new way to protect privacy in large-scale genome-wide association studies". In: *Bioinformatics '13*.

[14] Emmanuel A Abbe, Amir E Khandani, and Andrew W Lo. "Privacy-preserving methods for sharing financial risk exposures". In: *American Economic Review '12*.

[15] *General Data Protection Regulation*. https://gdpr-info.eu/.

[16] *Rights related to automated decision making including profiling*. https://bit.ly/3ALUJ6m.

[17] *Assessing Data Quality for Healthcare Systems Data Used in Clinical Research*. https://dcricollab.dcri.duke.edu/sites/NIHKR/KR/Assessing-data-quality_V1%200.pdf.

[18] Rishabh Poddar, Sukrit Kalra, Avishay Yanai, Ryan Deng, Raluca Ada Popa, and Joseph M. Hellerstein. "Senate: A Maliciously-Secure MPC Platform for Collaborative Analytics". In: *SEC '20*.

[19] Wenting Zheng, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. "Helen: Maliciously Secure Coopetitive Learning for Linear Models". In: *S&P '19*.

[20] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. "Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning". In: *PETS '21*.

[21] Sameer Wagh, Divya Gupta, and Nishanth Chandran. "SecureNN: 3-Party Secure Computation for Neural Network Training". In: *PETS '19*.

[22] *Multi-Protocol SPDZ*. https://github.com/data61/MP-SPDZ.

[23] *SCALE and MAMBA*. https://github.com/KULeuven-COSIC/SCALE-MAMBA.

[24] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. "QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field". In: *CCS '21*.

[25] Srinath Setty. "Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup". In: *CRYPTO '20*.

[26] Oded Goldreich. "Towards a theory of software protection". In: *CRYPTO '86*.

[27] Oded Goldreich. "Towards a theory of software protection and simulation by oblivious RAMs". In: *STOC '87*.

[28] Oded Goldreich and Rafail Ostrovsky. "Software protection and simulation on oblivious RAMs". In: *JACM '96*.

[29] Benny Pinkas and Tzachy Reinman. "Oblivious RAM revisited". In: *CRYPTO '10*.

[30] Elaine Shi, T-H Hubert Chan, Emil Stefanov, and Mingfei Li. "Oblivious RAM with O $(\log N^3)$ worst-case cost". In: *ASIACRYPT '11*.

[31] Emil Stefanov, Elaine Shi, and Dawn Song. "Towards practical oblivious RAM". In: *NDSS '12*.

[32] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. "Path ORAM: An extremely simple oblivious RAM protocol". In: *CCS '13*.

[33] Craig Gentry, Kenny A Goldman, Shai Halevi, Charanjit Julta, Mariana Raykova, and Daniel Wichs. "Optimizing ORAM and using it efficiently for secure computation". In: *PETS '13*.

[34] William B Johnson and Joram Lindenstrauss. "Extensions of Lipschitz mappings into a Hilbert space". In: *Contemporary mathematics '84*.

[35] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. “POSEIDON: A New Hash Function for Zero-Knowledge Proof System”. In: *SEC ’21*.

[36] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. “Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols”. In: *FSE ’20*.

[37] Ivan Bjerre Damgård. “On the randomness of Legendre and Jacobi sequences”. In: *CRYPTO ’88*.

[38] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. “MPC-friendly symmetric key primitives”. In: *CCS ’16*.

[39] Dmitry Khovratovich. “Key recovery attacks on the Legendre PRFs within the birthday bound”. In: *IACR ePrint 2019/862*.

[40] Alexander May and Floyd Zweydinger. “Legendre PRF (Multiple) Key Attacks and the Power of Preprocessing”. In: *IACR ePrint 2021/645*.

[41] Mark N. Wegman and J. Lawrence Carter. “New hash functions and their use in authentication and set equality”. In: *JCSS ’81*.

[42] Moni Naor and Moti Yung. “Universal one-way hash functions and their cryptographic applications”. In: *STOC ’89*.

[43] Sérgio Moro, Paulo Cortez, and Paulo Ritaa. “A data-driven approach to predict the success of bank telemarketing”. In: *Decision Support Systems ’14*.

[44] *Bank Marketing Data Set*. https://archive.ics.uci.edu/ml/datasets/Bank+Marketing.

[45] Beata Strack, Jonathan P. DeShazo, Chris Gennings, Juan L. Olmo, Sebastian Ventura, Krzysztof J. Cios, and John N. Clore. “Impact of HbA1c measurement on hospital readmission rates: Analysis of 70,000 clinical database patient records”. In: *BioMed Research International ’14*.

[46] *Diabetes 130-US hospitals for years 1999-2008 Data Set*. https://archive.ics.uci.edu/ml/datasets/Diabetes+130-US+hospitals+for+years+1999-2008.

[47] *Real Time Advertiser’s Auction*. https://www.kaggle.com/saurav9786/real-time-advertisers-auction.

[48] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. “Universally composable two-party and multi-party secure computation”. In: *STOC ’02*.

[49] Oded Goldreich, Silvio Micali, and Avi Wigderson. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”. In: *STOC ’87*.

[50] Yehuda Lindel. “How to simulate it: A tutorial on the simulation proof technique”. In: *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*. 2017.

[51] Manuel Blum. “Coin flipping by telephone a protocol for solving impossible problems”. In: *ACM SIGACT News ’83*.

[52] Carl Friedrich Gauss. “Theoria combinationis obsevationum erroribus minimis obnoxiae”. In: *Carl Friedrich Gauss Werke*. 1823.

[53] Ery Arias-Castro, Bruno Pelletier, and Venkatesh Saligrama. “Remember the curse of dimensionality: the case of goodness-of-fit testing in arbitrary dimension”. In: *Journal of Nonparametric Statistics ’18*.

[54] Robert B Davies. “Algorithm AS 155: The distribution of a linear combination of χ^2 random variables”. In: *Applied Statistics* (1980), pp. 323–333.

[55] J Sheil and I O’Muircheartaigh. “Algorithm AS 106: The distribution of non-negative quadratic forms in normal variables”. In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 26.1 (1977), pp. 92–98.

[56] Jean-Pierre Imhof. “Computing the distribution of quadratic forms in normal variables”. In: *Biometrika* 48.3/4 (1961), pp. 419–426.

[57] Dimitris Achlioptas. “Database-Friendly Random Projections: Johnson-Lindenstrauss with Binary Coins”. In: *Journal of Computer and System Sciences ’03*.

[58] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. “Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes”. In: *CRYPTO ’21*.

[59] Marcel Keller. “MP-SPDZ: A Versatile Framework for Multi-Party Computation”. In: *CCS ’20*.

[60] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. “Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation”. In: *STOC ’88*.

[61] Andrew Chi-Chih Yao. “Protocols for Secure Computations”. In: *FOCS ’82*.

[62] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. “Multiparty Computation from Somewhat Homomorphic Encryption”. In: *CRYPTO ’12*.

[63] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. “Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits”. In: *ESORICS ’13*.

[64] Marcel Keller, Valerio Pastro, and Dragos Rotaru. “Overdrive: Making SPDZ Great Again”. In: *EUROCRYPT ’18*.

[65] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. “Global-Scale Secure Multiparty Computation”. In: *CCS ’17*.

[66] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. “Low Cost Constant Round MPC Combining BMR and Oblivious Transfer”. In: *ASIACRYPT ’17*.

[67] Kang Yang, Xiao Wang, and Jiang Zhang. “More Efficient MPC from Improved Triple Generation and Authenticated Garbling”. In: *CCS ’20*.

[68] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. “Ferret: Fast Extension for Correlated OT with small communication”. In: *CCS ’20*.

[69] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. “The Knowledge Complexity of Interactive Proof-Systems”. In: *STOC ’85*.

[70] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. “Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting”. In: *EUROCRYPT ’16*.

[71] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. “Bulletproofs: Short Proofs for Confidential Transactions and More”. In: *S&P ’18*.

[72] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. “Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation”. In: *CRYPTO ’19*.

[73] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. “Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof”. In: *S&P ’20*.

[74] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. “Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning”. In: *SEC ’21*.

[75] Chenkai Weng, Kang Yang, Xiao Wang, and Jonathan Katz. “Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits”. In: *S&P ’21*.

[76] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. “Line-Point Zero Knowledge and Its Applications”. In: *ITC ’21*.

[77] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. “Mac’n’Cheese: Zero-Knowledge Proofs for Boolean and Arithmetic Circuits with Nested Disjunctions”. In: *CRYPTO ’21*.

[78] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. “Compressing Vector OLE”. In: *CCS ’18*.

[79] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. “Efficient Pseudorandom Correlation Generators: Silent OT Extension and More”. In: *CRYPTO ’19*.

[80] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. “Line-Point Zero Knowledge and Its Applications”. In: *ITC ’21*.

[81] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. *Improving Line-Point Zero Knowledge: Two Multiplications for the Price of One*. Cryptology ePrint Archive, Paper 2022/552. https://eprint.iacr.org/2022/552. 2022. URL: https://eprint.iacr.org/2022/552.

[82] Alexandr Andoni, Tal Malkin, and Negev Shekel Nosatzki. “Two party distribution testing: Communication and security”. In: *arXiv preprint arXiv:1811.04065* (2018).

[83] Varun Narayanan, Manoj Mishra, and Vinod M Prabhakaran. “Private two-terminal hypothesis testing”. In: *ISIT ’20*.

[84] Henry Corrigan-Gibbs and Dan Boneh. “Prio: Private, robust, and scalable computation of aggregate statistics”. In: *NSDI ’17*.

[85] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. “Prio+: Privacy Preserving Aggregate Statistics via Boolean Shares”. In: *IACR ePrint 2021/576*.

[86] Payman Mohassel and Yupeng Zhang. “SecureML: A system for scalable privacy-preserving machine learning”. In: *S&P ’17*.

[87] Melissa Chase, Ran Gilad-Bachrach, Kim Laine, Kristin E. Lauter, and Peter Rindal. “Private Collaborative Neural Network Learning”. In: *IACR ePrint 2017/762*.

[88] Irene Giacomelli, Somesh Jha, Marc Joye, C David Page, and Kyonghwan Yoon. “Privacy-preserving ridge regression with only linearly-homomorphic encryption”. In: *ACNS ’18*.

[89] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. “Oblivious neural network predictions via MiniONN transformations”. In: *CCS ’17*.

[90] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. “GAZELLE: A low latency framework for secure neural network inference”. In: *SEC ’18*.

[91] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. “XONN: XNOR-based oblivious deep neural network inference”. In: *SEC ’19*.

[92] Valerie Chen, Valerio Pastro, and Mariana Raykova. “Secure Computation for Machine Learning With SPDZ”. In: *NeurIPS ’18*.

[93] Anselme Tueno, Florian Kerschbaum, and Stefan Katzenbeisser. “Private Evaluation of Decision Trees using Sublinear Cost”. In: *PETS ’19*.

[94] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. “Privacy-preserving ridge regression on hundreds of millions of records”. In: *S&P ’13*.

[95] 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: *PETS ’17*.

[96] Hao Chen, Miran Kim, Ilya Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. “Maliciously secure matrix multiplication with applications to private deep learning”. In: *ASIACRYPT ’20*.

[97] Wenting Zheng, Ryan Deng, Weikeng Chen, Raluca Ada Popa, Aurojit Panda, and Ion Stoica. “Cerebro: A Platform for Multi-Party Cryptographic Collaborative Learning”. In: *SEC ’21*.

[98] Christopher A. Choquette-Choo, Natalie Dullerud, Adam Dziedzic, Yunxiang Zhang, Somesh Jha, Nicolas Papernot, and Xiao Wang. “CaPC Learning: Confidential and Private Collaborative Learning”. In: *Arxiv:2102.05188*. 2021.

[99] Mark Abspoel, Daniel Escudero, and Nikolaj Volgushev. “Secure training of decision trees with continuous attributes”. In: *PETS ’21*.

[100] Frank R Hampel, Elvezio M Ronchetti, Peter J Rousseeuw, and Werner A Stahel. *Robust statistics: The approach based on influence functions*. Vol. 196. John Wiley & Sons, 2011.

[101] Peter J Huber. *Robust statistics*. Vol. 523. John Wiley & Sons, 2004.

[102] Ricardo A Maronna, R Douglas Martin, Victor J Yohai, and Matías Salibián-Barrera. *Robust statistics: Theory and methods (with R)*. John Wiley & Sons, 2019.

[103] David Evans, Vladimir Kolesnikov, and Mike Rosulek. “Defining Multi-Party Computation”. In: *A Pragmatic Introduction to Secure Multi-Party Computation*. 2018.

[104] Oded Goldreich. *Foundations of Cryptography*. Cambridge university press Cambridge, 2004.

[105] Suresh Venkatasubramanian and Qiushi Wang. “The Johnson-Lindenstrauss transform: An empirical study”. In: *Workshop on Algorithm Engineering and Experiments (ALENEX) ’11*.

[106] Ward Beullens, Tim Beyne, Aleksei Udovenko, and Giuseppe Vitto. “Cryptanalysis of the Legendre PRF and generalizations”. In: *FSE ’20*.

[107] Novak Kaluđerović, Thorsten Kleinjung, and Dušan Kostić. “Cryptanalysis of the generalised Legendre pseudorandom function”. In: *ANTS ’20*.

[108] Michael O. Rabin. “Probabilistic algorithms in finite fields”. In: *SIAM Journal on Computing ’80*.

[109] Carl Friedrich Gauss. *Carl Friedrich Gauss’ Untersuchungen uber hohere Arithmetik*. 1889.

[110] Sunil K Chebolu and Ján Mináč. “Counting irreducible polynomials over finite fields using the inclusion-exclusion principle”. In: *Mathematics Magazine ’11*.

[111] Cunsheng Ding, Tor Hesseseth, and Weijuan Shan. “On the linear complexity of Legendre sequences”. In: *IEEE TIT ’98*.

[112] Viktória Tóth. “Collision and avalanche effect in families of pseudorandom binary sequences”. In: *Periodica Mathematica Hungarica ’07*.

[113] Christian Mauduit and András Sárközy. “On finite pseudorandom binary sequences I: Measure of pseudorandomness, the Legendre symbol”. In: *Acta Arithmetica ’97*.

[114] *Legendre pseudo-random function*. https://legendreprf.org/.

[115] István András Seres, Máté Horváth, and Péter Burcsi. “The Legendre pseudorandom function as a multivariate quadratic cryptosystem: Security and applications”. In: *IACR ePrint 2021/182*.

[116] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlos. “A sparse Johnson-Lindenstrauss transform”. In: *STOC ’10*.

[117] Daniel M Kane and Jelani Nelson. “A derandomized sparse Johnson-Lindenstrauss transform”. In: *ArXiv:1006.3585* (2010).

[118] Henry Corrigan-Gibbs and Dmitry Kogan. “The discrete-logarithm problem with preprocessing”. In: *EUROCRYPT ’18*.

[119] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. “Zero-Knowledge Contingent Payments Revisited: Attacks and Payments for Services”. In: *CCS ’17*.

A Definitions

We present some basic cryptographic definitions. First, we define secure computation in the UC model [48, 103] and of zero-knowledge protocols. Then, we give the definitions of additive secret sharing and homomorphic commitments.

Definition 3 (MPC (See [104])). *We define* multiparty secure computation (MPC) *(with abort) using the ideal/real-world paradigm in the UC-framework.*

Real world: *Let Π be a t-party protocol among $(\mathcal{P}_1,\ldots,\mathcal{P}_t)$ computing a t-party function $\mathcal{F}:(\{0,1\}^*)^t \to (\{0,1\}^*)^t$, let $C \subseteq [t]$ denote the set of indices of the corrupted parties and Z be an interactive Turing machine representing the environment. At setup, Z provides $(1^\lambda,\mathbf{x}_i)$ to each party $\mathcal{P}_i$ and $(C,\{\mathbf{x}_i\}_{i\in C},\mathsf{aux})$ to the adversary $\mathcal{A}$, where aux denotes some auxiliary input. The real-world execution on input $\vec{\mathbf{x}}=(\mathbf{x}_1,\ldots,\mathbf{x}_t)$ and security parameter λ with respect to $(Z,\mathcal{A},C)$, denoted by $\mathsf{real}_{\Pi,C,\mathcal{A}(\mathsf{aux})}(\vec{\mathbf{x}},\lambda)$, is the output of Z resulting from the protocol interaction.*

Ideal world: *Let $\mathcal{F}:(\{0,1\}^*)^t \to (\{0,1\}^*)^t$ be a t-party function and let Z be an interactive Turing machine, representing the environment, that at setup provides $(1^\lambda,\mathbf{x}_i)$ to each party $\mathcal{P}_i$ and $(C,\{\mathbf{x}_i\}_{i\in C},\mathsf{aux})$ to the simulator Sim, where $C\subseteq[t]$ represents the set of corrupted parties and aux is some auxiliary input. The ideal-world execution on input $\vec{\mathbf{x}}=(\mathbf{x}_1,\ldots,\mathbf{x}_t)$ and security parameter λ with respect to (Z,Sim,C), denoted by $\mathsf{ideal}_{\mathcal{F},C,\mathsf{Sim}(\mathsf{aux})}(\vec{\mathbf{x}},\lambda)$, is the output of Z from the following process:*

- Trusted party receives inputs: *Each honest party $\mathcal{P}_i$ sends to the trusted party its input $\mathbf{x}_i$. The simulator Sim may send to the trusted party arbitrary inputs corresponding to the corrupted parties $\{\mathbf{x}_i\}_{i\in C}$.*
- Trusted party computes outputs: *The trusted party computes $(y_1,\ldots,y_t):=\mathcal{F}(\mathbf{x}_1,\ldots,\mathbf{x}_t)$ and sends $\{y_i\}_{i\in C}$ to the simulator Sim.*
- Simulator decides to abort or not: *The simulator Sim responds with abort or $\perp$.*
- Trusted party answers to honest parties: *If the trusted party received $\perp$, then it sends $\perp$ to each honest party $\mathcal{P}_i$. Otherwise, it sends y_i to each party $\mathcal{P}_i$.*
- Environment computes output: *The environment Z receives the outputs of the honest parties and an arbitrary function of Sim's view of the protocol. Finally, Z outputs a bit.*

An MPC protocol is secure if for every probabilistic polynomial-time adversary $\mathcal{A}$ there exists a simulator Sim such that the distributions $\mathsf{real}_{\Pi,C,\mathcal{A}(\mathsf{aux})}(\vec{\mathbf{x}},\lambda)\}_{\vec{\mathbf{x}}\in(\{0,1\}^*)^t,\lambda\in\mathbb{N}}$ and $\{\mathsf{ideal}_{\mathcal{F},C,\mathsf{Sim}(\mathsf{aux})}(\vec{\mathbf{x}},\lambda)\}_{\vec{\mathbf{x}}\in(\{0,1\}^*)^t,\lambda\in\mathbb{N}}$ are (computationally) indistinguishable.

Definition 4 (Zero-knowledge proofs (See [104])). *Let Π be a protocol between two parties, a prover $\mathbb{P}$ and a verifier $\mathbb{V}$. We say that Π is a zero-knowledge interactive argument for some language $\mathcal{L}=\{\mathbb{x}:$ there exists $\mathbb{w}$ s.t. $(\mathbb{w},\mathbb{x})\in\mathcal{R}\}$, corresponding to a relation $\mathcal{R}$, if it satisfies the following properties:*

- ***Completeness:*** *For all $(\mathbb{w},\mathbb{x})\in\mathcal{R}$, if $\mathbb{P}$ has input $(\mathbb{w},\mathbb{x})$ and $\mathbb{V}$ has input $\mathbb{x}$, then the verifier's output after running Π is 1.*
- ***Soundness:*** *For all $\mathbb{x}\notin\mathcal{L}$, for all $\lambda\in\mathbb{N}$, and for all polynomial-time algorithms $\mathcal{A}$, on input $\mathbb{x}$, if $\mathcal{A}$ has input $(\mathbb{x},1^\lambda)$ and $\mathbb{V}$ has input $\mathbb{x}$, then the verifier's output after running Π is 0 except with negligible probability in λ.*
- ***Zero-knowledge:*** *For every probabilistic polynomial-time interactive machine $\mathbb{V}^*$, there exists an probabilistic polynomial-time algorithm Sim so that for every $x\in\mathcal{L}$ the transcript of the protocol between $\mathbb{V}^*$ and $\mathbb{P}$ on common input $\mathbb{x}$ and the output of Sim on input $\mathbb{x}$ are (computationally) indistinguishable.*

Definition 5. *A t-additive secret sharing over $\mathbb{F}_p$ for an element $x\in\mathbb{F}_p$ is a tuple $(\mathsf{share}_1(x),\ldots,\mathsf{share}_t(x))$ such that $\sum_{j=1}^{t}\mathsf{share}_j(x)=x$ where addition is performed over $\mathbb{F}_p$.*

Definition 6. *A* commitment scheme *is a tuple of polynomial-time probabilistic algorithms $\mathsf{CM}=(\mathsf{Setup},\mathsf{KeyGen})$ with the following syntax.*

- $\mathsf{Setup}(1^\lambda)\to\mathsf{pp}_{\mathsf{CM}}$*: samples public parameters given a security parameter and a message length.*
- $\mathsf{KeyGen}(\mathsf{pp}_{\mathsf{CM}})\to\mathsf{ck}$*: samples a commitment key.*
- $\mathsf{Com}_{\mathsf{ck}}(\mathbf{x};\rho)\to\mathsf{C}$*: computes commitment C for message $\mathbf{x}$ using key ck and randomness ρ. For ease of notation, we may ignore ρ and write $\mathsf{Com}_{\mathsf{ck}}(\mathbf{x})$.*

We require CM to satisfy the following binding and hiding properties.

- ***Binding:*** *If $\mathsf{pp}_{\mathsf{CM}}\leftarrow\mathsf{Setup}(1^\lambda)$ and $\mathsf{ck}\leftarrow\mathsf{KeyGen}(\mathsf{pp}_{\mathsf{CM}})$, then no probabilistic polynomial-time adversary $\mathcal{A}$ can find $(\mathsf{C},\mathbf{x}_0,\mathbf{x}_1,\rho_0,\rho_1)$ such that $\mathsf{C}=\mathsf{Com}_{\mathsf{ck}}(\mathbf{x}_0;\rho_0)$, $\mathsf{C}=\mathsf{Com}_{\mathsf{ck}}(\mathbf{x}_1;\rho_1)$, and $\mathbf{x}_0\neq\mathbf{x}_1$*
- ***Hiding:*** *If $\mathsf{pp}_{\mathsf{CM}}\leftarrow\mathsf{Setup}(1^\lambda)$ and $\mathsf{ck}\leftarrow\mathsf{KeyGen}(\mathsf{pp}_{\mathsf{CM}})$, then no probabilistic polynomial-time adversary $\mathcal{A}$ can distinguish between $\mathsf{Com}_{\mathsf{ck}}(\mathbf{x}_0;\rho)$ and $\mathsf{Com}_{\mathsf{ck}}(\mathbf{x}_1;\rho)$, where $\mathbf{x}_0,\mathbf{x}_1$ are chosen by $\mathcal{A}$ and ρ is random.*

A commitment scheme is (message) homomorphic *over $\mathbb{F}_p$ if for every $x_0,x_1,\alpha,\beta\in\mathbb{F}_p$ and randomness ρ_0,ρ_1, it holds that $\alpha\cdot\mathsf{Com}_{\mathsf{ck}}(x_0;\rho_0)+\beta\cdot\mathsf{Com}_{\mathsf{ck}}(x_1;\rho_1)=\mathsf{Com}_{\mathsf{ck}}(\alpha x_0+\beta x_1;\rho')$, where ρ' is a function of ρ_0 and ρ_1.*

B Consistency check

We present the formal definition of consistency checks.

Definition 7. *Let Π_{cc} be a t-party protocol between a prover $\mathbb{P}$ and $t-1$ verifiers $\mathbb{V}_1,\ldots,\mathbb{V}_{t-1}$. The prover $\mathbb{P}$ takes as input*

$(\mathbf{x}, \mathsf{aux}, \{\ell_{\text{mpc}}[j], \ell_{\text{izk}}[j]\}_{j\in[t]})$, *where* aux *is auxiliary information (e.g., commitment keys and randomness), and each verifier* $\mathbb{V}_j$ *takes as input* $(\ell_{\text{mpc}}[j], \ell_{\text{izk}}[j])$. *The protocol* Π_{cc} *is a* consistency check *(*CC*) if it satisfies the following properties:*

- ***Completeness:*** *If* $\ell_{\text{mpc}}[j] = \mathsf{load}_{\text{mpc}}(j, \mathbf{x})$ *and* $\ell_{\text{izk}}[j] = \mathsf{load}_{\text{izk}}(j, \mathbf{x})$ *for all* $j \in [t-1]$*, then the verifiers accept.*
- ***Soundness:*** *If there exists no* $\mathbf{x}$ *such that for all* $j \in [t-1]$ $\ell_{\text{mpc}}[j] = \mathsf{load}_{\text{mpc}}(j, \mathbf{x})$ *and* $\ell_{\text{izk}}[j] = \mathsf{load}_{\text{izk}}(j, \mathbf{x})$*, then for all* $\lambda \in \mathbb{N}$*, and for all polynomial-time algorithms* $\mathcal{A}$*, on input* $(\mathsf{aux}, \{\ell_{\text{mpc}}[j], \ell_{\text{izk}}[j]\}_{j\in[t-1]})$*, the verifiers reject except with negligible probability in* λ*.*
- ***Zero-knowledge:*** *For every probabilistic polynomial-time interactive machine* $\mathbb{V}^*$ *that plays the role of the verifiers, there exists a probabilistic polynomial-time algorithm* Sim *so that if* $\ell_{\text{mpc}}[j] = \mathsf{load}_{\text{mpc}}(j, \mathbf{x})$ *and* $\ell_{\text{izk}}[j] = \mathsf{load}_{\text{izk}}(j, \mathbf{x})$ *all* $j \in [t-1]$*, the transcript of the protocol between* $\mathbb{V}^*$ *and* $\mathbb{P}$ *and the output of* Sim *on input* $(\{\ell_{\text{mpc}}[j], \ell_{\text{izk}}[j]\}_{j\in[t-1]})$ *are (computationally) indistinguishable.*

We now prove Lemma 1.

Proof. The protocol Π_{cc} satisfies the following properties:

- **Completeness:** Since the commitment scheme is homomorphic, it holds that $\mathsf{Com}_{\mathsf{ck}_j}(r) + \sum_{k=1}^{N} \mathsf{Com}_{\mathsf{ck}_j}(x_k) \cdot \beta^k = \mathsf{Com}_{\mathsf{ck}_j}(r + \sum_{k=1}^{N} x_k \cdot \beta^k)$. Also, from the correctness guarantee of the MPC protocol, it holds that $\rho = r + \sum_{k=1}^{N} x_k \cdot \beta^k$. Hence, $\mathsf{Com}_{\mathsf{ck}_j}(r) + \sum_{k=1}^{N} \mathsf{Com}_{\mathsf{ck}_j}(x_k) \cdot \beta^k = \mathsf{Com}_{\mathsf{ck}_j}(\rho)$ and the verifiers accept, because of the completeness of the IZK protocol.
- **Soundness:** If there is no $\mathbf{x}$ such that for all $j \in [t-1]$, $\ell_{\text{mpc}}[j] = \mathsf{load}_{\text{mpc}}(j, \mathbf{x})$, then the MPC protocol in Step 3 outputs $\perp$. This is because of the guarantees of MPC with input-loading (Definition 1). Otherwise, there exists an $\mathbf{x}$ with $\ell_{\text{mpc}}[j] = \mathsf{load}_{\text{mpc}}(j, \mathbf{x})$ for all $j \in [t-1]$. Similarly, the values $\mathsf{load}_{\text{mpc}}(j, r)$ sent by the prover must correspond to the input-loading of a value r. From the guarantees of the MPC protocol, this implies that ρ, computed in Step 3, must equal to $r + \sum_{k=1}^{N} x_k \cdot \beta^k$ or the protocol aborts.
  Now, assume that there exists a $j \in [t-1]$ such that $\ell_{\text{izk}}[j] \neq \mathsf{load}_{\text{izk}}(j, \mathbf{x})$. From the input-loading property and the soundness of IZK, either $\ell_{\text{izk}}[j]$ is commitment to $\mathbf{x}' := (x'_1, \ldots, x'_N) \in \mathbb{F}_p^N$ or verifier $\mathbb{V}_j$ rejects. Similarly, the value sent from the prover in Step 1 corresponding to $\mathsf{load}_{\text{izk}}(j, r)$ is a commitment to a value r'.
  Then, from the homomorphic property of the commitment scheme, which implies that $\mathsf{Com}_{\mathsf{ck}_j}(r') + \sum_{k=1}^{N} \mathsf{Com}_{\mathsf{ck}_j}(x'_k) \cdot \beta^k = \mathsf{Com}_{\mathsf{ck}_j}(r' + \sum_{k=1}^{N} x'_k \cdot \beta^k)$, and Lemma 2, it holds that
  $$\Pr_{\beta}\left[\mathsf{Com}_{\mathsf{ck}_j}(r') + \sum_{k=1}^{N} \mathsf{Com}_{\mathsf{ck}_j}(x'_k) \cdot \beta^k \neq \mathsf{Com}_{\mathsf{ck}_j}(\rho)\right] \leq N/p.$$
  Lemma 2 (DeMillo-Lipton-Schwartz–Zippel lemma). *Let* $f(x)$ *be a non-zero polynomial of degree* N *in a prime field* $\mathbb{F}_p$*. Pick* $\beta \leftarrow\$\, \mathbb{F}_p$*. Then, we have* $\mathbf{Pr}[f(\beta) = 0] \leq N/p$*.*
  Hence, if p has size 2^λ, the verifiers reject except with negligible probability in λ.
- **Zero-knowledge:** The simulator Sim works as follows. First, it samples random values corresponding to $(\mathsf{load}_{\text{mpc}}(j, r), \mathsf{load}_{\text{izk}}(j, r))_{j\in[t]}$ in the real execution. Then, it samples $\rho \leftarrow\$\, \mathbb{F}_p$ and runs the MPC simulator $\mathsf{Sim}_{\text{mpc}}$ to produce the transcript for the computation of ρ. Finally, it runs $t-1$ copies of the IZK simulator $\mathsf{Sim}_{\text{izk}}$ to produce the transcript for the IZK proofs.
  The indistinguishability with the real execution follows from the fact that ρ is random in $\mathbb{F}_p$ and from the properties of $\mathsf{Sim}_{\text{izk}}$ and $\mathsf{Sim}_{\text{mpc}}$.

□

C Cost analysis of HOLMES's individual components

We describe the cost of distribution tests as follows: In IZK, the cost corresponds to the size of an arithmetic circuit over a field $\mathbb{F}_p$ for the IZK relation $\mathcal{R}$, i.e., outputs 1 if $(\mathbb{w}, \mathbb{x}) \in \mathcal{R}$ and 0 otherwise; in MPC, the cost measures the size of an arithmetic circuit over a field $\mathbb{F}_p$ that computes the MPC functionality $\mathcal{F}$.

IZK. The cost of range, histogram, mean, variance, and trimmed mean is as follows.

- **Range:** $O(N \log(b-a))$ operations for size N dataset.
- **Histogram:** $O(N \cdot (D + \max_{j\in[D]} \log(b_j - a_j)))$ operations for a dataset of size N.
- **Mean and variance:** $O(\log N)$ operations for a dataset of size N. This cost includes the range check to prove that $N \cdot \bar{x} \approx \sum_{j=1}^{N} x_j$ and/or $s^2 \approx \frac{N}{N-1}(\bar{y} - \bar{x}^2)$.
- **Trimmed mean:** $O(N \cdot \ell)$ operations for parameter $\theta < 2^\ell$ and a dataset of size N.

Computing the distribution tests of §3.2.2 for an individual dataset requires an additional range check in IZK. In the case of joint datasets, the MPC cost is as follows.

MPC. The computation of distribution tests uses basic statistics verified by IZK, so the cost is often independent of the size of the dataset N.

- ***z*-, *t*-, *F*-tests:** $O(1)$ operations and one comparison in $\mathbb{F}_p$.
- **Pearson's χ^2-test:** $O(D \cdot \mathsf{cost}_\div)$ operations and one comparison, where $\mathsf{cost}_\div$ is the cost of division in $\mathbb{F}_p$.

D Johnson-Lindenstrauss lemma

We state the Johnson-Lindenstrauss lemma, specialized in our setting. This lemma guarantees that our sketching is a good approximation of the unnormalized χ^2 statistic.

Lemma 3 (Johnson-Lindenstrauss lemma [57]). *Let* $k \geq \frac{4+2\beta}{\varepsilon^2/2+\varepsilon^3/3}$, $\mathbf{R}$ *be a random* $r \times D$ *matrix with uniform entries* *in $\{-1/\sqrt{r}, 1/\sqrt{r}\}$, then for any $\vec{x}, \vec{y} \in \mathbb{Z}_{2^\ell}^D$ it holds that with probability at least* $1 - 2^{-\beta}$

$$(1-\varepsilon) \cdot \mathsf{dist}(\vec{x}, \vec{y}) \leq \mathsf{dist}(\mathbf{R} \cdot \vec{x}, \mathbf{R} \cdot \vec{y}) \leq (1+\varepsilon) \cdot \mathsf{dist}(\vec{x}, \vec{y}) \; .$$

This theoretical result requires an extremely conservative parameter r, which prevents using it in practice. Venkatasubramanian and Wang [105] shows that empirically, in our setting, one can choose $r = 2/\varepsilon^2$. In our implementation, we choose $r = 200$ which results in error $\varepsilon = 1/10$.

E Security Proof of Theorem 1

Proof. To show that HOLMES as described in Fig. 3, securely computes $\mathcal{F}_{HOLMES}$ we have to define the non-uniform probabilistic polynomial-time simulator Sim and show that the outputs in the ideal world and the real world, which are defined as in Definition 3 are computationally indistinguishable.

Let $\mathsf{Sim}_{\text{mpc}}$ be the simulator of Π_{mpc} (Definition 3), $\mathsf{Sim}_{\text{izk}}$ be the simulator of Π_{izk} (Definition 4) and Sim_{cc} be the simulator of Π_{cc}. The simulator Sim proceeds as follows:

- *Inputs sent to the trusted party:* Run $\mathsf{Sim}_{\text{mpc}}$ to compute the inputs corresponding to the corrupted parties $\{\mathbf{x}_i\}_{i \in C}$.
- *Outputs received from the trusted party:* Receive $\{y_i\}_{i \in C}$ from the trusted party.
- *Simulator decides to abort or not:* The simulator computes its view as follows.
  1. **MPC input-loading:** Run $\mathsf{Sim}_{\text{mpc}}$ up to the point where the values $\ell_{\text{mpc}}[i, j]$ are computed.
  2. **IZK input-loading:** Run a copy of the algorithm $\mathsf{Sim}_{\text{izk}}$, for each pair $(\mathbb{P}, \mathbb{V}) = (\mathcal{P}_i, \mathcal{P}_j)$ with $i \notin C$ and $i \neq j$, up to the point where the values $\ell_{\text{izk}}[i, j]$ are computed. For each pair $(\mathbb{P}, \mathbb{V}) = (\mathcal{P}_i, \mathcal{P}_j)$ with $i \in C$ and $i \neq j$, simulate $\mathcal{P}_j$ as an honest verifier up to the point where the values $\ell_{\text{izk}}[i, j]$ are computed.
  3. **Distribution tests:** Run $\mathcal{A}$ with input $(\ell_{\text{mpc}}[i, j], \ell_{\text{izk}}[i, j])_{i \in [t] \setminus C, j \in C}$ to compute tests_j for $j \in C$.
  4. **Consistency check:** Run a copy of the algorithm Sim_{cc}, for each pair $(\mathbb{P}, \mathbb{V}) = (\mathcal{P}_i, \mathcal{P}_j)$ with $i \notin C$ and $i \neq j$. For each pair $(\mathbb{P}, \mathbb{V}) = (\mathcal{P}_i, \mathcal{P}_j)$ with $i \in C$ and $i \neq j$, simulate $\mathcal{P}_j$ as an honest verifier.
  5. **IZK protocol:** Continue the copies of the algorithm $\mathsf{Sim}_{\text{izk}}$ for the languages corresponding to each test in $\overrightarrow{\mathsf{tests}}$ as described in §3.2 and §3.3. For each pair $(\mathbb{P}, \mathbb{V}) = (\mathcal{P}_i, \mathcal{P}_j)$ with $i \in C$ and $i \neq j$, continue simulating $\mathcal{P}_j$ as an honest verifier.
  6. **MPC protocol:** Continue the algorithm $\mathsf{Sim}_{\text{mpc}}$ for the functionalities corresponding to each test in $\overrightarrow{\mathsf{tests}}$ as described in §3.2 and §3.3.

  If $\mathsf{Sim}_{\text{mpc}}$ decides to abort or if any consistency checks fail or any IZK protocol fails, then send $\perp$ to the trusted party.
- *Simulator sends view to environment:* Send the view computed in the previous step.

Now, we show that $\mathsf{real}_{\Pi, C, \mathcal{A}(\mathsf{aux})}(\vec{\mathbf{x}}, \lambda)\}_{\vec{\mathbf{x}} \in (\{0,1\}^*)^t, \lambda \in \mathbb{N}}$ and $\{\mathsf{ideal}_{\mathcal{F}, C, \mathsf{Sim}(\mathsf{aux})}(\vec{\mathbf{x}}, \lambda)\}_{\vec{\mathbf{x}} \in (\{0,1\}^*)^t, \lambda \in \mathbb{N}}$ are (computationally) indistinguishable. We construct a series of hybrid to move from the ideal-world execution to a real-world execution.

Hybrid 1: We replace Item 1 (**MPC input-loading**) and Item 6 (**MPC protocol**) with the transcript in the real execution of the protocol with adversary $\mathcal{A}$. Also, Sim sends $\perp$ to the trusted party if $\mathcal{A}$ outputs $\perp$ in the MPC protocol. All other steps of Sim remain the same.

The output of Hybrid 1 is indistinguishable from the output of Sim. First, Item 2 (**IZK input-loading**), Item 3 (**Distribution tests**), and Item 5 (**IZK protocol**) remain unchanged, hence their output distributions are the same. Also, Π_{mpc} is a secure MPC protocol, so the outputs of Item 1 (**MPC input-loading**) and Item 6 (**MPC protocol**) are indistinguishable from the transcript of the real execution of the MPC protocol. It remains to show that the outputs of Item 4 (**Consistency check**) in Sim and Hybrid 1 are indistinguishable.

For each pair $(\mathbb{P}, \mathbb{V}) = (\mathcal{P}_i, \mathcal{P}_j)$ with $i \notin C$ and $i \neq j$, indistiguishability follows from the properties of secure MPC protocols. If an attacker were able to distinguish between such transcripts of Sim_{cc} in Sim and Hybrid 1, then this attacker would distinguish between the real and simulated execution of the MPC protocol.

For each pair $(\mathbb{P}, \mathbb{V}) = (\mathcal{P}_i, \mathcal{P}_j)$ with $i \in C$ and $i \neq j$, the outputs of Sim and Hybrid 1 remain unchanged, and hence are indistinguishable.

Hybrid 2: We replace Item 2 (**IZK input-loading**) and Item 5 (**IZK protocol**) with the transcript in the real execution of the protocol with adversary $\mathcal{A}$. All other steps are as in Hybrid 1.

The output of Hybrid 2 is indistinguishable from the output of Hybrid 1. From the zero-knowledge property of the IZK protocol, for each pair $(\mathbb{P}, \mathbb{V}) = (\mathcal{P}_i, \mathcal{P}_j)$ with $i \notin C$ and $i \neq j$ it must be that the output of $\mathsf{Sim}_{\text{izk}}$ is indistinguishable from the transcript with respect to $\mathcal{A}$. Additionally, from the soundness property of the IZK protocol for each pair $(\mathbb{P}, \mathbb{V}) = (\mathcal{P}_i, \mathcal{P}_j)$ with $i \in C$ and $i \neq j$ it must be that either $\ell_{\text{izk}}[i, j] = \mathsf{load}_{\text{izk}}(j, \mathbf{x}_i)$ and $\mathbf{x}_i$ satisfies the conditions specified by $\overrightarrow{\mathsf{tests}}$, or the protocol aborts. Finally, we can show that the outputs of Sim_{cc} in

Hybrid 1 and Hybrid 2 are indistinguishable as before, using the zero-knowledge property of the IZK protocol.

Hybrid 3: We replace Item 4 (**Consistency check**) transcript in the real execution of the protocol with adversary $\mathcal{A}$. All other steps are as in Hybrid 2. The output of Hybrid 3, which corresponds to the real execution, is indistinguishable from the output of Hybrid 2. From the zero-knowledge property of the CC protocol, for each pair $(\mathbb{P}, \mathbb{V}) = (\mathcal{P}_i, \mathcal{P}_j)$ with $i \notin C$ and $i \neq j$ it must be that the output of Sim_{cc} is indistinguishable from the transcript with respect to $\mathcal{A}$. Additionally, from the soundness property of the CC protocol for each pair $(\mathbb{P}, \mathbb{V}) = (\mathcal{P}_i, \mathcal{P}_j)$ with $i \in C$ and $i \neq j$ it must be that either $\ell_{\text{izk}}[i,j] = \mathsf{load}_{\text{izk}}(j, \mathbf{x}_i)$ and $\ell_{\text{mpc}}[i,j] = \mathsf{load}_{\text{mpc}}(j, \mathbf{x}_i)$, or the protocol aborts. Finally, since the MPC protocol enables input-loading, it must be that the output is either $\mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_t)$ or $\perp$.

We remark that our definition of security guarantees *correctness* and *privacy*, as the definition of secure MPC (Definition 3). Correctness holds because the simulator outputs either $\perp$ or the correct output value, as computed by the trusted party, and its transcript is indistinguishable from the adversary's view. Security holds because the simulator has no access to the honest parties' inputs and again its transcript is indistinguishable from the adversary's view.

$\square$

F Pseudocode for IZK protocols

We describe various checks as a system of arithmetic operations over a field $\mathbb{F}_p$ with $|\mathbb{F}_p| = p$. These operations can be easily transformed in the typical computational models of ZK systems, such as R1CS (e.g., used in [25]) or arithmetic circuits (e.g., used in [24]). We assume that the IZK protocol enables input-loading, i.e., the private input is loaded before the statement is revealed (Definition 2) using a cryptographic commitment.

F.1 Range Check

If there is a witness that passes the checks of Algorithm 1, it must be that $a \leq x \leq b$. Otherwise, assume that $x < a$, then, if operations are defined in $\mathbb{F}_p$, $x - a > p/2$, and hence there is not an ℓ-vector $(\gamma_w)_{w=1}^{\ell}$ such that $x - a = \sum_{w=1}^{\ell} \gamma_w \cdot 2^{w-1}$. A similar argument holds if $x > b$.

F.2 Histogram Check

We first show how to check that $(\sigma_k)_{k=1}^{D}$ is the one-hot encoding (OHE) of an element $x \in \mathbb{F}_p$ with respect to buckets $(\mathfrak{b}_k)_{k=1}^{D}$, which is defined as follows: $\sigma_k = 1$ if $x \in \mathfrak{b}_k$, and $\sigma_k = 0$ otherwise. We assume that each bucket contains element in $\mathbb{F}_p$ in a specific range, i.e., $\mathfrak{b}_k = [a_k, b_k]$.

Algorithm 1 Range check

```
Statement: $a \leq x \leq b$
Public Input: $a, b \in \mathbb{Z}_{2^\ell}$, where $2^\ell < p/2$
Witness: $x \in \mathbb{F}_p$, $(\gamma_w)_{w=1}^{\ell} \in \mathbb{F}_p^{\ell}$ and $(\eta_w)_{w=1}^{\ell} \in \mathbb{F}_p^{\ell}$
Perform the following checks:
1: Check that $(\gamma_w)_{w=1}^{\ell}$ and $(\eta_w)_{w=1}^{\ell}$ are bits. For $w \in [\ell]$, check that $\gamma_w \cdot (\gamma_w - 1) = 0$ and $\eta_w \cdot (\eta_w - 1) = 0$
2: Check that $(\gamma_w)_{w=1}^{\ell}$ and $(\eta_w)_{w=1}^{\ell}$ are the bit-decompositions of $(x-a)$ and $(b-x)$ respectively:
    (a) Check that $x - a = \sum_{w=1}^{\ell} \gamma_w \cdot 2^{w-1}$
    (b) Check that $b - x = \sum_{w=1}^{\ell} \eta_w \cdot 2^{w-1}$
```

Algorithm 2 One-hot encoding check

```
Statement: $(\sigma_k)_{k=1}^{D}$ is OHE of $x \in \mathbb{F}_p$ with respect to $(\mathfrak{b}_k)_{k=1}^{D}$
Public Input: $(\mathfrak{b}_k)_{k=1}^{D} = ([a_k, b_k])_{k=1}^{D}$, where $a_k, b_k \in \mathbb{Z}_{2^w}$ and $2^\ell < p/2$
Witness: $x \in \mathbb{F}_p$, $(\sigma_k)_{k=1}^{D} \in \mathbb{F}_p^{D}$, $\alpha \in \mathbb{F}_p$, $\beta \in \mathbb{F}_p$, $(\gamma_w)_{w=1}^{\ell} \in \mathbb{F}_p^{\ell}$, and $(\eta_w)_{w=1}^{\ell} \in \mathbb{F}_p^{\ell}$
Perform the following checks:
1: Check that $(\sigma_k)_{k=1}^{D}$ are bits. For $k \in [D]$, check that $\sigma_k \cdot (\sigma_k - 1) = 0$
2: Check that $(\sigma_k)_{k=1}^{D}$ has single coordinate equal to 1. Check that $\sum_{k=1}^{D} \sigma_k = 1$
3: Check that if $\sigma_\kappa = 1$, then $a_\kappa \leq x \leq b_\kappa$.
    (a) Check that $\sum_{k=1}^{D} \sigma_k \cdot a_k = \alpha$
    (b) Check that $\sum_{k=1}^{D} \sigma_k \cdot b_k = \beta$
    (c) For $w \in [\ell]$, check that $\gamma_w \cdot (\gamma_w - 1) = 0$
    (d) For $w \in [\ell]$, check that $\eta_w \cdot (\eta_w - 1) = 0$
    (e) Check that $x - \alpha = \sum_{w=1}^{\ell} \gamma_w \cdot 2^{w-1}$
    (f) Check that $\beta - x = \sum_{w=1}^{\ell} \eta_w \cdot 2^{w-1}$
```

Algorithm 3 Histogram check

Statement: $\overrightarrow{\mathsf{count}}$ histogram of $(x_j)_{j=1}^N$ in buckets $(\mathfrak{b}_k)_{k=1}^D$
Public Input: $(\mathfrak{b}_k)_{k=1}^D = ([a_k, b_k])_{k=1}^D$, where $a_k, b_k \in \mathbb{Z}_{2^\ell}$ and $2^\ell < p/2$
Witness: $(x_j)_{j=1}^N \in \mathbb{F}_p^N$, $\overrightarrow{\mathsf{count}} = (c_k)_{k=1}^D \in \mathbb{F}_p^D$, and $(\sigma_{j,k})_{k=1}^D \in \mathbb{F}_p^D$ for $j \in [N]$
Perform the following checks:

```
1: Check that $(\sigma_{j,k})_{k=1}^D$ is OHE of $x_j$. For $j \in [N]$, perform a OHE check for $x_j$ (Algorithm 2)
2: Check counting. For $k \in [D]$, check that $\sum_{j=1}^N \sigma_{j,k} = c_k$
```

Note that Step 3 is a *private range check*. Specifically, the prover wants to perform the range check without revealing the corresponding bucket. The checks in Item (a) and Item (b) guarantee that the range check is performed for the range corresponding to bucket $\mathfrak{b}_\kappa$ for which $\sigma_\kappa = 1$.

An easy generalization of Algorithm 2 allows us to prove OHE for multidimensional buckets. In particular, a multidimensional bucket $\mathfrak{b}$ is defined by a set of ranges $([a_i, b_i])_{i=1}^{\mathsf{d}}$, one for each dimension. For instance, assume that our dataset has the attributes marital status and age, where marital status takes 5 different values (single, married, widowed, divorced, other) and age takes 12 values (1-10, 11-20, etc.). Then, each bucket corresponds to a specific value for the marital status and a specific range for the age, and the OHE of a data point has $5 \cdot 12 = 60$ coordinates. The generalization for multidimensional buckets follows the steps of Algorithm 2 except that in Step 3 the prover performs d private range checks, one for each attribute of the dataset.

Now, we are ready to describe the histogram check. We assume that each input takes a value in $\mathbb{F}_p$ for ease of notation. The histogram check can also be performed on multidimensional entries $\mathfrak{e} = (x)_{i=1}^{\mathsf{d}} \in \mathbb{F}_p^{\mathsf{d}}$ by calling the OHE check for multidimensional buckets.

F.3 Trimmed Mean Check

In the trimmed mean check, the prover $\mathbb{P}$ first proves whether each entry x_j falls in the range $[0, \theta]$ or not. Next, $\mathbb{P}$ performs a mean check for the population S_θ containing only the entries in $[0, \theta]$.

The final step is the same as in the proof of computation of the mean $\bar{x}$, which is discussed in §3.2.1.

F.4 Linear index

We show how to convert the multidimensional value of a data point for the attributes $(\mathsf{attr}_1, \ldots, \mathsf{attr}_{\mathsf{d}})$ into a field element $o \in \mathbb{F}_p$. Assuming that attr_i takes D_i distinct values, this value is akin to linear addressing a D-dimensional array where $D = \prod_{i=1}^{\mathsf{d}} D_i$. Namely, if $\overrightarrow{\sigma} = (0, 0, \ldots, 1, \ldots, 0)$ is the one-hot encoding of the data point for the attributes $(\mathsf{attr}_1, \ldots, \mathsf{attr}_{\mathsf{d}})$, then o is the linear index of the element of value 1.

If attribute attr_i consists of D_i non-overlapping ranges $([a_k, b_k])_{k=1}^{D_i}$, instead of distinct values, we perform linear index by first mapping the k-th range $[a_k, b_k]$ to the field element k. This can be verified in IZK using range checks.

Algorithm 4 Trimmed mean check

Statement: $\tilde{x}$ is the mean of $(x_j)_{j=1}^N$ trimmed at θ
Public Input: $\theta \in \mathbb{Z}_{2^\ell}$ with $\theta < 2^\ell < p/2$
Witness: $(x_j)_{j=1}^N \in \mathbb{F}_p^N$, $(b_j)_{j=1}^N \in \mathbb{F}_p^N$, $N_\theta \in \mathbb{F}_p$, and $\tilde{x} \in \mathbb{F}_p$
Perform the following checks:

```
1: Check that $(b_j)_{j=1}^N$ are bits. For $j \in [N]$, check that $b_j \cdot (b_j - 1) = 0$
2: Check that $b_j = 1$ if and only if $x_j \in [0, \theta]$. For $j \in [N]$, perform a range check for $b_j \cdot (\theta - x_j) + (1 - b_j) \cdot (x_j - \theta) \in [0, 2^\ell]$ (Algorithm 1)
3: Check that $N_\theta$ is the number of $x_j$'s that belong in $[0, \theta]$. Check that $\sum_{j=1}^N b_j = N_\theta$
4: Check that $\tilde{x}$ is the trimmed mean. Check that $\sum_{j=1}^N b_j \cdot x_j \approx N_\theta \cdot \tilde{x}$ using a range check (Algorithm 1)
```

Algorithm 5 Linear index

Statement: o is the linear index of $(x_1, \ldots, x_{\mathsf{d}})$ for attributes $(\mathsf{attr}_1, \ldots, \mathsf{attr}_{\mathsf{d}})$ where attr_i takes D_i distinct values
Public Input: $D_i \in \mathbb{F}_p$ for $i \in [\mathsf{d}]$ such that $\prod_{i=1}^{\mathsf{d}} D_i < p$
Witness: $\mathfrak{e} = (x_i)_{i=1}^{\mathsf{d}} \in \mathbb{F}_p^{\mathsf{d}}$, $o \in \mathbb{F}_p$
Perform the following checks:

```
1: Check that $o$ is the linear index of $\mathfrak{e}$. Check that $o = \sum_{i=1}^{\mathsf{d}} x_i \cdot \prod_{j=i+1}^{\mathsf{d}} D_j$.
```

G Unnormalized χ^2- Test

First, we define the generalized χ^2 distribution, denoted by

$$\tilde{\chi}^2(\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_D)\ .$$

A sample from $\tilde{\chi}^2(\mathsf{p}_1, \ldots, \mathsf{p}_D)$ is computed as follows: sample $(Z_1, \ldots, Z_D) \leftarrow\$ \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the multivariate normal distribution with mean vector $\boldsymbol{\mu} = (0, \ldots, 0)$ and covariance matrix $\boldsymbol{\Sigma}$ such that $\boldsymbol{\Sigma}[i][j] = \mathbb{E}[Z_i \cdot Z_j] = -\mathsf{p}_i \mathsf{p}_j$ and $\boldsymbol{\Sigma}[i][i] = \mathbb{E}[Z_i^2] = \mathsf{p}_i(1 - \mathsf{p}_i)$, and output $\sum_{j=1}^D Z_j^2$. We can compute the critical value $T_{\alpha, N, \mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_D}$ based on the public parameters $(\alpha, N, \mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_D)$.

Now, we show that the test passes if and only if the dataset is close to the public distribution. The unnormalized χ^2-test says that if the data is close to the public distribution, then

$$T = \sum_{j=1}^D \frac{(\mathsf{count}[j] - N\mathsf{p}_j)^2}{N} \xrightarrow{N \to \infty} \tilde{\chi}^2(\mathsf{p}_1, \ldots, \mathsf{p}_D),$$

otherwise the test statistic T is larger than expected.

When the dataset follows the public distribution, $\mathsf{count}[j]$ follows a binomial distribution with parameter p_j, since each of the N samples has probability p_j of belonging to the j-th bucket. Hence, $\mathbb{E}[\mathsf{count}[j]] = N \cdot \mathsf{p}_j$ and $\mathsf{Var}[\mathsf{count}[j]] = N \cdot \mathsf{p}_j(1-\mathsf{p}_j)$. If $Z_j := \mathsf{count}[j]/\sqrt{N} - \sqrt{N}\mathsf{p}_j$, then $\mathbb{E}[Z_j] = 0$, $\mathsf{Var}[Z_j] = \mathsf{p}_j(1-\mathsf{p}_j)$ and

$$\begin{aligned}\mathbb{E}[Z_i \cdot Z_j] &= \mathbb{E}[\mathsf{count}[i] \cdot \mathsf{count}[j]]/N - N\mathsf{p}_i\mathsf{p}_j \\ &= (N-1)\mathsf{p}_i\mathsf{p}_j - N\mathsf{p}_i\mathsf{p}_j = -\mathsf{p}_i\mathsf{p}_j.\end{aligned}$$

Let $\mathbf{X}_1, \dots \mathbf{X}_N$ be the one-hot encoding of the i-th sample in the D buckets. Then, $\sqrt{N}\left(\sum_{i=1}^{N} \mathbf{X}_i/N - \overrightarrow{\mathsf{p}}\right) = (Z_1, \dots, Z_D)$, where $\overrightarrow{\mathsf{p}} = (\mathsf{p}_1, \dots, \mathsf{p}_D)$. Thus, from the central limit theorem, $(Z_1, \dots, Z_D) \xrightarrow{N\to\infty} \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = (0, \dots, 0)$ and $\boldsymbol{\Sigma}$ such that $\boldsymbol{\Sigma}[i][j] = \mathbb{E}[Z_i \cdot Z_j] = -\mathsf{p}_i\mathsf{p}_j$ and $\boldsymbol{\Sigma}[i][i] = \mathbb{E}[Z_i^2] = \mathsf{p}_i(1-\mathsf{p}_i)$. So, the test statistic is close to the expected value.

When the dataset does not follow the public distribution, then there is some Z_j such that $\mathbb{E}[Z_j] \neq 0$. So, $T \not\xrightarrow{N\to\infty} \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and the test statistic will be larger than expected.

H Legendre PRF

In HOLMES, the multidimensional distribution testing requires the sampling of a pseudorandom matrix $\mathbf{R}$ as described in §3.3. We now provide details regarding the Legendre PRF used to sample this matrix.

Definition. Let p be an odd prime and $d \geq 2$ be an integer. The degree-d Legendre PRF [37–40] is a family of functions $L_{\mathsf{k}} : \mathbb{Z}_p^* \to \{-1, 1\}$ where $\mathsf{k} = (k_0, k_1, \dots, k_d) \leftarrow\$ \left(\mathbb{Z}_p^*\right)^d$, defined as $L_{\mathsf{k}}(x) = \left(\frac{f_{\mathsf{k}}(x)}{p}\right)$, where $f_{\mathsf{k}}(x) = k_0 + \sum_{i=1}^{d} k_i x^i$ is an irreducible degree-d polynomial and does not have a non-trivial stabilizer, and $\left(\frac{x}{p}\right) \in \{-1, 0, 1\}$ is the Legendre symbol. For $x \in \mathbb{Z}_p^*$, $L_{\mathsf{k}}(x) \neq 0$ because $f_{\mathsf{k}}(x)$ is irreducible and has degree at least 2.

Verification. Proving the evaluation of the Legendre PRF in IZK is efficient. The prover evaluates $f_{\mathsf{k}}(x) = k_0 + \sum_{i=1}^{d} k_i x^i$. To show that $\left(\frac{f_{\mathsf{k}}(x)}{p}\right) = y \in \{-1, 1\}$, it suffices to provide $a = \sqrt{f_{\mathsf{k}}(x)} \bmod p$ (if $y = 1$) or $a = \sqrt{b \cdot f_{\mathsf{k}}(x)} \bmod p$ (if $y = -1$) where b is any quadratic nonresidue in $\mathbb{F}_p$ as shown in Algorithm 6.

Sampling the Legendre PRF key. The protocol for the Legendre PRF key generation needs to sample a random key $\mathsf{k} = (k_0, k_1, \dots, k_r)$ such that $f_{\mathsf{k}}(x)$ is irreducible in $\mathbb{Z}_p^*$ and has no non-trivial stabilizer. This requirement has arisen recently [106, 107] as a safeguard against known attacks on Legendre PRF.

The Legendre PRF key generation is performed as follows. First, the parties sample a random degree-d polynomial and check its reducibility using Rabin's irreducibility test [108]. The polynomial f_{k} is irreducible with probability about $1/d$ [109, 110]. Even though the testing for non-trivial stabilizer is known to be very inefficient, for $d \geq 3$ a random polynomial over $\mathbb{Z}_p^*$ has non-trivial stabilizers only with probability at most $9/p$ [107]. Hence, following the recommendations in [107], we avoid non-trivial stabilizers by having $\gcd(p^2-1, d) = 1$. For our choice of parameter $d = 3$, this condition is satisfied for any prime larger than or equal to 3.

Algorithm 6 Legendre PRF Evaluation

```
Statement: $\left(\frac{x}{p}\right) = y \in \{-1, 1\}$
Public Input: $b \in \mathbb{F}_p$
Witness: $f_{\mathsf{k}}(x) \in \mathbb{F}_p$, $a \in \mathbb{F}_p$, $y \in \mathbb{F}_p$
Perform the following checks:
1: Check that $y \in \{-1, 1\}$. Check that $(y+1) \cdot (y-1) = 0 \bmod p$.
2: Check that $a$ is the modular square root of $f_{\mathsf{k}}(x) \bmod p$ (if $y = 1$) or $b \cdot f_{\mathsf{k}}(x) \bmod p$ (if $y = -1$).
   Check that $2a^2 = (1-y) \cdot b \cdot f_{\mathsf{k}}(x) + (1+y) \cdot f_{\mathsf{k}}(x) \bmod p$
```

Pseudorandomness of Legendre PRF. Though there is no known reduction to standard cryptographic assumptions, it has been conjectured, like other symmetric key primitives, that Legendre PRF is a pseudorandom function. The pseudorandomness of Legendre PRF has been studied in many different aspects, including linear complexity, collision, avalanche effect, and so on [111–113]. There is a recent trend to study Legendre PRF as a MPC-friendly PRF function [38, 39, 106, 107, 114, 115].

Note that in our setting, where we apply the Johnson-Lindenstrauss lemma, we do not require a high level of pseudorandomness—hash functions with bounded independence already suffice [116, 117]. We only need to ensure that the functions are chosen independently from the input, which is guaranteed by loading the data in IZK and MPC before sampling the PRF keys.

Recently, there have been several works on key-recovery attacks on Legendre PRF [39, 106, 107], which are not relevant to our settings because our protocol never hides the Legendre PRF keys. Our protocol only uses the Legendre PRF to generate a pseudorandom mapping. Still, in HOLMES, we conservatively choose our parameters to be sufficient to resist key-recovery attacks, so that we have some security margin against future attacks on pseudorandomness. Even with state-of-the-arts attack techniques [118], there are no known PRF distinguishing attacks on the Legendre PRF that are not based on key-recovery attacks [40]. Ethereum foundation currently has an active bounty program [114] on this open problem.

H.1 Comparison with other PRFs

In our ZK-friendly sketching, we use PRFs to obtain r pseudorandom values in $\{-1, 1\}$ given an index o. In the description of §3.3, these values are obtained using r different PRF keys, i.e., computing $\mathsf{PRF}_{\mathsf{k}_1}(o), \dots, \mathsf{PRF}_{\mathsf{k}_r}(o)$, and extracting a value from each of the outputs. This is optimal for the Legendre PRF, that we use, since its output is already a value in $\{-1,1\}$ However, if the PRF output contains ℓ bits, a more efficient solution is to use only $\lceil \frac{r}{\ell} \rceil$ keys to extract r values from the concatenation of their output.

Tab. 4 demonstrates the cost for evaluating r pseudorandom values in $\{-1,1\}$ given an index o, as well as their best-possible amortized cost by dividing the per-entry cost by r, for four PRFs: SHA-256, Rescue, Poseidon, and Legendre. A hash function (e.g., SHA-256) acts as a PRF by including the key as the prefix of the input, a technique known as domain separation.

We now explain where these costs, when the evaluation happens in an MPC protocol, like SCALE-MAMBA, over a field of size approximately 2^{62}.

- SHA-256 is often evaluated as a binary circuit in MPC. The state-of-the-art circuit, by Steven Goldfeder [119], takes 22272 AND gates and outputs 256 bits in the end. To obtain r bits, we need to invoke SHA-256 for $\lceil r/256 \rceil$ times.
- We consider Rescue and Poseidon with 128-bit security, a state of length 3, and parameter $\alpha = 5$. Each PRF invocation outputs 2 elements in $\mathbb{F}_p$ (i.e., 124 bits), the cost of which is 384 and 200 operations for Rescue and Poseidon, respectively. However, to obtain unbiased bits, we need to discard the highest 40 bits of each random element, which leaves us with 42 bits per PRF invocation. Since the PRF output is two field elements, we also need to compute their bit decomposition, which additionally requires a bit-by-bit comparison, and this leads to the cost of 124 operations and an additional input of 64 bits per 21 bits. Hence, the cost of bit decomposition for a value with 21 bits in 186 input and multiplication gates.
- The Legendre PRF computes $f_k(x) = k_0 + \sum_{i=1}^{d} k_i x^i$ of degree d ($d = 3$ in our setting) and extract the Legendre symbol $b = \left(\frac{f_k(x)}{p}\right) \in \{-1,1\}$. As described in Algorithm 6, this requires 2 input and 6 multiplication gates.

I Additional dataset workflows

I.1 Diabetes dataset workflow

The dataset [45, 46] consists of admission records for patients with diabetes. It includes of patient profile, treatment, and admission history. A use case is for hospitals to compare the success rate of different treatments. Therefore, hospitals can consider the following checks:

- Pearson's χ^2-test for the medical specialty of first checkups,
- t-test on the number of lab procedures across hospitals, and
- range checks for all the attributes.

We present the cost breakdown in Tab. 5.

	Cost per entry	Amortized by r
SHA-256	$22272 \cdot \lceil r/256 \rceil$	87
Rescue	$384 \cdot \lceil r/42 \rceil + 186 \cdot \lceil r/21 \rceil$	18
Poseidon	$200 \cdot \lceil r/42 \rceil + 186 \cdot \lceil r/21 \rceil$	14
Legendre	$8 \cdot r$	8

Table 4: Cost per entry for different PRFs counting the number of input gates and multiplication gates combined.

Table 5: Cost breakdown for the diabetes dataset (two-party).

Number of entries (101766×2)	203532
Number of attributes	49
Total time	33.43 s
Average time per entry	0.16 ms
Loading the data to IZK	1.59 s
Range tests for all attributes	23.27 s
Histogram and χ^2 test on *medical specialty*	7.59 s
Mean, variance, t test on *number of lab procedures*	0.23 s
Consistency check	< 0.01 s

I.2 Auctioning dataset workflow

The dataset [47] consists of advertisement biding history. It includes the advertisement location, its expected number of displays, and revenue. Bidders may want to train a model that predict the number of displays. So, bidders can consider the following two tests:

- z-test on the number of displays that are lower than a specific threshold, which checks if the bidding history between different parties is similar, excluding outliers, and
- range checks for all the attributes.

We present the cost breakdown in Tab. 6.

Table 6: Cost breakdown for the auctioning dataset (two-party).

Number of entries (567291×2)	1134582
Number of attributes	15
Total time	103.49 s
Average time per entry	0.09 ms
Loading the data to IZK	2.59 s
Range tests for all attributes	92.48 s
Trimmed mean and z test on *total displays*	7.45 s
Consistency check	< 0.01 s

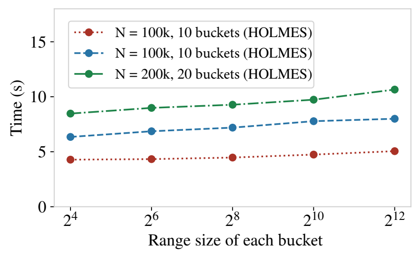
J Missing Figures

In this section, we include the figures which because of lack of space could not fit in the main body of the paper.

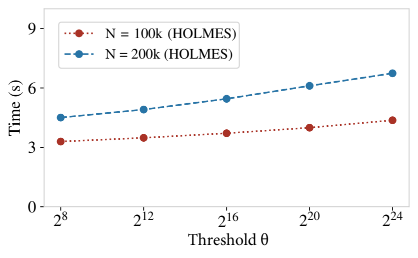
- Tab. 7 presents the tests and various distribution test gadgets of HOLMES.
- Tab. 1 shows the performance of range check on different systems.
- Tab. 3 shows the performance of ZK-friendly sketching on different systems.
- Fig. 7 presents the overhead of basic statistics and multi-dimensional χ^2-test for HOLMES and the generic MPC baseline in the case of $t = 2$ parties.

Table 7: Distribution test gadgets in HOLMES (α denotes the significance level).

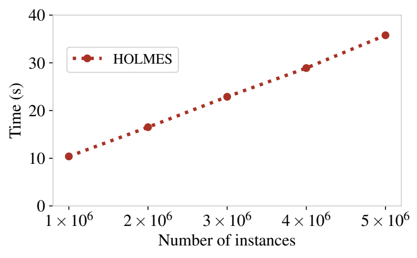
Gadget	**Description**
$\mathsf{range}(\langle S\rangle, \mathsf{attr}, [a,b]) \to \{\mathsf{yes}, \mathsf{no}\}$	check that values of attribute attr in population S fall in the range $[a,b]$
$\mathsf{histogram}(\langle S\rangle, (\mathsf{attr}_1, ..., \mathsf{attr}_\mathsf{d}), (\mathfrak{b}_1, \ldots, \mathfrak{b}_D)) \to \overrightarrow{\mathsf{count}}$	count values of attributes $(\mathsf{attr}_1, \ldots, \mathsf{attr}_\mathsf{d})$ in S in non-overlapping buckets $\mathfrak{b}_1, \ldots, \mathfrak{b}_D$
$\mathsf{mean}(\langle S\rangle, \mathsf{attr}) \to \bar{x}$	mean of values of attribute attr in population S
$\mathsf{variance}(\langle S\rangle, \mathsf{attr}) \to s^2$	variance of attribute attr in population S
$\mathsf{trimmedMean}(\langle S\rangle, \mathsf{attr}, \theta) \to \tilde{x}$	mean of values of attribute attr in S belonging in the range $[0, \theta]$
$\mathsf{zTest}(\langle S_1\rangle, \langle S_2\rangle, \mathsf{attr}, \sigma_1, \sigma_2, \alpha) \to \{\mathsf{yes}, \mathsf{no}\}$	z-test for mean equality for attribute attr of populations S_1 and S_2 with known standard deviations σ_1, σ_2, respectively
$\mathsf{tTest}(\langle S_1\rangle, \langle S_2\rangle, \mathsf{attr}, \alpha) \to \{\mathsf{yes}, \mathsf{no}\}$	t-test for mean equality for attribute attr of populations S_1 and S_2
$\mathsf{FTest}(\langle S_1\rangle, \langle S_2\rangle, \mathsf{attr}, \alpha) \to \{\mathsf{yes}, \mathsf{no}\}$	F-test for variance equality for attribute attr of populations S_1 and S_2
$\mathsf{chiSquaredTest}(\langle S\rangle, (\mathsf{attr}_1, \ldots, \mathsf{attr}_\mathsf{d}), (\mathfrak{b}_1, \ldots, \mathfrak{b}_D), (\mathsf{p}_1, \ldots, \mathsf{p}_D), \alpha) \to \{\mathsf{yes}, \mathsf{no}\}$	χ^2-test for goodness-of-fit for attributes $(\mathsf{attr}_1, \ldots, \mathsf{attr}_\mathsf{d})$ for population S for non-overlapping buckets $\mathfrak{b}_1, \ldots, \mathfrak{b}_D$ and public distribution $(\mathsf{p}_1, ..., \mathsf{p}_D)$



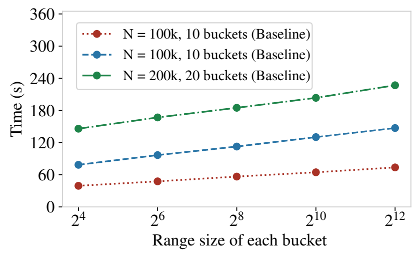
(a) Histogram (HOLMES)



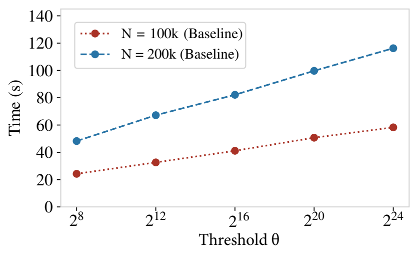
(b) Trimmed mean (HOLMES)



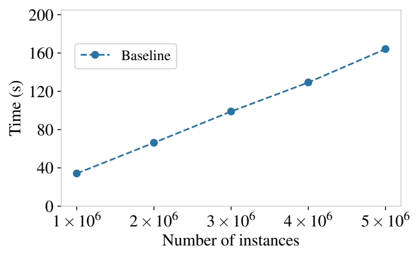
(c) Mean + variance (HOLMES)



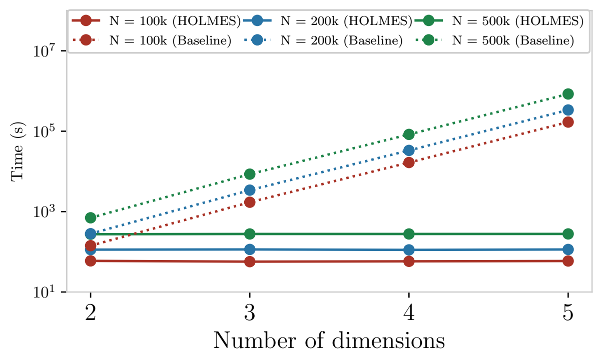
(d) Histogram (generic MPC)



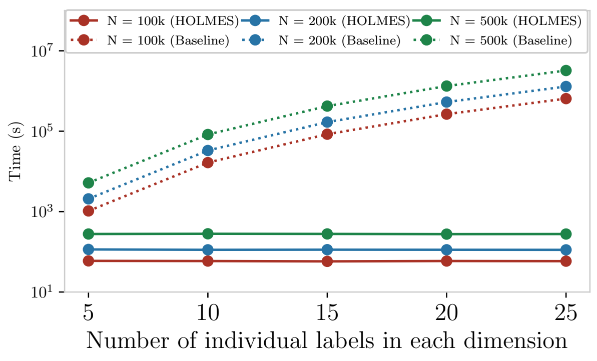
(e) Trimmed mean (generic MPC)



(f) Mean + variance (generic MPC)



(g) Multidimensional χ^2-test: $D_1 = \cdots = D_\mathsf{d} = 10$



(h) Multidimensional χ^2-test: $\mathsf{d} = 4$

Figure 7: Overhead of basic statistics and multidimensional χ^2-test for HOLMES and the generic MPC baseline for $t = 2$. The overhead of the naive χ^2-test and HOLMES multidimensional χ^2-test is in log scale.