HOLMES: A Platform for Detecting Malicious Inputs in Secure Collaborative Computation

Weikend Chen, Katerina Sotiraki, Ian Chan, Murat Kantarcioglu and Raluca Ada Popa

***Abstract*—Though maliciously secure multiparty computation (SMPC) ensures confidentiality and integrity of the computation from malicious parties, malicious parties can still provide malformed inputs. As a result, when using SMPC for collaborative computation, input can be manipulated to perform biasing and poisoning attacks. Parties may defend against many of these attacks by performing statistical tests over one another's input, before the actual computation.**

We present HOLMES, a platform for expressing and performing statistical tests securely and efficiently. Using HOLMES, parties can perform well-known statistical tests or define new tests. For efficiency, instead of performing such tests naively in SMPC, HOLMES blends together zero-knowledge proofs (ZK) and SMPC protocols, based on the insight that most computation for statistical tests is local to the party who provides the data.

High-dimensional tests are critical for detecting malicious inputs but are prohibitively expensive in secure computation. To reduce this cost, HOLMES provides a new secure dimensionality reduction procedure tailored for high-dimensional statistical tests. This new procedure leverages recent development of algebraic pseudorandom functions.

Our evaluation shows that, for a variety of statistical tests, HOLMES is $18\times$ to $40\times$ more efficient than naively implementing the statistical tests in a generic SMPC framework.

I. Introduction

To meet the increasing demands of big data, many services today try to gain access to diverse and wide-ranging datasets. For this reason, a recent trend between competing business organizations is to perform collaborative computation over their joint datasets, so they can make decisions based on more than just their own data [1–3]. This approach, however, comes with data privacy issues, as organizations are often unwilling, sometimes prohibited by regulators, to share data [4, 5].

A solution to this problem is secure multiparty computation (SMPC), which enables such collaboration without compromising privacy. SMPC has been used in various settings, such as data analytics and machine learning [6–22], and in a wide range of applications, such as medical [23] and financial [24].

Though SMPC ensures the privacy and correctness of the computation, it does not ensure that parties provide well-formed datasets as input. As a result, state-of-the-art works offering malicious security, such as Senate [6], Helen [11], and Private Deep Learning [19], have assumed that all parties provide well-formed data, though the security of these systems ensures that parties cannot deviate from the protocol in many other ways. However, if we consider real-world use cases, such as training models for anti-money laundering or for medical studies, manipulated input can lead to grave consequences. For instance, a malicious organization can gain an unfair advantage in market competition by contributing grossly biased data to make the result of collaborative computation unusable. This raises the following question:

Can we practically detect malicious input in secure collaborative computation?

Though identifying every possible malicious input is infeasible, in many scenarios we know properties that the honest input must satisfy. For instance, we know that age data must lie in a specific range (e.g., 0–100), and that in a typical city, only a small fraction of the population has age over 90. In fact, range checks [25–27] are frequently used to limit the effect of misreported values in secure computation.

However, range checks are not always enough. For example, assume that two banks with the same number of clients use the age of clients to predict the success of a bank marketing campaign. A malicious bank can decrease the combined mean age from, for example 20, to 10 by contributing manipulated data where all the ages are 1. Therefore, if some statistical characteristics of the data must be enforced, statistical hypothesis testing [28, 29] can be used as a general tool to check the quality of the input. Indeed, statistical testing has been a major tool in quality control [30–32], which checks the quality of all factors involved in manufacturing.

Building defenses against ill-formed or biased input is an active research area in machine learning [33–46]. In biasing attacks, the attacker provides biased data to reduce the accuracy of the model. In poisoning attacks, the attacker injects malicious input to the training dataset and influences a model. These attacks can not only affect the correctness, but also reveal information about the training data [47–49]. Various defenses against poisoning attacks are also based on computing statistical characteristics of the input [50, 51]. Thus, statistical tests are building blocks of many known and future defenses against biased or poisoned data in various settings.

We present HOLMES, a platform for expressing a rich class of statistical tests and performing them efficiently and securely. HOLMES does not aim to prescribe which specific tests each application should run because they depend on the use case, and new research may open up new defenses. Nonetheless, HOLMES enables parties in secure collaborative computation to express checks of statistical properties over the input by offering a rich set of statistical tests and building blocks, and performs these tests securely and efficiently. The efficiency gain is indeed important, as it allows parties to run more statistical tests with the same cost.

In sum, we envision that users of secure collaborative computation can use HOLMES to perform input checks before the actual computation, to detect malformed input.

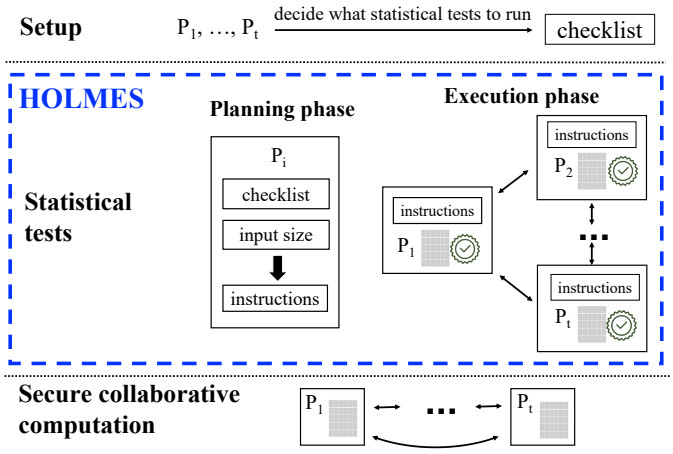


Fig. 1. Pipeline of secure collaborative computation: Identifying the set of required statistical checks and performing the collaborative computation are outside HOLMES. HOLMES's planning phase outputs a set of instructions that depend on the checklist and the input size, which are executed in HOLMES's execution phase for parties to check each other's input.

A. HOLMES's techniques

To illustrate HOLMES's techniques, we introduce the use case of bank marketing data [52, 53], the goal of which is to predict the success of a bank marketing initiative. Combining data from multiple banks results in better prediction mechanisms since, apart from having more data, the data reflects more diversified client demographics.

Overcoming the inefficiency of generic SMPC. Performing statistical tests in generic SMPC frameworks can be very inefficient, especially with malicious security [6, 11, 54, 55]. Interestingly, in many statistical tests, a large fraction of the computation involves only local computation, i.e., computation on the input of a single party.

However, in our setting where parties are malicious, we cannot trust them to perform local computation honestly. To leverage local computation for efficiency, HOLMES uses zero-knowledge proofs (ZK) to check that such local computation is performed correctly. More concretely, HOLMES divides statistical tests into a local part and a nonlocal part. The local part is checked using ZK, while the nonlocal part is executed in SMPC. HOLMES's planning algorithm devises a secure and efficient execution plan by resolving the dependencies of each test and avoids redundant computation in the various tests.

Finally, HOLMES incorporates a lightweight mechanism, described in Section III-A that checks the consistency between the data proved in ZK and the data provided to SMPC.

Efficient support of multidimensional statistical tests. One of the most prominent statistical tests is the goodness-of-fit test [56], which tests whether the input is close to a public distribution. Sometimes, the input domain is high-dimensional, i.e., involves various features, and has large support. For the bank marketing use case, a high-dimensional goodness-of-fit test would check if an input dataset has a specific distribution with respect to the attributes of age, job, marital status, and educational level. The different combinations of these attributes are approximately $100(\text{age}) \cdot 15(\text{jobs}) \cdot 5(\text{marital status}) \cdot 5(\text{education}) = 37,500$. We can see that the support, which—in the case of high-dimensional distributions corresponds to the number of different combinations—increases rapidly.

When the number of dimensions is high, goodness-of-fit tests are expensive, even without any security requirements. Naively implementing goodness-of-fit will lead to poor performance. HOLMES's approach is to first perform dimensionality reduction based on random projections, in which a vector in a high-dimensional space is linearly mapped to a space of lower dimension. Importantly for goodness-of-fit tests, this random linear mapping preserves distances approximately, as shown by the Johnson-Lindenstrauss (JL) lemma [57].

However, performing this random projection is challenging. In goodness-of-fit tests for high-dimensional distributions, the instance needs to be encoded as a one-hot vector of dimension equal to the support (e.g., $37,500$ in our example). Though this vector is sparse, we cannot use the sparseness to compute the encoding in secure computation, since revealing the positions of non-zero values leaks information about the dataset.

HOLMES's insight is to instantiate this random projection tailored for statistical tests without encoding, but approximately via the Legendre PRF [58–61], a pseudorandom function whose output is evenly distributed in $\{-1, 1\}$ and can be efficiently computed in many ZK and SMPC protocols. The Legendre PRF output naturally fits into the structure of the random projection matrix, and can significantly reduce the cost of dimensionality reduction by $5\times$ to $15000\times$ from a naive use of the JL lemma, based on our experiments in Section VII-F.

B. Our contributions

HOLMES enables parties to perform statistical checks on data before secure collaborative computation. The checks may involve statistical properties of data coming from a single party or from multiple parties. HOLMES supports common statistical tests, including tests for goodness-of-fit for high-dimensional distributions. Our design is modular, so new statistical tests can be easily defined using our building blocks. We implement and evaluate HOLMES in several scenarios, where we show useful input checks can be performed practically. In sum, HOLMES's contributions are the following:

- HOLMES provides a platform to express and securely perform well-known statistical tests, e.g., z-test, t-test, F-test, and χ^2 test, which is $18\times$ to $40\times$ more efficient than running the same tests in generic SMPC frameworks;
- HOLMES provides a planning algorithm that ensures security and efficiency for running multiple statistical tests by identifying the dependencies and removing the redundancy between these tests;
- HOLMES provides a new secure and efficient dimensionality reduction procedure based on the Johnson-Lindenstrauss lemma [57], which provides an efficiency improvement of up to $10^4\times$ compared with a straightforward baseline.

We plan to open-source HOLMES to enable users of SMPC to start taking advantage of statistical input checking.

II. Overview

When a number of parties want to perform a secure collaborative computation, they can use HOLMES to detect malicious inputs before the actual computation. To use HOLMES, as shown in Fig. 1, the parties need to first identify the statistical tests that are necessary to ensure a high quality input, based on the use cases, and assemble these tests into a checklist, which is provided to HOLMES. If the checks pass, the parties perform the SMPC computation using the committed input that has been checked by HOLMES.

In bank marketing example, parties may want to check are that age is within range, that the combined dataset distribution matches some known distribution, e.g., that ratios of the dataset in different combination of age, job, marital status, and educational level are close to some public ratios. The parties can also check the input quality by comparing the success rates of different banks for specific populations, assuming that in high quality data these rates should be similar across banks.

A. HOLMES architecture

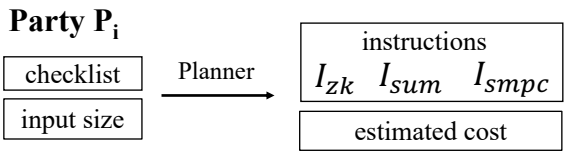
HOLMES consists of two phases—the planning phase and the execution phase—and three modules—the summary module, the ZK module, and the SMPC module. The workflow of HOLMES is summarized in Fig. 2.

Planning. In the planning phase, each party uses the predetermined checklist of statistical tests and information related to the size of its input to construct instructions $(I_{\mathsf{sum}}, I_{\mathsf{zk}}, I_{\mathsf{smpc}})$ for each module. The planning algorithm also estimates the cost for the execution phase.

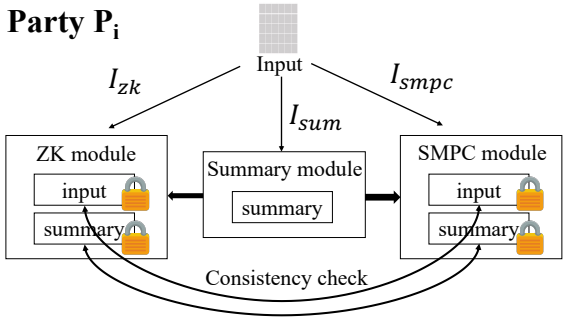
Each module takes as input the corresponding set of instructions and the dataset. In the summary module, the party locally computes some values that depend only on the party's dataset. Computation of statistical tests can be split into two parts: local computation—which involves the data of a single party—and nonlocal computation—which involves the input of multiple parties. The party commits the dataset and the summary to both the ZK and SMPC modules. A consistency check is performed between the ZK and SMPC modules to ensure that the values committed in the two modules are the same. In the ZK module, each party proves the local computation, and in the SMPC module, the party computes the nonlocal part of the statistical test using the committed data.

Summary module. Instructions I_{sum} specify the summary information that each party precomputes on its input. For instance, quantities that are typically needed in statistical analysis are the mean and variance. Another useful precomputation for some tests is to count how many data belong to a particular range. For example, for age, the party can count how many data points belong to ranges $[1, 10]$, $[11, 20]$, and so on.

ZK module. The ZK module with input I_{zk}, the dataset, and the summary values is responsible for verifying the local computations. The parties first commit their dataset and summary.



(a) Planning phase: outputs instructions for each module depending on the checklist and the input size.



(b) Execution phase: the summary includes precomputed values of the dataset. Dataset and summary are committed to both the ZK module and the SMPC module. The consistency check is performed over the committed data in these modules.

Fig. 2. Workflow of a party in HOLMES.

Then, they engage in a zero-knowledge protocol to prove that the local computation was done correctly.

SMPC module. Apart from the single-party computation that happens in the ZK module, many checks involve multiple parties; for instance checking that the combined dataset follows a public distributions requires multiparty computation. These statistical checks are computed in the SMPC module, where the input is I_{smpc}, the dataset, and the summary.

B. Statistical tests

Input checking is motivated by statistical hypothesis testing, a fundamental tool in statistics that rigorously checks if observed data satisfies some requirements. Examples include χ^2 goodness-of-fit test, the goal of which is to test whether data is drawn from a known distribution. In hypothesis testing, there are a null hypothesis and an alternative hypothesis. Our goal is to check if we have evidence to reject the null hypothesis. A statistical hypothesis test often consists of the following steps:

1) Compute a test statistic T, which is a quantity derived from the samples;
2) Calculate the critical value for a significance level α; the critical value T_α is the value for which the probability that we obtain a test statistic at most as extreme as T_α under the null hypothesis is α;
3) Check if T is bigger than T_α. If yes, then reject the null hypothesis in favor of the alternative hypothesis.

HOLMES supports well-known statistical tests, such as mean equality testing using z-test (known variance) or t-test (unknown variance), variance equality testing using F-test, and χ^2 test for goodness-of-fit. HOLMES also enables parties to

TABLE I
THE CHECKLIST API (α DENOTES THE SIGNIFICANCE LEVEL)

API function	Description
Basic tests and building blocks	
$\mathsf{range}(\langle S\rangle, \mathsf{attr}, [a,b]) \to \{\mathsf{yes}, \mathsf{no}\}$	check that values for an attribute attr in population S fall in the range $[a,b]$
$\mathsf{histogram}(\langle S\rangle, (\mathsf{attr}_1, ..., \mathsf{attr}_d), ([a_{11}, b_{11}], \ldots, [a_{\mathsf{d}u}, b_{\mathsf{d}u}])) \to \overrightarrow{\mathsf{count}}$	count elements of S in the non-overlapping multidimensional bins $\left[[a_{1k}, b_{1k}], \ldots, [a_{\mathsf{d}k}, b_{\mathsf{d}k}]\right]_{k=1}^{u}$ for attributes $(\mathsf{attr}_1, \ldots, \mathsf{attr}_\mathsf{d})$
$\mathsf{mean}(\langle S\rangle, \mathsf{attr}) \to \overline{x}$	mean of S for an attribute attr
$\mathsf{variance}(\langle S\rangle, \mathsf{attr}) \to s^2$	variance of S for an attribute attr
$\mathsf{trimmedMean}(\langle S\rangle, \mathsf{attr}, \theta) \to \overline{x}$	mean of elements in S belonging in the range $[0,\theta]$ for an attribute attr
$\mathsf{dimensionalityReduce}(\langle S\rangle, \mathsf{attr}_1, \ldots, \mathsf{attr}_d) \to (\mathsf{attr}'_1, \cdots, \mathsf{attr}'_\ell)$	perform dimensionality reduction for attributes $(\mathsf{attr}_1, \cdots, \mathsf{attr}_d)$ for S into a projection $(\mathsf{attr}'_1, \cdots, \mathsf{attr}'_\ell)$
Unidimensional tests	
$\mathsf{zTest}(\langle S_1\rangle, \langle S_2\rangle, \mathsf{attr}, \sigma_1, \sigma_2, \alpha) \to \{\mathsf{yes}, \mathsf{no}\}$	z-test for mean equality for an attribute attr of populations S_1 and S_2 with 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 an 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 an attribute attr of populations S_1 and S_2
Multidimensional tests	
$\mathsf{chiSquaredTest}(\langle S\rangle, (\mathsf{attr}_1, \ldots, \mathsf{attr}_\mathsf{d}), ([a_{11}, b_{11}], \ldots, [a_{\mathsf{d}u}, b_{\mathsf{d}u}]), (p_{11}, \ldots, p_{\mathsf{d}u}), \alpha) \to \{\mathsf{yes}, \mathsf{no}\}$	χ^2 test for goodness-of-fit of attributes $(\mathsf{attr}_1, \ldots, \mathsf{attr}_\mathsf{d})$ for populations S_1 and S_2 for multidimensional bins $[[a_{ik}, b_{ik}]_{i=1}^{\mathsf{d}}]_{k=1}^{u}$ and public distribution $[[p_{ik}]_{i=1}^{\mathsf{d}}]_{k=1}^{u}$
User-defined tests	
$\mathsf{userDefinedTest}(\langle S_1\rangle, \langle S_2\rangle, f, CVF_\alpha) \to \{\mathsf{yes}, \mathsf{no}\}$	user-defined statistical test on populations S_1 and S_2 described by f, with a critical value function CVF_α for significance level α

express new statistical tests, by providing parties with access to common test building blocks.

C. Checklist API

HOLMES provides a checklist API, as shown in Tab. I, for parties to express the statistical tests. HOLMES can then estimate the cost of the tests or run the tests. The API consists of well-known statistical tests. Parties can also define their own tests, using the basic tests and building blocks that the API provides. Every API function takes as input a description of a population S, which we denote by $\langle S\rangle$, a list of attributes of S where the test is performed, and other statistical parameters.

Example. We show how to use this API to express statistical tests for a bank marketing dataset described in Section VII-C.

Consider that two banks want to perform some statistical tests on their inputs. Assume that each of the banks has data for their clients, denoted by S_1 and S_2 respectively, with attributes including their age (e.g., 18 to 90), educational level (e.g., nominal values indexed by 0 to 3), and whether the marketing initiative was successful (e.g., 0 or 1).

- *Range check:* The age, the educational level, and the success indicator should fall in the right ranges. This can be expressed in the API as follows, for $i \in \{1,2\}$:

$$\mathsf{range}(\langle S_i\rangle, \text{“age”}, [18,90]) = \mathsf{yes}$$
$$\mathsf{range}(\langle S_i\rangle, \text{“education”}, [0,3]) = \mathsf{yes}$$
$$\mathsf{range}(\langle S_i\rangle, \text{“success”}, [0,1]) = \mathsf{yes}$$

- *Unidimensional test:* S_1 and S_2 should have similar mean age, which can be checked via a z-test when the standard deviation of the datasets is known, as follows:

$$\mathsf{zTest}(\langle S_1\rangle, \langle S_2\rangle, \text{“age”}, \sigma_1 = 15, \sigma_2 = 10, \alpha = 0.95) = \mathsf{yes}$$

- *Multidimensional test:* check that the histogram of the union of S_1 and S_2 for educational level equal to 0 and 1 and marital status equal to 0 and 1 has ratios $(0.1, 0.4, 0.3, 0.2)$, as follows:

$$\mathsf{chiSquaredTest}\left(\begin{array}{l}\langle S_1 \cup S_2\rangle,\\ (\text{“education”}, \text{“marital status”}),\\ ((0,0),(0,1),(1,0),(1,1)),\\ (0.1, 0.4, 0.3, 0.2), 0.90\end{array}\right) = \mathsf{yes}$$

- *User-defined test:* Besides the well-known statistic tests in the API, banks can add a customized test. One example is a test appearing in [62], which checks if the success rate of the marketing initiative p_1 for bank $\mathcal{P}_1$ is larger than the success rate p_2 of $\mathcal{P}_2$ by *at least* 0.2 using the test statistic $\frac{p_2 - p_1 + 0.2}{\sqrt{\frac{1}{n_1}p_1(1-p_1) + \frac{1}{n_2}p_2(1-p_2)}}$. This can be expressed in the API as “$\mathsf{userDefinedTest}(\langle S_1\rangle, \langle S_2\rangle, f, CVF_\alpha) = \mathsf{yes}$”, where

$$f = \frac{\overline{x}_2 - \overline{x}_1 + 0.2}{\sqrt{\frac{1}{n_1}\overline{x}_1(1-\overline{x}_1) + \frac{1}{n_2}\overline{x}_2(1-\overline{x}_2)}},$$

given the mean values $\overline{x}_1 = \mathsf{mean}(S_1, \text{"success"})$ and $\overline{x}_2 = \mathsf{mean}(S_2, \text{"success"})$, and CVF_α is a function that computes the critical value for this statistical test.

D. Threat model and security guarantees

HOLMES considers t parties who want to perform some secure computation together. They can collude with one another and arbitrarily deviate from the protocol, including in manipulating their input. In HOLMES, we assume that at least one of the t parties is honest and does not collude with any

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

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

- **Agreement:** $\mathcal{F}_{\mathsf{HOLMES}}$ first checks if all the parties send the same checklist and function to compute f, i.e.,
  - requires all $\mathsf{checklist}_1, \cdots, \mathsf{checklist}_t$ to be equal, and
  - requires all $f_1, \cdots, f_t$ to be equal.

  If not, $\mathcal{F}_{\mathsf{HOLMES}}$ sends the message $(\mathsf{disagreement}, sid)$ to each $\mathcal{P}_i$ and Sim and halt.
- **Security with abort:** $\mathcal{F}_{\mathsf{HOLMES}}$ awaits a message $(\mathsf{deliver}, sid)$ or (abort, sid) from Sim to decide whether the computation should move forward. $\mathcal{F}_{\mathsf{HOLMES}}$ proceeds to the next step if the message is $(\mathsf{deliver}, sid)$. Otherwise, $\mathcal{F}_{\mathsf{HOLMES}}$ sends (abort, sid) to each $\mathcal{P}_i$ and Sim and halt.
- **Input check:** $\mathcal{F}_{\mathsf{HOLMES}}$ runs the statistical tests specified on the checklist on each party's input. If the inputs do not pass the entire checklist, $\mathcal{F}_{\mathsf{HOLMES}}$ sends $(\mathsf{check\text{-}fail}, sid)$ to each $\mathcal{P}_i$ and Sim and halt.
- **Computation:** Now that all the input check has passed, $\mathcal{F}_{\mathsf{HOLMES}}$ now evaluates the function f on the parties' inputs input_1, $, \cdots, \mathsf{input}_t$. The result of the computation is t outputs, one for each party: $\mathsf{output}_1, \cdots, \mathsf{output}_t$. $\mathcal{F}_{\mathsf{HOLMES}}$ sends the message $(\mathsf{output}, sid, \mathcal{P}_i, \mathsf{output}_i)$ to $\mathcal{P}_i$ (and if $\mathcal{P}_i$ is corrupted, to Sim).

Fig. 3. Ideal functionality for secure computation with input checks.

corrupted parties, because if all the parties are corrupted, it is not even meaningful to check the inputs.

We define the security of HOLMES in the real/ideal world paradigm, with an ideal functionality $\mathcal{F}_{\mathsf{HOLMES}}$, shown in Fig. 3, for checking statistical properties of the input, based on existing definitions for secure function evaluation with abort [63, 64]. The ideal functionality $\mathcal{F}_{\mathsf{HOLMES}}$ takes as input a checklist and the input to the computation, and it outputs the computation results only if the tests in the checklist pass.

Based on $\mathcal{F}_{\mathsf{HOLMES}}$, we define the security of HOLMES using a standard definition for (standalone) malicious security [65]. We provide the security proof sketches in Appendix B.

Definition 1. *A protocol Π is said to* securely compute *$\mathcal{F}_{\mathsf{HOLMES}}$ in the presence of static malicious adversaries that compromise up to $t-1$ of the t parties, if, for every non-uniform probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ in the real world, there exists a non-uniform PPT simulator Sim in the ideal world, such that for every $I \subseteq \{1, 2, ..., N\}$,*

$$\{IDEAL_{\mathcal{F}_{\mathsf{HOLMES}}, I, \mathsf{Sim}(z)}(\vec{x})\}_{\vec{x}, z} \overset{c}{\approx} \{REAL_{\Pi, I, \mathcal{A}(z)}(\vec{x})\}_{\vec{x}, z}$$

where $\vec{x}$ denotes all parties' input, z denotes the auxiliary input for the adversary $\mathcal{A}$, $IDEAL_{\mathcal{F}_{\mathsf{HOLMES}}, I, \mathsf{Sim}(z)}(\vec{x})$ denotes the joint output of Sim and the honest parties, and $\mathrm{REAL}_{\Pi, I, \mathcal{A}}(\vec{x})$ denotes the joint output of $\mathcal{A}$ and the honest parties.

III. HOLMES's Workflow

HOLMES uses both ZK and SMPC protocols to perform statistical tests, which improves the efficiency compared with simply running tests in generic SMPC. However, this hybrid approach brings up a new challenge: HOLMES must ensure that ZK and SMPC protocols see the same data. Otherwise, a malicious party who provides different data to ZK and SMPC might bypass the tests.

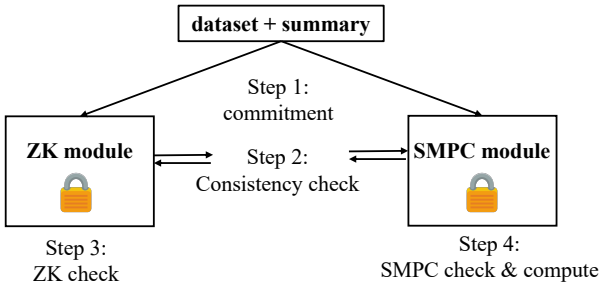


Fig. 4. The workflow of HOLMES in the execution phase.

HOLMES's approach is to design an execution phase that enables consistency checking between data in the ZK and SMPC modules, as shown in Fig. 4. The ZK and SMPC protocols that we use have a commitment phase, where parties commit their private data before engaging in the ZK or SMPC protocol. After the commitment phase, the parties can no longer change their data. It is worth noting that no actual "cryptographic commitment" is computed in the commitment phase; the commitment phase simply refers to the guarantee of the ZK and SMPC protocols that data cannot be changed after the initial insertion to the module.

A family of ZK protocols, called commit-and-prove ZK [63, 66–69], has such a commitment phase. In maliciously secure SMPC, the input phase is naturally a commitment phase.

The fact that the data has been committed allows us to check their consistency between the two modules with a *probabilistic* test. A malicious party cannot adapt its input data to circumvent the probabilistic test, since the data is committed and cannot be changed. We use polynomial testing [70–72], a common tool in zero-knowledge proof systems, to perform the consistency checking, which we now describe.

A. Polynomial testing

HOLMES assumes that the input to the ZK and SMPC module is represented as n field elements, $\{x_i^{\mathrm{ZK}}\}_{i=1}^n$ and $\{x_i^{\mathrm{SMPC}}\}_{i=1}^n$ respectively, over the same field $\mathbb{F}_p$. After the input has been committed in both modules, HOLMES performs the polynomial test. In this test, we represent the input as a polynomial $f(z)$ in the ZK and the SMPC module, and we query this polynomial at a random point $\beta \leftarrow\$ \mathbb{F}_p$. The polynomials in ZK and SMPC are defined as follows:

$$f^{\mathrm{ZK}}(z) = r + \sum_{i=1}^{n} x_i^{\mathrm{ZK}} \cdot z^i, \;\; f^{\mathrm{SMPC}}(z) = r + \sum_{i=1}^{n} x_i^{\mathrm{SMPC}} \cdot z^i \;\;,$$

where r is a random element that is also committed during the commitment phase. The random value r is used so that the query to $f(z)$ does not reveal information about the input. The random point $z = \beta$ is decided through a coin toss protocol [73] among the parties.

The parties use the SMPC module to compute $y^{\text{SMPC}} = f^{\text{SMPC}}(\beta)$ and release it to all parties. Then, the ZK module is invoked to prove that $y^{\text{SMPC}} = y^{\text{ZK}} = f^{\text{ZK}}(\beta)$. Such computation is relatively cheap in ZK and SMPC, as evaluating $f(\beta)$ is merely scalar multiplication with public values.

If the polynomial test passes, we know that with high probability $f^{\text{ZK}} = f^{\text{SMPC}}$, which implies that $x_i^{\text{ZK}} = x_i^{\text{SMPC}}$ for $i \in \{1, \cdots, n\}$. This holds due to the Schwartz-Zippel lemma [70–72], which states that for a random point β the probability that $f^{\text{ZK}}(\beta) = f^{\text{SMPC}}(\beta)$, but $f^{\text{ZK}} \neq f^{\text{SMPC}}$ is small.

Lemma 1 (Schwartz-Zippel Lemma)**.** *Let $f(x)$ be a non-zero polynomial of degree d in a prime field $\mathbb{F}_p$. Pick $y \leftarrow\$\, \mathbb{F}_p$. Then, we have $\mathbf{Pr}[f(y) = 0] \leq d/|\mathbb{F}_p|$.*

To achieve a sufficient level of statistical security, one can query the polynomials on more than one random point, while supplying more random elements to hide the polynomials.

B. Detailed execution phase workflow

We now summarize HOLMES's execution phase workflow, as shown in Fig. 4.

1) **Commitment:** Each party $\mathcal{P}_i$ commits their input, which consists of the dataset and the summary, to both the ZK and SMPC modules. As a result, parties cannot change the committed data in the rest of the execution phase.
2) **Consistency check:** All parties $\mathcal{P}_1, \cdots, \mathcal{P}_t$ together execute the polynomial test to check that each party's inputs to the ZK and SMPC module are the same. The parties halt if the polynomial test fails.
3) **ZK check:** Each party $\mathcal{P}_i$ invokes the ZK module, which executes the instructions provided by the planner, to prove to one another that the summary is correctly computed from the dataset. The parties halt if the ZK check fails.
4) **SMPC check and compute:** All parties $\mathcal{P}_1, \cdots, \mathcal{P}_t$ now run the remaining tests using the dataset and summary in the SMPC module. If the tests pass, the parties start the secure collaborative computation, using the dataset that has been committed in the SMPC module.

IV. HOLMES's statistical tests

In this section, we describe how HOLMES instantiates the API in Section II-C using the the summary, ZK, and SMPC modules.

A. Single-party basic tests and building blocks

HOLMES provides functions for basic tests and building blocks that involve the input of a single party: range, histogram, mean, variance, trimmed mean, and dimensionality reduction. These building blocks can be used to assemble more complicated tests, such as a z-test.

Range. The data provider uses the ZK module to prove that the input data is within a given range $[a, b]$. To prove that x lies in $[a, b]$, the data provider proves that $x - a \geq 0$ and $b - x \geq 0$, which can be done by proving that the bit decomposition of $x - a$ and $b - x$ uses at most $\lceil \log_2(b - a) \rceil$ bits.

Histogram. A histogram of a dataset is generated by counting the number of instances in a set of groups. This counting can be proved using the ZK module, since the data provider can locally compute the counting as part of the summary.

For each instance, the data provider inputs to the ZK module a one-hot encoding vector $\vec{v} = (0, 0, ..., 1, ..., 0, 0)$, the length of which is the same as the number of non-overlapping bins in the histogram. If $\vec{v}[i]$ is 1, then the instance belongs to the i-th bin. First, the data provider proves that $\vec{v}$ is one-hot, meaning that it has Hamming weight and Euclidean norm of 1. Next, the data provider shows that the instance entry belongs to the i-th bin, by performing range checks over the bounds for this bin. These bounds can be defined as inner products with $\vec{v}$.

If a histogram is needed, then this building block is included in the summary generated by the summary module. The ZK module checks if the summary has the correct histogram, and the SMPC module uses this histogram in the summary for subsequent statistical tests. In addition, computing a histogram on an attribute subsumes proving a range checking of this attribute. HOLMES's planning phase removes this redundancy by dropping the range check when appropriate.

Mean and variance. Mean and variance are building blocks in many statistical tests, such as z-tests and t-tests.

For mean, the data provider computes the mean $\overline{x}$ as part of the summary and proves the computation using the ZK module. If the dataset has n instances, the data provider shows that $n \cdot \overline{x} \approx \sum_{i=1}^{n} x_i$. In practice, we may want to keep a few decimal places for $\overline{x}$ (e.g., $\overline{x} = 12.34$ with two decimal places). This is done by defining $\overline{x}' = 1234$, which is a fixed-point representation of $\overline{x}$, and the data provider shows that $100 \cdot \sum_{i=1}^{n} x_i \geq n \cdot \overline{x}'$ and $100 \cdot \sum_{i=1}^{n} x_i < n \cdot (\overline{x}' + 1)$.

For variance, the data provider locally computes the mean $\overline{x}$ of the dataset and the mean of the squares of the dataset $\overline{y}$. Then, the variance can be computed as $s^2 = \frac{n}{n-1}(\overline{y} - \overline{x}^2)$. Here, the term $\frac{n}{n-1}$ corrects the bias of the variance because $\overline{x}$ is computed from the data [74].

If the mean or variance are needed, then the corresponding building blocks are included in the summary generated by the summary module. The ZK module checks if they are correct, and the SMPC module uses them for other tests. Before computing mean and variance on an attribute, a range checking of the attribute is needed. HOLMES's planning phase resolves this dependency and adds the range checks, if necessary.

Trimmed mean. Trimmed mean is a variant of mean that only considers instances that fall within a certain range $[0, \theta]$. This is useful for robust statistics, as one can exclude extreme values before computing the mean.

The data provider locally computes the trimmed mean in the summary and proves that the computation is correct in the ZK module. For each instance, the data provider first computes a bit b indicating whether each value falls in $[0, \theta]$ and proves that this bit is computed correctly through range checks. Next, the data provider computes the sum of the values s_θ in the entries where $b = 1$ and counts the number of such values n_θ, and then proves such computation in ZK. Finally, the data

provider computes and proves the trimmed mean, which is close to s_θ/n_θ, following the same procedure as for mean.

Dimensionality reduction. High-dimensional statistical tests, such as the χ^2 test for goodness-of-fit across many dimensions, are prohibitively expensive. A common solution in statistics is to first reduce the number of dimensions, while preserving some useful properties. HOLMES implements a secure and efficient procedure of dimensionality reduction based on the Johnson-Lindenstrauss lemma [57, 75], which compresses an entry with data of high dimensions into a vector of very few dimensions, denoted by $\vec{v} = (\mathsf{attr}'_1, \cdots, \mathsf{attr}'_\ell)$. The projected vector approximately preserves the norm of the original. This procedure involves more theoretical analysis, so we describe the details later in Section VI.

The result of dimensionality reduction, a short vector $\vec{v}$, is included in the summary by the summary module. The ZK module checks that this vector is computed correctly, and the SMPC module uses this vector for the χ^2 test for goodness-of-fit for high-dimension settings.

B. Multiparty basic tests and building blocks

Most single-party building blocks can be easily generalized to their multiparty variants, where the dataset is split among many parties. For range checks, the multiparty variant coincides with performing a range check to the input of each party independently. For the histogram construction, the final result is the summation of the individual histograms, so it suffices for the parties to perform independent histogram checks; then the summation of the individual histograms is done in the SMPC module. For the (trimmed) mean and variance, parties compute the necessary sums of their dataset and of the squares of their dataset in the summary and prove the correctness of the sums in the ZK module; then, the multiparty mean and variance are computed in the SMPC using the same equations as in the single-party case.

C. Instantiations of statistical tests

HOLMES implements four well-known statistical tests: z-test, t-test, F-test, and χ^2-test, as described in Section II-B. These statistical tests make use of the results from the local computation, which is included in the summary and is checked against the committed dataset of ZK module. The general form of statistical hypothesis tests is described in Section II-B. The test statistic and the comparison with the critical value are performed in the SMPC module. The critical value can be either given as a public parameter or is computed through a function, which typically is an oblivious lookup over a critical value table, if the critical value depends on secret information.

***z*-test.** The test statistic is defined as

$$T = \frac{\mathsf{mean}(S_1, \mathsf{attr}) - \mathsf{mean}(S_2, \mathsf{attr})}{\sqrt{\sigma_1^2/n_1 + \sigma_2^2/n_2}},$$

and the computation of the critical value depends only on n_1 and n_2, which are public information.

***t*-test.** Let $\overline{x}_1 = \mathsf{mean}(S_1, \mathsf{attr})$, $\overline{x}_2 = \mathsf{mean}(S_2, \mathsf{attr})$, $s_1^2 = \mathsf{variance}(S_1, \mathsf{attr})$, and $s_2^2 = \mathsf{variance}(S_2, \mathsf{attr})$. The test statistic is defined as:

$$T = \frac{\overline{x}_1 - \overline{x}_2}{\sqrt{\frac{1}{n_1}s_1^2 + \frac{1}{n_2}s_2^2}}, \mathrm{df} = \frac{\left(\frac{1}{n_1}s_1^2 + \frac{1}{n_2}s_2^2\right)^2}{\frac{1}{n_1-1}\left(\frac{1}{n_1}s_1^2\right)^2 + \frac{1}{n_2-1}\left(\frac{1}{n_2}s_2^2\right)^2}.$$

Note that in this case the degrees of freedom (df), which affect the computation of the critical value, depend on secret information. So, finding the critical value in the lookup table of critical values for different df happens in the SMPC module.

***F*-test.** The test statistic is defined as

$$T = \frac{\mathsf{variance}(S_1, \mathsf{attr})}{\mathsf{variance}(S_2, \mathsf{attr})}$$

and the computation of the critical value depends only on n_1 and n_2, so it can be performed publicly.

χ^2-test. Given the histogram $\overrightarrow{\mathsf{count}}$ of S over d attributes $(\mathsf{attr}_1, \cdots, \mathsf{attr}_d)$, the test statistic is

$$T = \sum_{k=1}^{m} \frac{(\mathsf{count}[k] - np_k)^2}{np_k},$$

where m is the total number of bins and p_k is the probability mass for the k-th bin of the public distribution. The critical value is given as a parameter. Observe that the computation in the SMPC module is proportional to the number of bins m, which might be a lot larger than the input size n. In Section VI, we explain HOLMES's approach for χ^2-test in the case of high-dimensional distributions, when $m \gg n$.

User-defined test. The parties can specify a new statistical test in HOLMES by providing a description of the test statistic f. The test statistic f can use the results of the basic tests and the building blocks. The parties also provide a function for computing the critical value CVF_α for a significance level α.

The data provider computes and proves the necessary results from the single-party basic tests and building blocks using the ZK module. These results are then included in the summary by the summary module. The SMPC module evaluates the multiparty building blocks and the test statistic f, finds out the corresponding critical value from CVF_α using information from the dataset (e.g., degree of freedom), and checks if the test statistic result is below the critical value. A typical CVF_α consists of a description of a distribution and a small table of critical values corresponding to significance level α for different degrees of freedom of the distribution.

D. Subsampling

HOLMES also supports statistical tests that are performed on a *random* subset of the entire dataset. Though this sacrifices some accuracy, it boosts the efficiency of individual tests and allows more tests to be performed with a given computational budget. The subsampling must be *maliciously secure*, since otherwise a malicious party knows which subset of the data would be selected.

HOLMES's approach is to decide this random subset only after the input data has been committed, and the random subset is chosen with the help of a pseudorandom function, using a seed that comes from a coin toss protocol among the t parties [73]. This ensures that if at least one of the t parties is honest, the corrupted parties cannot predict or decide what data would be selected in the subsampling.

V. HOLMES's Planning Algorithm

HOLMES's planning algorithm is responsible for choosing an efficient and secure execution plan for the statistical tests in the checklist and is executed by each party individually. The planning algorithm performs the following tasks.

Dependency resolution. Some of the statistical tests depend on other tests or building blocks. For example, both the t-test and the F-test depend on the variance of the datasets, and the tests for mean and variance require the attribute to go through a range check first. The planning algorithm resolves such dependencies, ensuring that all necessary checks are included.

Redundancy elimination. Several building blocks overlap with each other. For example, the computation of histogram subsumes range check. So, if the checklist computes a histogram on an attribute, the planning algorithm can drop the range check on this attribute without loss of security. Since range check contributes a modest overhead, as shown later in Section VII-D, such redundancy elimination is useful.

Instruction generation. The planning algorithm produces the instructions for the module for each building block or statistical test in the checklist. The planning algorithm separates the test between local and nonlocal checking, as described in Section IV. The local checking is performed in the ZK module, while the nonlocal checking is performed in the SMPC module, using the results of local checking.

For example, consider a z-test over four parties $\mathcal{P}_1$–$\mathcal{P}_4$ such that S_1 contains the dataset of $\mathcal{P}_1$ and $\mathcal{P}_2$, and S_2 contains the dataset of $\mathcal{P}_3$ and $\mathcal{P}_4$. The planning algorithm produces the summary instructions I_{sum} for computing the local part of mean computation, the ZK instructions I_{zk} for verifying the summary, and the SMPC instructions I_{smpc} for computing the combined means and the final test result.

VI. HOLMES's High-Dimensional Tests

In some situations, parties want to check if several attributes in each other's input dataset follow a public multidimensional distribution. Unfortunately, this test, known as *goodness-of-fit*, becomes prohibitively expensive when the number of possible combinations of attributes is high, which affects the applicability of the test. Recall that in the bank marketing example from Section I-A, the four attributes, age, jobs, marital status, and education, already have in total $37,500$ possible combinations. To improve the efficiency, we provide a new secure dimensionality reduction technique to handle this "curse of dimensionality".

A. HOLMES's goodness-of-fit

The most typical goodness-of-fit test is based on the χ^2-test, which is described in Section IV-C. In order to improve efficiency, especially in the high-dimensional case, HOLMES supports a modified χ^2-test, which we call *unnormalized* χ^2.

The unnormalized χ^2-test works as follows. Given a histogram $\overrightarrow{\mathsf{count}}$ of S over d attributes $(\mathsf{attr}_1, \cdots, \mathsf{attr}_d)$, the test statistic is

$$T = \sum_{k=1}^{m} (\mathsf{count}[k] - np_k)^2,$$

where m is the number of bins and p_k is the probability mass for the k-th bin of the public distribution. Now, the critical value is computed from a generalized χ^2 distribution with weight parameters $(np_1, \ldots, np_m)$ [76–78]. Since the weight parameters are public, we can find the critical value outside ZK and SMPC.

The unnormalized χ^2 test has improved efficiency compared to the original χ^2-test even when the number of bins is small, since it avoids divisions in SMPC. More importantly, in the high-dimensional regime the new test enjoys great efficiency gains using our dimensionality reduction techniques, as we show in the next section.

B. HOLMES's dimensionality reduction for goodness-of-fit

We provide the necessary background for dimensionality reuction in HOLMES by starting with a strawman.

Strawman: random linear projection. We perform the goodness-of-fit test using the unnormalized χ^2-test, which requires the computation $\|\mathsf{count}[k] - np_k\|^2 = \sum_{k=1}^{m} (\mathsf{count}[k] - np_k)^2$, i.e., the distance between two vectors. To reduce the dimensions of the vectors, one can apply a suitable random linear projection $\mathbb{F}_p^m \to \mathbb{F}_p^r$, which can be represented by a matrix A of size $r \times m$, to these vectors. The Johnson-Lindenstrauss (JL) lemma says that the projected vector approximately preserves the norm of the original vector.

Lemma (Johnson-Lindenstrauss lemma (informal))**.** *Let $\mathbf{x} \in \mathbb{Z}^m$ and let A be a random $r \times m$ matrix that satisfies certain uniformity and normality requirements, then $\sum_{k\in[m]} x_k^2 \approx \sum_{v\in[r]} (A\mathbf{x})_v^2$, where $r = O(\log m)$.*

To apply this random projection, parties jointly sample a public matrix A after data has been committed. Then, the unnormalized χ^2 tests with random projection is computed as follows:

1) In the summary module, Party i computes $\overrightarrow{\mathsf{JLvector}_i} = A \cdot \overrightarrow{\mathsf{count}_i}$, where $\overrightarrow{\mathsf{count}_i}$ is the histogram of Party i.
2) In the ZK module, Party i proves the correctness of $\overrightarrow{\mathsf{JLvector}_i}$.
3) In the SMPC module, parties approximate the unnormalized χ^2 statistic by computing

$$\widetilde{T} = \left\| \left(\sum_{i=1}^{t} \overrightarrow{\mathsf{JLvector}_i} - \overrightarrow{Q} \right) \right\|^2,$$

where $\overrightarrow{Q} = A \cdot (n\overrightarrow{p})$ depends only on public information. Finally, parties compare the test statistic with the publicly compute critical value.

Observe that the computation of the test statistic requires adding the JL vectors of all parties and $O(r) = O(\log(n))$ squaring operations, which is efficient in the SMPC module when $m \gg n$. However, in the ZK module, proving the computation of $\overrightarrow{\mathsf{JLvector}_i}$ still requires $O(m \cdot r)$ operations. As a result, applying the JL lemma naively provides only a small efficiency gain for multidimensional tests.

HOLMES's approach: pseudorandom projection via PRF. To overcome the inefficiency of computing $\overrightarrow{\mathsf{JLvector}_i}$ in the ZK module, we modify the random projection with a computation that requires $O(n \cdot r)$ operations in the ZK module, but otherwise preserves the efficiency of the strawman solution. A crucial step towards this goal is to replace the random matrix A with a ZK-friendly pseudorandom matrix.

Our main insight is that for input $x = \{x_1, \ldots, x_n\}$, where $1 \leq x_j \leq m$, it holds that $\overrightarrow{\mathsf{JLvector}_i} = \sum_{j=1}^{n} A_{x_j}$, where A_{x_j} is the x_j-th column of the matrix A. The equality holds because

$$\overrightarrow{\mathsf{JLvector}_i} = A \cdot \overrightarrow{\mathsf{count}_i} = \sum_{k=1}^{m} A_k \cdot \mathsf{count}_i[k] = \sum_{j=1}^{n} A_{x_j}, \tag{1}$$

where A_k is the k-th column of A.

Equation (1) shows that the random projection, when it is tailored for the histogram of a population, can be computed in a way that avoids computing the counting explicitly. In HOLMES, we can take advantage of this insight when we compute the matrix A using a pseudorandom function. Specifically, we set the (ℓ, k)-th matrix element $A_{\ell,k}$ to be $\mathrm{PRF}_{K_\ell}(k)$, for a pseudorandom function PRF_{K_ℓ} with seed K_ℓ. Then, the k-th column of A is equal to

$$A_k = \begin{pmatrix} \mathrm{PRF}_{K_1}(k) \\ \mathrm{PRF}_{K_2}(k) \\ \cdots \\ \mathrm{PRF}_{K_r}(k) \end{pmatrix}, \tag{2}$$

and hence, we can compute $\overrightarrow{\mathsf{JLvector}_i}$ as

$$\sum_{j=1}^{n} A_{x_j} = \sum_{j=1}^{n} \begin{pmatrix} \mathrm{PRF}_{K_1}(x_j) \\ \mathrm{PRF}_{K_2}(x_j) \\ \cdots \\ \mathrm{PRF}_{K_r}(x_j) \end{pmatrix}. \tag{3}$$

This new formula avoids the counting and can be computed with $n \cdot r$ calls to the PRF.

In sum, the (single-party) dimensionality reduction $A \cdot \overrightarrow{\mathsf{count}_i}$ is now efficiently checkable in ZK. The multiparty case works similarly, with the exception that in the end parties compute the summation of their output vectors in SMPC.

C. HOLMES's use of pseudorandom function

We now discuss the pseudorandom function that HOLMES uses. For HOLMES's random projection, we invoke the JL lemma of [75], where the JL matrix has elements in $\{-\sqrt{1/r}, \sqrt{1/r}\}$, because operations with this type of entries are easier to prove.

HOLMES uses Legendre PRF [58–61] for pseudorandom projection. This PRF can be computed efficiently and is more suitable than other algebraic pseudorandom functions [79–83] for our use case, as its output is supposed to be distributed evenly in $\{-1, 1\}$, which is only a constant away from the format of JL matrices ($\{-\sqrt{1/r}, \sqrt{1/r}\}$). We specify the variant of Legendre PRF used in HOLMES and refer readers to Appendix A for more background.

Definition 2 (degree-d Legendre PRF)**.** *Let p be an odd prime and $d \geq 2$ be an integer. The degree-d Legendre PRF [58–61] is a family of functions $L_K : \mathbb{Z}_p^* \to \{-1, 1\}$ where $K = (k_0, k_1, ..., k_d)$ such that $K \leftarrow\$ \left(\mathbb{Z}_p^*\right)^d$, and $L_{p,K}$ is defined as $\left(\frac{f_K(x)}{p}\right)$, where $f_K(x) = k_0 + \sum_{i=1}^{d} k_i x^i$ needs to be an irreducible degree-d polynomial and does not have nontrivial stabilizer, and $\left(\frac{x}{p}\right) \in \{-1, 0, 1\}$ is the Legendre symbol. For $x \in \mathbb{Z}_p^*$, $L_{p,K}(x) \neq 0$ because $f_K(x)$ is irreducible and has degree at least 2.*

In HOLMES's pseudorandom projection, we compute the random JL matrix A, whose entries are in $\{-\sqrt{1/r}, \sqrt{1/r}\}$, as follows:

$$A_{\ell,k} = L_{p,K_\ell}(k)\sqrt{1/r}\ ,$$

where $K_\ell \in \left(\mathbb{Z}_p^*\right)^d$ are different Legendre PRF keys.

Proving the evaluation of Legendre PRF in the ZK module is efficient. To show $\left(\frac{x}{p}\right) = y \in \{-1, 1\}$, it suffices to provide $a = \sqrt{x} \mod p$ (if $y = 1$) or $a = \sqrt{bx} \mod p$ (if $y = -1$) where b is a quadratic nonresidue. Thus, the correctness of a PRF evaluation is verified in the ZK module by showing that $2a^2 = (1-y) \cdot bx + (y+1) \cdot x \mod p$ and $(y+1)(y-1) = 0 \mod p$.

VII. Evaluation

We implement HOLMES and present the evaluation results, which answer the following questions:

- What is the overhead of HOLMES on real-world data? What contributes to this overhead? (Section VII-D)
- How do the individual components of HOLMES scale with more data? (Section VII-E)
- How does HOLMES's multidimensional testing compare with a strawman implementation? (Section VII-F)

A. Setup

We ran our experiments on two to five AWS c5.9xlarge instances, each with 36 cores and 72 GB of memory. We use the Linux `tc` tool to limit each instance's bandwidth to 2 Gbps and add a round-trip latency of 20 ms, which is to approximate the setting where the parties are in different states.

B. Implementation

We implement HOLMES and the baseline using the state-of-the-art cryptographic libraries, as follows.

HOLMES. We use QuickSilver [84] to instantiate the ZK module. HOLMES invokes QuickSilver in a pairwise manner, where each of the N parties proves the consistency of its local data and summary to the other $N - 1$ parties. We use SCALE-MAMBA [85] and MP-SPDZ [86, 87] for SMPC, where the

TABLE II
BREAKDOWN OF THE COST FOR THE BANK MARKETING DATASET (TWO-PARTY).

Number of instances (41188×2)	82376
Number of attributes	21
Total time	19.69 s
Average time per instance	0.24 ms
Loading the data to ZK	0.27 s
Loading the data to SMPC	8.09 s
Range checks for all attributes	5.63 s
Histogram over *age*	0.79 s
Multidimensional test over *age, job, marital status, education*	4.65 s
Variance over *call duration*	0.01 s
Mean over *call duration*	< 0.01 s
Consistency check	< 0.01 s

TABLE III
BREAKDOWN OF THE COST FOR THE DIABETES DATASET (TWO-PARTY).

Number of instances (101766×2)	203532
Number of attributes	49
Total time	79.73 s
Average time per instance	0.39 ms
Loading the data to ZK	1.52 s
Loading the data to SMPC	46.62 s
Range checks for all attributes	22.19 s
Histogram over *medical specialty*	6.79 s
Variance over *number of lab procedures*	0.03 s
Mean over *number of lab procedures*	< 0.01 s
Consistency check	< 0.01 s

TABLE IV
BREAKDOWN OF THE COST FOR THE AUCTIONING DATASET (TWO-PARTY).

Number of instances (567291×2)	1134582
Number of attributes	15
Total time	183.01 s
Average time per instance	0.16 ms
Loading the data to ZK	2.59 s
Loading the data to SMPC	79.55 s
Range checks for all attributes	92.48 s
Trimmed mean for *total impressions*	7.42 s
Consistency check	< 0.01 s

Low Gear protocol in MP-SPDZ is used for the offline phase of SMPC due to its efficiency, and SCALE-MAMBA is used for the online phase of SMPC. In HOLMES, these protocols operate on the same prime field $\mathbb{F}_p$ where $p = 2^{62} - 2^{16} + 1$, so we can perform the polynomial test in Section III-A efficiently.

Baseline. The baseline runs the same checking algorithm in SMPC, using SCALE-MAMBA and MP-SPDZ.

C. Datasets

To show how to perform various statistical and well-formedness tests with HOLMES for real-world data, we evaluate HOLMES's tests on three datasets. For each dataset, we demonstrate a few tests that fit the specific use case.

Bank marketing. The dataset [52, 53] consists of telemarketing records for financial products. It consists of 41188 instances and 21 attributes, which include client profile and call records. Banks' goal in a secure computation might be to train a classifier for predicting the success of a campaign. Before the training, though, they want to ensure that their joint dataset has a balanced number of customers from different backgrounds, which will improve the classifier's quality, and is not too different from each other, which is an indication that the provided data is correct. Therefore, they may consider the following checks: (1) χ^2 goodness-of-fit test over age, grouped into the bins 10–19, 20–29, $\cdots$, 90–99, (2) χ^2 goodness-of-fit test over age, job, educational level, and marital status, which depicts the customer profiles, and (3) t-test over the telemarketing call duration, to check whether their telemarketing records are similar enough to train a model together, and (4) range checks, for all the attributes.

Diabetes. The dataset [88, 89] consists of admission records for patients with diabetes. It consists of 101766 instances and 50 attributes, which include patient profile, treatment, and admission history. A use case of secure computation with this dataset is for evaluating the quality of service provided in the health system of a state. We consider the following checks: (1) χ^2 goodness-of-fit test for the medical specialty of initial admissions to compare with the publicly known distribution of specialties for doctors practicing in the state, (3) t-test on the number of lab procedures across hospitals, as an indication that hospitals provide the similar quality of service, and (4) range checks, for all the attributes.

Auctioning. The dataset [90] consists of advertisement biding history. It consists of 567291 instances and 17 attributes, which include the advertisement location and its expected number of impressions (i.e., number of ad displays) and revenue. Bidders want to train a machine learning system, but first they need a guarantee that each other has contributed reasonable data. So, they may consider the following tests: (1) t-test on the number of displays that are lower than a specific threshold, to exclude the extreme values, which shows if the bidding history between different parties is comparable enough to be trained together, and (2) range checks, for all the attributes.

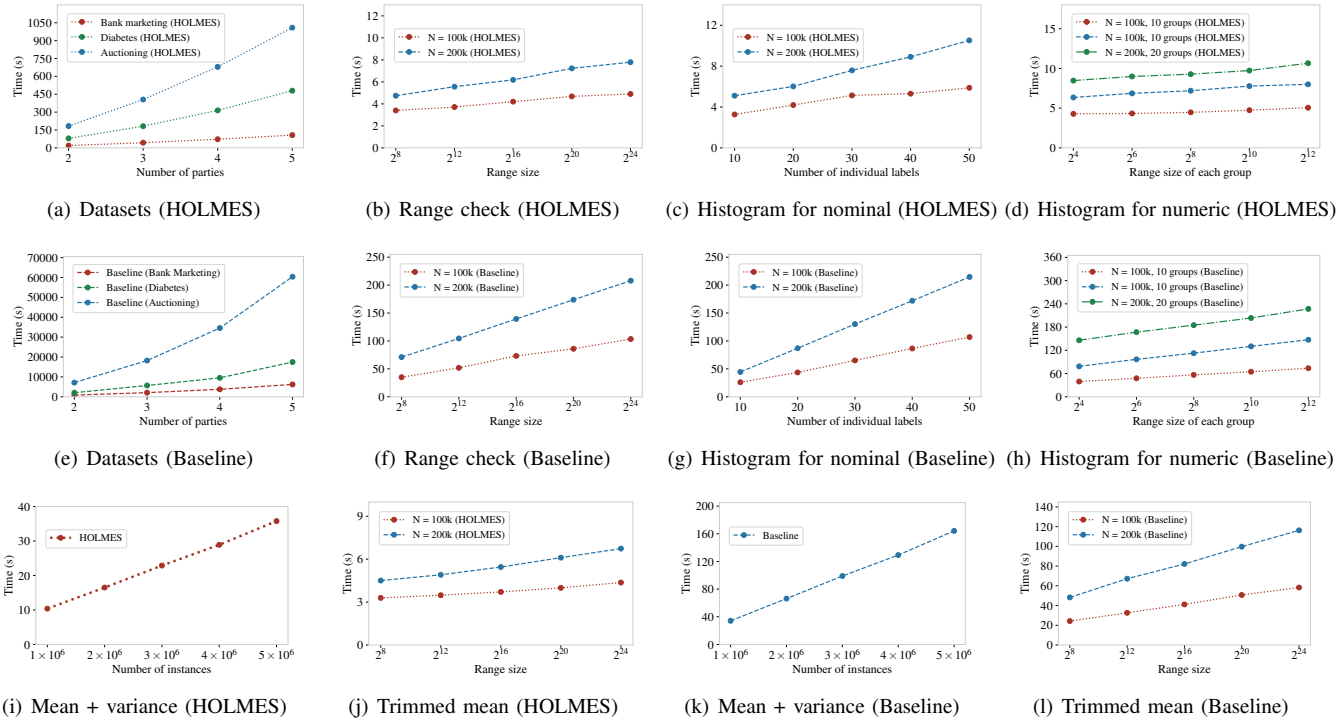
D. End-to-end overhead

We discuss the end-to-end overhead of HOLMES evaluated on the three datasets and compare HOLMES with the baseline.

In our experiment, we vary the number of parties from 2 to 5 to see how HOLMES scales with more parties. We assume that each party provides the same amount of the data, so when there are t parties, there are t times more data. For example, in our experiment for the bank marketing dataset, we assume each party provides $n = 41188$ instances (which is the size of our dataset). When there are five parties, the entire computation is now over $n \cdot t = 205940$ instances. We choose this approach because in secure collaborative computation with more parties, they should have access to more data. We choose the dimension parameter $r = 40$ for HOLMES's goodness-of-fit test based on Venkatasubramanian and Wang's study [91] by choosing an error rate $\epsilon = 3/4$.

Cost analysis. To understand how the overhead increases with more parties as well as more data, we first analyze the cost of HOLMES and the baseline, which we define as the wall-clock time to perform the tests.

The main overhead consists of performing the local checks in ZK or SMPC, loading data to SMPC, and computing the test statistics in SMPC. We express the cost as formulas where

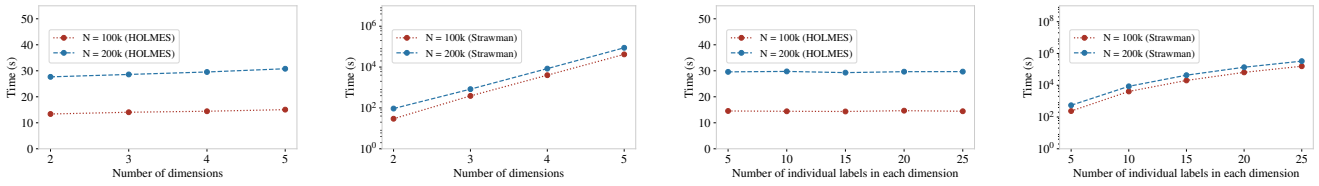


(a) Datasets (HOLMES) (b) Range check (HOLMES) (c) Histogram for nominal (HOLMES) (d) Histogram for numeric (HOLMES)

(e) Datasets (Baseline) (f) Range check (Baseline) (g) Histogram for nominal (Baseline) (h) Histogram for numeric (Baseline)

(i) Mean + variance (HOLMES) (j) Trimmed mean (HOLMES) (k) Mean + variance (Baseline) (l) Trimmed mean (Baseline)

Fig. 5. Overhead of the end-to-end checks on the datasets and individual components, for HOLMES and for the baseline.



(a) HOLMES, fixing the size of each dimension to 10 (b) Strawman, fixing the size of each dimension to 10 (c) HOLMES, fixing the number of dimensions to 4 (d) Strawman, fixing the number of dimensions to 4

Fig. 6. Overhead of the multidimensional goodness-of-fit test with dimensionality reduction, in HOLMES and in the strawman.

$C_1, C_2(t), C_3(t), C_4(t)$ represent the cost of cryptographic operations, n denotes the amount of instances provided by each party, and t denotes the number of parties.

$$\underline{\text{HOLMES:}} \quad \underbrace{C_1 \cdot n(t-1)}_{\text{local data checks in ZK}} + \underbrace{C_2(t) \cdot nt}_{\text{loading data to SMPC}} + \underbrace{C_3(t)}_{\text{test statistics}}$$

$$\underline{\text{Baseline:}} \quad \underbrace{C_4(t) \cdot nt}_{\text{local data checks in SMPC}} + \underbrace{C_2(t) \cdot nt}_{\text{loading data to SMPC}} + \underbrace{C_3(t)}_{\text{test statistics}}$$

As shown in the highlighted part, HOLMES and the baseline differ in the cost of performing local data checks, which is also the dominating part of the overhead. The differences between HOLMES and baseline are due to two reasons. First, recall that HOLMES uses the ZK in a pairwise manner between the parties. Since a party does not need to prove anything to itself, each party only needs to prove $(t-1)$ times instead of t times, while the SMPC is run over all parties' dataset, of size $n \cdot t$. Second, C_1 tends to be smaller than $C_4(t)$, on the one hand because ZK is more restricted than SMPC and oftentimes can be implemented more efficiently, and on the other hand, while $C_4(t)$ tends to grow (at least) linearly with the number of parties, C_1 is independent of the number of parties (since HOLMES invokes ZK in a pairwise manner).

Results. We present the results in Fig. 5(a). Because the overheads of HOLMES and baseline are very different, and we still want to discuss their individual growth patterns, we present them in separate graphs. For the three datasets, HOLMES outperforms the baseline by $18\times$ to $40\times$. In addition, this gap is larger when the number of parties increases, which is consistent with the cost analysis. From Fig. 5(a) we can see that the overhead of HOLMES grows almost linearly, while the overhead of baseline grows slightly faster than linear. This is consistent with the cost analysis, as the term $C_4(t) \cdot nt$ in baseline is growing faster than $C_1 \cdot n(t-1)$ when the number of parties t increases, because $C_4(t)$ also grows linearly to t.

Breakdown. To understand what contributes to this overhead, we present the breakdown of the overhead in Tab. II, Tab. III,

and Tab. IV. As the tables show, the average time per instance varies between datasets because we perform different tests on them. In all three experiments, we can see that range check contributes to a large portion of the overhead, followed by the cost of loading data into SMPC. In contrast, multidimensional testing is quite efficient. The consistency check between ZK and SMPC also has a small overhead.

E. Overhead of the individual components

To understand more concretely the overhead of HOLMES and how it outperforms the baseline, we present the overheads for a number of basic tests and building blocks: range check, histogram, mean, variance, and trimmed mean, as shown in Fig. 5. We defer the discussion of dimensionality reduction to Section VII-F since we focus later on how it compares with a strawman implementation. We now discuss these components.

Range check. As shown in Fig. 5(b) and Fig. 5(f), the overhead of range check grows almost linearly to the number of instances and logarithmically to the size of the range, as expected. HOLMES performs range checks about $4\times$ to $19\times$ more efficiently than the baseline.

Histograms. We consider histograms over nominal and numeric attributes. The difference is that the latter needs to arrange data belonging to a range into a group (e.g., age $[20, 29]$), which incurs additional overhead. As shown in Fig. 5(c) and Fig. 5(g), the overhead grows almost linearly to the number of individual labels in a nominal attribute, and linearly to the number of instances, as expected. From Fig. 5(d) and Fig. 5(h), for the numeric case, the overhead grows linearly to the number of groups and the number of instances, and logarithmically to the range size of each group. HOLMES is about $4\times$ to $10\times$ more efficient than baseline.

Mean and variance. Mean and variance are relatively cheap to compute, so we experiment with more instances to better show the growth patterns (one million to five million). As shown in Fig. 5(i) and Fig. 5(k), the overhead is linear to the number of instances (mainly the cost to load data into SMPC). HOLMES is only about $2\times$ to $3\times$ more efficient.

Trimmed mean. We show the overhead of trimmed mean in Fig. 5(j) and Fig. 5(l). The overhead grows almost linearly to the number of instances and logarithmically to the range size, as expected. HOLMES is about $4\times$ to $10\times$ more efficient than the baseline.

F. Overhead of dimensionality reduction

We now describe the overhead of HOLMES's dimensionality reduction technique and compare it with a strawman that only uses random linear projection (from the JL lemma).

Cost analysis. The overheads of HOLMES's approach and the strawman are very different. Consider a multidimensional test with C_5 nominal attributes each with C_6 independent labels where $C_6^{C_5} \ll p \approx 2^{61}$. Assume that the projected vector has r dimensions. Then, for each instance in the dataset, the overhead for dimensionality reduction can be expressed as the number of input and multiplications of private values in ZK, as follows.

$$\underline{\text{HOLMES:}}\ \underbrace{5 \cdot r}_{\text{cost of PRF}},\ \underline{\text{Strawman:}}\ \underbrace{C_6^{C_5} \cdot (C_5 - 1) + 2 \cdot C_5 \cdot C_6}_{\text{cost to create one-hot encoding}}$$

Results. We choose the parameter r based on [91] with an error rate $\epsilon = 3/4$. We now discuss the growth patterns in Fig. 6. As shown in Fig. 6(a) and Fig. 6(c), as long as $C_6^{C_5} \ll p$, HOLMES's approach is almost independent of C_5 and C_6, but just linear to the number of dimensions in the projected vector r and the number of instances, as expected.

We present the strawman case, drawn in log scale, in Fig. 6(b) and Fig. 6(d). The strawman is very expensive, as we could only run the experiment in smaller scales and extrapolate the results. We see that the overhead grows exponentially to the number of dimensions. With a fixed number of dimensions, the overhead grows much slower than an exponential function (indeed, it is expected to be close to a degree-4 polynomial function). Both are expected based on the cost analysis. In sum, HOLMES's approach improves the efficiency of dimensionality reduction especially for high dimensions, up to $10^4\times$ based on our evaluation.

VIII. Related work

Secure collaborative computation systems. We envision that HOLMES can be combined with systems for secure analytics and machine learning [6–9, 11–22]. Our platform is especially useful when malicious security is required, and parties are not trusted to provide their honest inputs.

Secure multiparty computation. A rich body of works propose SMPC protocols [92–94] for malicious adversaries and dishonest majority, with SPDZ [95–97] and authenticated garbling [98–101] being the state-of-the-art. HOLMES uses SPDZ because it is more suitable for arithmetic computation.

There are works [102, 103] that use SMPC techniques for statistical tests. In contrast to our platform, they mostly focus on the two-party case and consider different threat models.

Zero-knowledge proofs. Zero-knowledge proofs (ZK) [104] enable a party to prove a statement without leaking any information. Recently, constructing practical ZK has gained much attention, especially since succinct non-interactive proofs [27, 105–108] have been used in blockchains. In HOLMES, we want small proving time, so succinctness and non-interactivity are not important. Hence, we use QuickSilver [84].

Verifiable computation. A research area related to HOLMES is verifiable computation [67, 109–114], in which one or more parties prove that some (secure) computation is done correctly. This is different from HOLMES, as we do not check that the collaborative computation is done correctly (which is the duty of SMPC) but instead focuses on the inputs that the parties provide. Moreover, instead of focusing on verifying general computation, HOLMES is tailored to perform statistical tests securely and efficiently, such as supporting efficient dimensionality reduction for multidimensional data.

Robust statistics. Robust statistics [115–117] focuses on statistics that are resilient to nonstandard input distributions. HOLMES shares the similar motivation, but works very differently. Robust statistics aims at making algorithms resilient to incoming malicious inputs, while HOLMES tries to detect them. HOLMES also provides the necessary privacy guarantees for secure collaborative computation.

Poisoning attacks. Poisoning attacks [118, 119] happen when the attacker can inject manipulated data to the training set and cause the model to fail on certain inputs. There are existing defenses trying to mitigate poisoning attacks [33]. One defense is to filter out potentially poison data by identifying outliers in the training data [120–123], which has the potential to be represented as a user-defined test in HOLMES.

Differential privacy. Differential privacy (DP) [124–126] is a mechanism that adds noise to the dataset or the model to hide information about the individual sample in the dataset. DP is slightly related to HOLMES because DP has been used to improve the robustness of the training phase, and is a common feature of secure collaborative learning systems [127–131]. HOLMES can provide robustness in addition to what DP offers by ensuring the input follows the desirable distributions. It is an open problem whether HOLMES can work more closely with DP, such as in checking that each party has correctly added a data-independent noise to their inputs.

Hardware enclave. An alternative to cryptographic collaborative computation is to use hardware enclaves [132]. Several systems have used enclaves for encrypted databases and data analytics [133–136]. However, a well-known issue with enclaves is that they are susceptible to side-channel attacks [137–144], which may reveal access patterns.

IX. Discussion and limitations

HOLMES is only the beginning of checking inputs for maliciously secure collaborative computation. We now discuss some challenges that HOLMES has not solved.

Identifying necessary tests. HOLMES enables parties to specify and perform statistical input checks specific to their use case before the collaborative computation. It does not, however, make any decisions regarding what the necessary tests are. Identifying attacks and proposing defenses in learning systems is an active area of research [33–46], but at this moment, there is no systematic approach that identifies the necessary tests for defending against a general class of attacks, such as biasing attacks. HOLMES's contribution is to enable expressing a rich class of statistical tests, both enabling tests that parties perform today [25–27, 120–123] and aiming to support input checks that will be developed in the future.

Beyond statistical tests. Not all desired input checks can be expressed as statistical tests. In data analytics applications, there might be consistency concerns *between* instances. For example, in a dataset consisting of patient admissions, the same patient may have several records, each with the patient's date of birth. The application may want to check that the same patient has only one date of birth in the dataset. In other applications, the parties might perform input checks with the help of machine learning model, e.g., run a classifier on the input data [120–123]. These checks are outside of HOLMES's scope, but can potentially be performed in other existing systems. Systems for secure data analytics [6] and for running machine learning models privately [11, 15, 19, 20, 54, 55, 145] are able to perform input checks again malicious parties. Integrating such systems with HOLMES would allow expressing an even richer class of input checks.

X. Conclusion

We present HOLMES, a platform for expressing and performing rich statistical tests over input data for secure collaborative computation. HOLMES then enables running such tests in an efficient and secure way. First, HOLMES's techniques blend together efficient zero-knowledge proofs and secure multiparty computation protocols to leverage the local computation in statistical tests. Second, for high-dimensional data, HOLMES contributes a new secure dimensionality reduction procedure that leverages the sparseness securely and significantly outperforms naive approaches. Finally, we implemented HOLMES and demonstrated that it can perform useful input checks for several use cases.

We plan to open-source HOLMES to enable users of SMPC to start taking advantage of statistical input checking.

REFERENCES

[1] Nature Communications Editorial, "Data sharing and the future of science," in *Nature Communications '18*.

[2] H. A. Piwowar and T. J. Vision, "Data reuse and the open data citation advantage," in *PeerJ '13*.

[3] M. Packer, "Data sharing in medical research," in *British Medical Journal '18*.

[4] H. L. Williams, "Intellectual property rights and innovation: Evidence from the human genome," in *Journal of Political Economy '13*.

[5] M. W. Carroll, "Sharing research data and intellectual property law: A primer," in *PLoS Biology '15*.

[6] R. Poddar, S. Kalra, A. Yanai, R. Deng, R. A. Popa, and J. M. Hellerstein, "Senate: A maliciously-secure MPC platform for collaborative analytics," in *SEC '20*.

[7] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *S&P '17*.

[8] M. Chase, R. Gilad-Bachrach, K. Laine, K. E. Lauter, and P. Rindal, "Private collaborative neural network learning," in *IACR ePrint 2017/762*.

[9] I. Giacomelli, S. Jha, M. Joye, C. D. Page, and K. Yoon, "Privacy-preserving ridge regression with only linearly-homomorphic encryption," in *ACNS '18*.

[10] M. Kantarcıoğlu and C. Clifton, "Privacy-preserving distributed mining of association rules on horizontally partitioned data," in *IEEE TKDE '04*.

[11] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *S&P '19*.

[12] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *CCS '17*.

[13] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *SEC '18*.

[14] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "XONN: XNOR-based oblivious deep neural network inference," in *SEC '19*.

[15] V. Chen, V. Pastro, and M. Raykova, "Secure computation for machine learning with SPDZ," in *NeurIPS '18*.

[16] A. Tueno, F. Kerschbaum, and S. Katzenbeisser, "Private evaluation of decision trees using sublinear cost," in *PETS '19*.

[17] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *S&P '13*.

[18] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans, "Privacy-preserving distributed linear regression on high-dimensional data." in *PETS '17*.

[19] H. Chen, M. Kim, I. Razenshteyn, D. Rotaru, Y. Song, and S. Wagh, "Maliciously secure matrix multiplication with applications to private deep learning," in *ASIACRYPT '20*.

[20] W. Zheng, R. Deng, W. Chen, R. A. Popa, A. Panda, and I. Stoica, "Cerebro: A platform for multi-party cryptographic collaborative learning," in *SEC '21*.

[21] C. A. Choquette-Choo, N. Dullerud, A. Dziedzic, Y. Zhang, S. Jha, N. Papernot, and X. Wang, "CaPC learning: Confidential and private collaborative learning," in *Arxiv:2102.05188*, 2021.

[22] M. Abspoel, D. Escudero, and N. Volgushev, "Secure training of decision trees with continuous attributes," in *PETS '21*.

[23] L. Kamm, D. Bogdanov, S. Laur, and J. Vilo, "A new way to protect privacy in large-scale genome-wide association studies," in *Bioinformatics '13*.

[24] E. A. Abbe, A. E. Khandani, and A. W. Lo, "Privacy-preserving methods for sharing financial risk exposures," in *American Economic Review '12*.

[25] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *NSDI '17*.

[26] S. Addanki, K. Garbe, E. Jaffe, R. Ostrovsky, and A. Polychroniadou, "Prio+: Privacy preserving aggregate statistics via Boolean shares," in *IACR ePrint 2021/576*.

[27] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *S&P '18*.

[28] E. L. Lehmann and J. P. Romano, *Testing statistical hypotheses*. Springer Science & Business Media, 2006.

[29] E. L. Lehmann, *Testing statistical hypotheses*. Wiley, 1959.

[30] W. A. Shewhart and W. E. Deming, *Statistical method from the viewpoint of quality control*. Courier Corporation, 1986.

[31] A. Mitra, *Fundamentals of quality control and improvement*. John Wiley & Sons, 2016.

[32] D. C. Montgomery, *Introduction to statistical quality control*. John Wiley & Sons, 2020.

[33] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, "Sok: Security and privacy in machine learning," in *EuroS&P '18*.

[34] M. Mozaffari-Kermani, S. Sur-Kolay, A. Raghunathan, and N. K. Jha, "Systematic poisoning attacks on and defenses for machine learning in healthcare," in *IEEE Journal of Biomedical and Health Informatics '15*.

[35] M. B. Zafar, I. Valera, M. G. Rodriguez, and K. P. Gummadi, "Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment," in *WWW '17*.

[36] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning," in *ACM Computing Surveys '21*.

[37] T. Speicher, H. Heidari, N. Grgic-Hlaca, K. P. Gummadi, A. Singla, A. Weller, and M. B. Zafar, "A unified approach to quantifying algorithmic unfairness: Measuring individual & group unfairness via inequality indices," in *KDD '18*.

[38] K. Webster, M. Recasens, V. Axelrod, and J. Baldridge, "Mind the GAP: A balanced corpus of gendered ambigu-

ous pronouns," in *Transactions of the Association for Computational Linguistics '18*.

[39] R. Mehrotra, J. McInerney, H. Bouchard, M. Lalmas, and F. Diaz, "Towards a fair marketplace: Counterfactual evaluation of the trade-off between relevance, fairness & satisfaction in recommendation systems," in *CIKM '18*.

[40] S. Corbett-Davies, E. Pierson, A. Feller, S. Goel, and A. Huq, "Algorithmic decision making and the cost of fairness," in *KDD '17*.

[41] J. Zhang and E. Bareinboim, "Fairness in decision-making: The causal explanation formula," in *AAAI '18*.

[42] N. Grgic-Hlaca, E. M. Redmiles, K. P. Gummadi, and A. Weller, "Human perceptions of fairness in algorithmic decision making: A case study of criminal risk prediction," in *WWW '18*.

[43] F. P. Calmon, D. Wei, B. Vinzamuri, K. N. Ramamurthy, and K. R. Varshney, "Optimized pre-processing for discrimination prevention," in *NeurIPS '17*.

[44] N. Grgic-Hlaca, M. B. Zafar, K. P. Gummadi, and A. Weller, "Beyond distributive fairness in algorithmic decision making: Feature selection for procedurally fair learning," in *AAAI '18*.

[45] R. J. Mooney, "Comparative experiments on disambiguating word senses: An illustration of the role of bias in machine learning," in *EMNLP '96*.

[46] N. Kilbertus, M. Rojas-Carulla, G. Parascandolo, M. Hardt, D. Janzing, and B. Schölkopf, "Avoiding discrimination through causal reasoning," in *NeurIPS '17*.

[47] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *ArXiv:1712.05526*, 2017.

[48] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning," in *S&P '18*.

[49] M. Chase, E. Ghosh, and S. Mahloujifar, "Property inference from poisoning," in *Arxiv:2101.11073*, 2021.

[50] M. Mozaffari-Kermani, S. Sur-Kolay, A. Raghunathan, and N. K. Jha, "Systematic poisoning attacks on and defenses for machine learning in healthcare," in *IEEE Journal of Biomedical and Health Informatics '14*.

[51] M. Goldblum, D. Tsipras, C. Xie, X. Chen, A. Schwarzschild, D. Song, A. Madry, B. Li, and T. Goldstein, "Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses," in *ArXiv:2012.10544*, 2020.

[52] S. Moro, P. Cortez, and P. Ritaa, "A data-driven approach to predict the success of bank telemarketing," in *Decision Support Systems '14*.

[53] "Bank marketing data set," https://archive.ics.uci.edu/ml/datasets/Bank+Marketing.

[54] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," in *PETS '21*.

[55] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-party secure computation for neural network training," in *PETS '19*.

[56] R. B. D'Agostino, *Goodness-of-fit-techniques*. CRC Press, 1986, vol. 68.

[57] W. B. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mappings into a Hilbert space 26," in *Contemporary mathematics '84*.

[58] I. B. Damgård, "On the randomness of Legendre and Jacobi sequences," in *CRYPTO '88*.

[59] L. Grassi, C. Rechberger, D. Rotaru, P. Scholl, and N. P. Smart, "MPC-friendly symmetric key primitives," in *CCS '16*.

[60] D. Khovratovich, "Key recovery attacks on the Legendre PRFs within the birthday bound," in *IACR ePrint 2019/862*.

[61] A. May and F. Zweydinger, "Legendre PRF (multiple) key attacks and the power of preprocessing," in *IACR ePrint 2021/645*.

[62] P. J. Costa, *Applied mathematics for the analysis of biomedical data: Models, methods, and MATLAB*. John Wiley & Sons, 2017.

[63] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *STOC '02*.

[64] D. Evans, V. Kolesnikov, and M. Rosulek, "Defining multi-party computation," in *A Pragmatic Introduction to Secure Multi-Party Computation*, 2018.

[65] Y. Lindel, *How to simulate it: A tutorial on the simulation proof technique*, 2017.

[66] J. Kilian, "Uses of randomness in algorithms and protocols," Ph.D. dissertation, MIT, 1990.

[67] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur, "Geppetto: Versatile verifiable computation," in *S&P '15*.

[68] D. Fiore, C. Fournet, E. Ghosh, M. Kohlweiss, O. Ohrimenko, and B. Parno, "Hash first, argue later: Adaptive verifiable computations on outsourced data," in *CCS '16*.

[69] M. Campanelli, D. Fiore, and A. Querol, "LegoSNARK: Modular design and composition of succinct zero-knowledge proofs," in *CCS '19*.

[70] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," in *JACM '80*.

[71] R. Zippel, "Probabilistic algorithms for sparse polynomials," in *EUROSAM '79*.

[72] R. A. Demillo and R. J. Lipton, "A probabilistic remark on algebraic program testing," in *Information Processing Letters '78*.

[73] M. Blum, "Coin flipping by telephone a protocol for solving impossible problems," in *ACM SIGACT News '83*.

[74] C. F. Gauss, "Theoria combinationis obsevationum erroribus minimis obnoxiae," in *Carl Friedrich Gauss Werke*, 1823.

[75] D. Achlioptas, "Database-friendly random projections: Johnson-Lindenstrauss with binary coins," in *Journal of Computer and System Sciences '03*.

[76] R. B. Davies, "Algorithm as 155: The distribution of

a linear combination of χ^2 random variables," *Applied Statistics*, pp. 323–333, 1980.

[77] J. Sheil and I. O'Muircheartaigh, "Algorithm as 106: The distribution of non-negative quadratic forms in normal variables," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 26, no. 1, pp. 92–98, 1977.

[78] J.-P. Imhof, "Computing the distribution of quadratic forms in normal variables," *Biometrika*, vol. 48, no. 3/4, pp. 419–426, 1961.

[79] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity," in *ASIACRYPT '16*.

[80] M. R. Albrecht, L. Grassi, L. Perrin, S. Ramacher, C. Rechberger, D. Rotaru, A. Roy, and M. Schofnegger, "Feistel structures for MPC, and more," in *ESORICS '19*.

[81] A. Aly, T. Ashur, E. Ben-Sasson, S. Dhooghe, and A. Szepieniec, "Design of symmetric-key primitives for advanced cryptographic protocols," in *FSE '20*.

[82] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "POSEIDON: A new hash function for zero-knowledge proof system," in *SEC '21*.

[83] A. Szepieniec, "On the use of the Legendre symbol in symmetric cipher design," in *IACR ePrint 2021/984*.

[84] K. Yang, P. Sarkar, C. Weng, and X. Wang, "QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field," in *CCS '21*.

[85] "SCALE and MAMBA," https://github.com/KULeuven-COSIC/SCALE-MAMBA.

[86] "Multi-protocol SPDZ," https://github.com/data61/MP-SPDZ.

[87] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *CCS '20*.

[88] B. Strack, J. P. DeShazo, C. Gennings, J. L. Olmo, S. Ventura, K. J. Cios, and J. N. Clore, "Impact of HbA1c measurement on hospital readmission rates: Analysis of 70,000 clinical database patient records," in *BioMed Research International '14*.

[89] "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.

[90] "Real time advertiser's auction," https://www.kaggle.com/saurav9786/real-time-advertisers-auction.

[91] S. Venkatasubramanian and Q. Wang, "The Johnson-Lindenstrauss transform: An empirical study," in *Workshop on Algorithm Engineering and Experiments (ALENEX) '11*.

[92] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *STOC '88*.

[93] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *STOC '87*.

[94] A. C. Yao, "Protocols for secure computations," in *FOCS '82*.

[95] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *CRYPTO '12*.

[96] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits," in *ESORICS '13*.

[97] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making SPDZ great again," in *EUROCRYPT '18*.

[98] X. Wang, S. Ranellucci, and J. Katz, "Global-scale secure multiparty computation," in *CCS '17*.

[99] C. Hazay, P. Scholl, and E. Soria-Vazquez, "Low cost constant round MPC combining BMR and oblivious transfer," in *ASIACRYPT '17*.

[100] K. Yang, X. Wang, and J. Zhang, "More efficient MPC from improved triple generation and authenticated garbling," in *CCS '20*.

[101] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated OT with small communication," in *CCS '20*.

[102] A. Andoni, T. Malkin, and N. S. Nosatzki, "Two party distribution testing: Communication and security," *arXiv preprint arXiv:1811.04065*, 2018.

[103] V. Narayanan, M. Mishra, and V. M. Prabhakaran, "Private two-terminal hypothesis testing," in *ISIT '20*.

[104] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *STOC '85*.

[105] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit, "Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting," in *EUROCRYPT '16*.

[106] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song, "Libra: Succinct zero-knowledge proofs with optimal prover computation," in *CRYPTO '19*.

[107] J. Zhang, T. Xie, Y. Zhang, and D. Song, "Transparent polynomial delegation and its applications to zero knowledge proof," in *S&P '20*.

[108] S. Setty, "Spartan: Efficient and general-purpose zkSNARKs without trusted setup," in *CRYPTO '20*.

[109] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *S&P '13*.

[110] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou, "vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases," in *S&P '17*.

[111] R. S. Wahby, Y. Ji, A. J. Blumberg, A. Shelat, J. Thaler, M. Walfish, and T. Wies, "Full accounting for verifiable outsourcing," in *CCS '17*.

[112] A. Kosba, C. Papamanthou, and E. Shi, "xJsnark: A framework for efficient verifiable computation," in *S&P '18*.

[113] S. Setty, S. Angel, T. Gupta, and J. Lee, "Proving the correct execution of concurrent services in zero-knowledge," in *OSDI '18*.

[114] A. Bois, I. Cascudo, D. Fiore, and D. Kim, "Flexible and efficient verifiable computation on encrypted data," in *PKC '21*.

[115] F. R. Hampel, E. M. Ronchetti, P. J. Rousseeuw, and W. A. Stahel, *Robust statistics: The approach based on influence functions*. John Wiley & Sons, 2011, vol. 196.

[116] P. J. Huber, *Robust statistics*. John Wiley & Sons, 2004, vol. 523.

[117] R. A. Maronna, R. D. Martin, V. J. Yohai, and M. Salibián-Barrera, *Robust statistics: Theory and methods (with R)*. John Wiley & Sons, 2019.

[118] J. Steinhardt, P. W. Koh, and P. Liang, "Certified defenses for data poisoning attacks," in *NeurIPS '17*.

[119] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! Targeted clean-label poisoning attacks on neural networks," in *NeurIPS'18*.

[120] I. Diakonikolas and D. M. Kane, "Recent advances in algorithmic high-dimensional robust statistics," in *Arxiv:1911.05911*, 2019.

[121] M. Goldblum, D. Tsipras, C. Xie, X. Chen, A. Schwarzschild, D. Song, A. Madry, B. Li, and T. Goldstein, "Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses," in *Arxiv:2012.10544*, 2020.

[122] A. Paudice, L. Munoz-González, and E. C. Lupu, "Label sanitization against label flipping poisoning attacks," in *ECML PKDD '18*.

[123] A. Paudice, L. Muñoz-González, A. Gyorgy, and E. C. Lupu, "Detection of adversarial training examples in poisoning attacks through anomaly detection," in *Arxiv:1802.03041*, 2018.

[124] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC '06*.

[125] C. Dwork, "Differential privacy," in *ICALP '06*.

[126] C. Dwork and A. Roth, *The Algorithmic Foundations of Differential Privacy*. Now Publishers, 2014.

[127] A. Narayan and A. Haeberlen, "DJoin: Differentially private join queries over distributed databases," in *OSDI '12*.

[128] A. Narayan, A. Feldman, A. Papadimitriou, and A. Haeberlen, "Verifiable differential privacy," in *EuroSys '15*.

[129] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *CCS '15*.

[130] E. Roth, D. Noble, B. H. Falk, and A. Haeberlen, "Honeycrisp: Large-scale differentially private aggregation without a trusted core," in *SOSP '19*.

[131] E. Roth, H. Zhang, A. Haeberlen, and B. C. Pierce, "Orchard: Differentially private analytics at scale," in *OSDI '20*.

[132] F. McKeen, I. Alexandrovich, A. Berenzon, C. Rozas, H. Shafi, V. Shanbhogue, and U. Savagaonkar, "Innovative instructions and software model for isolated execution," in *HASP '13*.

[133] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Opaque: An oblivious and encrypted distributed analytics platform," in *NSDI '17*.

[134] C. Priebe, K. Vaswani, and M. Costa, "EnclaveDB: A secure database using SGX," in *S&P '18*.

[135] S. Eskandarian and M. Zaharia, "ObliDB: Oblivious query processing for secure databases," in *VLDB '19*.

[136] F. Shaon, M. Kantarcioglu, Z. Lin, and L. Khan, "SGX-BigMatrix: A practical encrypted data analytic framework with trusted processors," in *CCS '17*.

[137] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *S&P '15*.

[138] J. V. Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution," in *SEC '17*.

[139] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX," in *CCS '17*.

[140] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *SEC '18*.

[141] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution," in *EuroS&P '19*.

[142] H. Ragab, A. Milburn, K. Razavi, H. Bos, and C. Giuffrida, "CROSSTALK: Speculative data leaks across cores are real," in *S&P '21*.

[143] S. van Schaik, M. Minkin, A. Kwong, D. Genkin, and Y. Yarom, "CacheOut: Leaking data on Intel CPUs via cache evictions," in *S&P '21*.

[144] S. van Schaik, A. Kwong, D. Genkin, and Y. Yarom, "SGAxe: How SGX fails in practice," https://sgaxe.com/files/SGAxe.pdf.

[145] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure Tensorflow inference," in *S&P '20*.

[146] W. Beullens, T. Beyne, A. Udovenko, and G. Vitto, "Cryptanalysis of the Legendre PRF and generalizations," in *FSE '20*.

[147] N. Kaluđerović, T. Kleinjung, and D. Kostić, "Cryptanalysis of the generalised Legendre pseudorandom function," in *ANTS '20*.

[148] M. O. Rabin, "Probabilistic algorithms in finite fields," in *SIAM Journal on Computing '80*.

[149] C. F. Gauss, *Carl Friedrich Gauss' Untersuchungen uber hohere Arithmetik*, 1889.

[150] S. K. Chebolu and J. Mináč, "Counting irreducible polynomials over finite fields using the inclusion-exclusion principle," in *Mathematics Magazine '11*.

[151] C. Ding, T. Hesseseth, and W. Shan, "On the linear complexity of Legendre sequences," in *IEEE TIT '98*.

[152] V. Tóth, "Collision and avalanche effect in families of pseudorandom binary sequences," in *Periodica Mathematica Hungarica '07*.

[153] C. Mauduit and A. Sárközy, "On finite pseudorandom

binary sequences i: Measure of pseudorandomness, the Legendre symbol," in *Acta Arithmetica '97*.
[154] "Legendre pseudo-random function," https://legendreprf.org/.
[155] I. A. Seres, M. Horváth, and P. Burcsi, "The Legendre pseudorandom function as a multivariate quadratic cryptosystem: Security and applications," in *IACR ePrint 2021/182*.
[156] A. Dasgupta, R. Kumar, and T. Sarlos, "A sparse Johnson-Lindenstrauss transform," in *STOC '10*.
[157] D. M. Kane and J. Nelson, "A derandomized sparse johnson-lindenstrauss transform," *ArXiv:1006.3585*, 2010.
[158] H. Corrigan-Gibbs and D. Kogan, "The discrete-logarithm problem with preprocessing," in *EUROCRYPT '18*.

APPENDIX

A. Parameters of Legendre PRF for HOLMES's dimensionality reduction

In HOLMES, dimensionality reduction requires the sampling of a pseudorandom matrix A as described in Section VI. We provide details regarding the key generation for Legendre PRF and other necessary background for Legendre PRF.

Sampling the Legendre PRF key. The t parties use the same JL matrix, and therefore the same set of PRF keys. Before sampling this random matrix, all parties must have already committed their input data in the ZK and SMPC modules. This is to prevent a malicious party from adaptively adjusting their input data according to the random matrix.

The parties first use a random coin toss protocol to agree on a freshly generated pseudorandom seed. Then, using this pseudorandom seed as source of randomness, they run the following PRF key generation algorithm. In the Legendre PRF, the protocol needs to sample a random key $K = (k_0, k_1, ..., k_d)$ such that $f_K(x)$ is irreducible in $\mathbb{Z}_p^*$ and has no nontrivial stabilizer, for each row of matrix A. This requirement has arisen recently [146, 147] as a safeguard against known attacks on Legendre PRF.

The parties sample a random degree-d polynomial and check its reducibility using Rabin's irreducibility test [148]. We can expect to find an irreducible polynomial with a probability of about $1/d$ [149, 150].

We skip the testing for nontrivial stabilizer, as it is known to be very inefficient, and for $d \geq 3$, a random polynomial over $\mathbb{Z}_p^*$ has nontrivial stabilizers only with a probability of at most $9/p$ [147]. One could, following the recommendations in [147], avoid nontrivial stabilizers from the beginning by having $\gcd(p^2 - 1, d) = 1$. But, for our choice of parameter $d = 3$, this condition cannot be satisfied for any prime larger than 3. For $d = 4$, it is also impossible because our prime p is chosen such that $p - 1$ has some two-arity (for the SMPC to be able to use the Low Gear protocol).

Pseudorandomness of Legendre PRF. The protocol relies on the pseudorandomness of the Legendre sequence generated by the PRF. Though there is no known reduction to standard cryptographic assumptions, it has been conjectured, like other symmetric key primitives, to have such a property. The pseudorandomness of Legendre PRF has been studied in many different aspects, including linear complexity, collision, avalanche effect, and so on [151–153]. There is a recent trend to study Legendre PRF as a SMPC-friendly PRF function [59, 60, 146, 147, 154, 155]. Note that JL lemma does not require a very high level of pseudorandomness—hash functions with bounded independence already suffice [156, 157]. We only need to ensure that the hash functions are chosen independently from the input data, which we guarantee by committing the data before sampling PRF keys.

Recently, there are several works on key-recovery attacks on Legendre PRF [60, 146, 147]. They are indeed not relevant to our settings because our protocol never hides the Legendre PRF keys, and our protocol 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 we have a modest security margin against future attacks on pseudorandomness. PRF distinguishing attacks on Legendre PRF that does not base on key-recovery attacks and is faster than a direct use of key-recovery attacks is an open problem with no positive result so far [61], even with state-of-the-arts attack techniques [158]. Ethereum foundation currently has an active bounty program [154] on this open problem.

In our use case where the party has already committed their input, it suffices to choose $d = 3$ for statistical security.

B. Security proof sketches

Theorem 1. *Under standard cryptographic assumptions and static corruptions, and under the random oracle and the ideal cipher models, the protocol of HOLMES securely realizes the ideal functionality* $\mathcal{F}_{\mathsf{HOLMES}}$.

Proof sketch. We prove the security in the $(\mathcal{F}_{\mathsf{CP}}, \mathcal{F}_{\mathsf{SFE}})$-hybrid world, where a commit-and-prove zero-knowledge proof ideal functionality $\mathcal{F}_{\mathsf{CP}}$ models the ZK module, and a secure function evaluation ideal functionality $\mathcal{F}_{\mathsf{SFE}}$ models the SMPC module. We also need to invoke a random oracle for subsampling, and/or an ideal cipher for our dimensionality reduction to model Legendre PRF. For ease of presentation, we focus on the part involving $(\mathcal{F}_{\mathsf{CP}}, \mathcal{F}_{\mathsf{SFE}})$ in this proof sketch.

We assume that the adversary corrupts at least one party, otherwise the adversary does not even know what the checklist is. And for simplicity, we assume malicious parties do not abort in the middle of the protocol. The simulator in the ideal world can simulate the transcripts of HOLMES's protocol, by invoking the corresponding simulators $\mathsf{Sim}_{\mathsf{CP}}$ and $\mathsf{Sim}_{\mathsf{SFE}}$, on the corresponding computation as well as the consistency check. Note that, during the consistency check, the simulator can easily evaluate the polynomial at a random point $z = \beta$ sampled by a simulated coin toss. Therefore, the simulator is able to simulate the view of the hybrid world. By using the hybrid arguments, we can show that HOLMES's protocol securely realizes the ideal functionality $\mathcal{F}_{\mathsf{HOLMES}}$. □