Drynx: Decentralized, Secure, Verifiable System for Statistical Queries and Machine Learning on Distributed Datasets

David Froelicher, Juan R. Troncoso-Pastoriza, Joao Sa Sousa and Jean-Pierre Hubaux

Abstract

Data sharing has become of primary importance in many domains such as big-data analytics, economics and medical research, but remains difficult to achieve when the data are sensitive. In fact, sharing personal information requires individuals' unconditional consent or is often simply forbidden for privacy and security reasons. In this paper, we propose Drynx, a decentralized system for privacy-conscious statistical analysis on distributed datasets. Drynx relies on a set of computing nodes to enable the computation of statistics such as standard deviation or extrema, and the training and evaluation of machine-learning models on sensitive and distributed data. To ensure data confidentiality and the privacy of the data providers, Drynx combines interactive protocols, homomorphic encryption, zero-knowledge proofs of correctness, and differential privacy. It enables an efficient and decentralized verification of the input data and of all the system's computations thus provides auditability in a strong adversarial model in which no entity has to be individually trusted. Drynx is highly modular, dynamic and parallelizable. Our evaluation shows that it enables the training of a logistic regression model on a dataset (12 features and 600,000 records) distributed among 12 data providers in less than 2 seconds. The computations are distributed among 6 computing nodes, and Drynx enables the verification of the query execution's correctness in less than 22 seconds.

Index Terms

decentralized system, distributed datasets, privacy, statistics, machine learning, homomorphic encryption, zero-knowledge proofs, differential privacy.

I. Introduction

To produce meaningful results, statistical and machine-learning analyses often demand large amounts of data. Although data storage and computation costs have dropped over the years, notably due to low-cost and powerful cloud-computing solutions, the sharing of these data is still cumbersome. Massive amounts of data are generated daily to track individuals' actions, health, shopping habits, interests, political and religious views [1], but privacy concerns and ethical/legal constraints often prohibit or discourage the sharing of personal and sensitive data. In Europe, the new data-protection regulation, General Data Protection Regulation (GDPR) [2], effective since May 2018, requires that (a) the collection and use of personal data can only be done with the consent of the subject and (b) that the data have to be anonymized or encrypted before being shared. This leads to a conundrum, especially in domains such as demography, finance and health, where data have to be shared, e.g., for enabling research, but they also need to be protected to ensure individuals' fundamental right to privacy. Cross-border data sharing is even more challenging, as the legislations among countries can be heterogeneous, forcing companies to geographically adapt their own privacy measures.

Multiple examples show that even when data can be shared, a centralization of the data can have serious consequences, affecting hundreds of millions of individuals [3], [4]; this was the case with the Equifax breach [4], in which personal information (including social-security numbers and credit-card information) of more than 143 million consumers (about 40% of the US population) was compromised. Centralized solutions are subject to multiple threats as the central database, which stores data from multiple mutually-untrusted sources, constitutes a high-value target for possible attackers and a single point of failure.

Existing solutions for secure databases [5], [6], [7], [8], [9] usually add a cryptographic layer on top of the query engine or focus exclusively on the data-release privacy, e.g., by using differential privacy. However, most of these solutions have a significant performance overhead or are still fully centralized hence either have a single point of failure, or do not protect the data during the query execution.

D. Froelicher is with the LCA1 and DeDiS Laboratories, Ecole Polytechnique Federale de Lausanne, 1015 Lausanne, Switzerland, e-mail: david.froelicher@epfl.ch.

J. R. Troncoso-Pastoriza, Joao Sa Sousa and Jean-Pierre Hubaux are with the LCA1 Laboratory, Ecole Polytechnique Federale de Lausanne, 1015 Lausanne, Switzerland, e-mail: name.surname@epfl.ch.

This work was partially supported by the grant $\#2017-201$ of the Strategic Focal Area "Personalized Health and Related Technologies (PHRT)" of the ETH Domain.

In this context, decentralized data-sharing systems [10], [11], [12], [13], [14], [15] have raised considerable interest and are key enablers for privacy-conscious big-data analysis. By distributing the storage and the computation, thus avoiding single points of failure, these systems enable data sharing and minimize the risks incurred by centralized solutions. Nevertheless, many of these systems rely on honest-but-curious or trusted third-party assumptions that might not provide sufficient guarantees when the data to be shared are highly sensitive, valuable, influential or private. Other solutions with stronger threat models, e.g., UnLynx [16], are limited in the computations they support, e.g., sum only. Moreover, none of these solutions considers the possibility that both computing entities and data providers can be malicious.

Improving upon and using some techniques introduced in UnLynx, we propose Drynx, an operational, decentralized and secure system that enables queriers to compute statistical functions and to train and evaluate machine-learning models on data hosted at different sources, i.e., on distributed datasets. Drynx ensures data confidentiality, data providers' (DPs) privacy and protects individuals' data from potential inferences stemming from the release of end results, i.e., it ensures differential privacy. It also provides computation correctness. Finally, it also ensures that strong outliers, either maliciously or erroneously input by DPs, cannot influence the results beyond a certain limit, and we denote this by *results robustness*. These guarantees are ensured in a strong adversarial model where no entity has to be individually trusted and a fraction of the system's entities can be malicious. Drynx relies on interactive protocols, homomorphic encryption, zero-knowledge proofs of correctness and distributed differential privacy. It is scalable, dynamic and modular: Any entity can leave or join the system at any time and Drynx offers security features or properties that can be enforced depending on the application, e.g., differential privacy.

In this paper, we make the following contributions:

- We propose Drynx, an efficient, modular and parallel system that enables privacy-preserving statistical queries and the training and evaluation of machine-learning regression models on distributed datasets.
- We present a system that provides data confidentiality and individuals' privacy, even in the presence of a strong adversary. It ensures the correctness of the computations, protects data providers' privacy and guarantees robustness of query results.
- We propose techniques that enable full and lightweight auditability of query execution. Drynx relies on a new efficient distributed solution for storing and verifying proofs of query validity, computation correctness, and input data ranges. We exemplify and evalutate the implementation of this solution by using a blockchain.
- We propose and implement an efficient, modular and multi-functionality query-execution pipeline by
  - introducing *Collective Tree Obfuscation*, a new distributed protocol that enables a collective and verifiable obfuscation of encrypted data;
  - presenting multiple data-encoding techniques that enable distributed computations of advanced statistics on homomorphically encrypted data. We propose new encodings, and improvements and adaptations of previously introduced private-aggregation encodings to our framework and security model;
  - adapting an existing zero-knowledge scheme for input-range validation to our security model;
  - proposing a new construction of the *Key Switching* protocol introduced in UnLynx [16], improving both its performance and capabilities.

To the best of our knowledge, Drynx is the only operational system that provides the aforementioned security and privacy guarantees. Drynx implementation is fully available at www.github.com/ldsec/drynx.

II. Related Work

Centralized systems for privacy-preserving data sharing [8], [17], [18], [19] and trusted-hardware based solutions [20] usually require one entity, i.e., a central entity or a hardware provider, to be trusted, which constitutes a single point of failure. Even though these systems can be more efficient than their decentralized counterparts, they often require a centralization or outsourcing of the data storage, which goes against regulations or is cumbersome to achieve [21] and can be inappropriate for sensitive data. In Drynx, we avoid these issues by decentralizing data-storage, computation and correctness verification, thus efficiently distributing trust.

In order to execute queries and compute statistics on distributed datasets, multiple decentralized solutions [10], [12], [14], [22], [23], [24], [25] rely on techniques that have a high expressive power, such as secret sharing and garbled circuits. These solutions are often flexible in the computations they offer but usually assume (a) honest-but-curious computing parties and (b) no collusion or a 2-party model. Furthermore, they do not provide a way to check the computations undertaken in the system. Although they might efficiently distribute trust, their strong honesty assumptions are risky when the data or the computed statistics are highly sensitive. Bater et al. [10] enable the evaluation of various SQL queries on datasets hosted by a set of distrustful data providers, but both the data providers and the computing entity are trusted to follow the protocol. Corrigan-Gibbs and Boneh [26] propose Prio, a system that ensures privacy as long as one computing entity out of n is honest, but it only guarantees end results robustness in the case where the involved parties are all honest-but-curious. Moreover, Prio does not protect against DPs colluding among themselves or with the computing nodes. In Drynx, no entity has to be individually trusted in order to provide both privacy and robustness.

Systems relying on homomorphic encryption [11], [13], [16], [27], [28], [29] are often limited in the functionalities they offer (e.g., sum only). They present high-performance overhead in comparison with their less secure counterparts or still rely on honest-but-curious parties. In our previous work, we presented UnLynx [16], a decentralized system that enables the computation of (only) sums on distributed datasets and ensures DPs' privacy and data confidentiality. UnLynx assumes DPs to be honest-but-curious and, unlike Drynx, it does not ensure end results robustness. Moreover, UnLynx does not provide a practical solution for auditability. In this work, we show how to overcome these limitations and provide a system that enables secure computations of multiple operations in a stronger threat model.

There are multiple solutions proposed for the problem of training machine-learning models on distributed data in a privacy-preserving way [13], [27], [30], [31], [32], [33], [34], [35], [36]. Mohassel and Zhang [30] propose a two-party solution, SecureML; it enables the training of specific models, e.g., linear regression. Boura et al. [31] present a solution that relies on a novel and more flexible approximation of the logistic regression function but assumes honest-but-curious parties. Nikolaenko et al. [27] and Juvekar et al. [32] combine homomorphic encryption and garbled circuits to perform private ridge-regression and neural-network inference, respectively. Aono et al. [33] and Kim et al. [13] rely on homomorphic encryption to train an approximated logistic regression function. Zheng et al. [36] combine homomorphic encryption and distributed convex optimization, in their system called Helen, in order to collaboratively train linear models. Recently, multiple solutions based on federated learning (relying on differential privacy and edge computing) have been proposed [24], [37], [38], [39], [40], [41], [42]. These solutions aim at protecting the resulting model from inference attacks [43], [44]. Some of these works [37], [39] assume a trusted party that holds the data, trains the machine-learning model, and performs the noise addition to achieve differential privacy guarantees. Other works [24], [29], [38], [45], [46] propose solutions for distributed settings in which the parties exchange differentially private model parameters with the help of an untrusted server that trains a collective global model. These approaches are computationally efficient but usually require very high privacy budgets to obtain a useful collective model (due to the noise addition); hence it is unclear what privacy protection they achieve in practice [47]. To this end, some works attempt to obtain more useful models in the distributed setting by combining differential privacy with homomorphic encryption [40], [41] or multi-party computation techniques [42]. However, most of these solutions are specifically tailored, parameterized and optimized for a given operation, e.g. gradient descent, and would require a redesign if used for different operations. Finally, they assume a weaker threat model with honest-but-curious computing parties and, unlike Drynx, they do not enable verification of computation correctness and results robustness.

III. Background

We introduce Drynx's main components and two exemplifying use cases. We describe the cryptographic tools that we use to distribute trust and workload. We present the blockchains that we use to implement our solution to ensure Drynx's correctness and auditability. Finally, we introduce the notion of differential privacy and verifiable shuffle, which are at the core of our solution to ensure individuals' privacy.

A. Use Cases

We illustrate Drynx's utility in the medical sector, as it is a paradigmatic example where privacy is paramount and data sharing is needed. Recently, multiple initiatives have emerged to realize the promise of personalized medicine and to address the challenges posed by the increasing digitalization of medical data [48], [49], [50]. In this context, the ability to share highly sensitive medical data while protecting patients' privacy is becoming of primary importance. We illustrate the possible use of Drynx in two specific settings that cover most medical data sharing scenarios: (1) Hospital Data Sharing (HDS), where multiple hospitals enable statistical computations and the training of machine-learning models across their datasets of patients (e.g., [50], [51]), and (2) Personal Data Sharing (PDS), where a medical institute runs studies, e.g., on heart issues, by directly computing on data collected from people's wearables (e.g., [52], [53]).

B. ElGamal Homomorphic Encryption

Drynx requires an additively homomorphic cryptosystem; we choose to rely on the Elliptic Curve ElGamal (ECEG) [54], which enables an efficient use of zero-knowledge proofs for correctness [55]. However, Drynx's functionality is not bound to this choice and can be achieved with other cryptosystems. ECEG relies on the difficulty of computing a discrete logarithm in a finite field; in this case, an Elliptic Curve subgroup of $\mathbb{Z}_p$, with p a big prime. The encryption of a message $m \in \mathbb{Z}_p$ is $\mathrm{E}_\Omega(m) = (rB, mB + r\Omega)$, where r is a uniformly-random nonce in $\mathbb{Z}_p$, B is a base point on an elliptic curve $\mathcal{G}$ and Ω a public key. The table of symbols is presented in Appendix A. The additive homomorphic property states that $\mathrm{E}_\Omega(\alpha m_1 + \beta m_2) = \alpha \mathrm{E}_\Omega(m_1) + \beta \mathrm{E}_\Omega(m_2)$ for any messages m_1 and m_2 and for any scalars α and β. In order to decrypt a ciphertext $(rB, mB + r\Omega)$, the holder of the corresponding private key ω ($\Omega = \omega B$) multiplies rB and ω yielding $\omega(rB) = r\Omega$ and subtracts this point from $mB + r\Omega$. The result mB is then mapped back to m, e.g., by using a hashtable. Drynx relies on fixed-point representation to encrypt floating values.

C. Zero-Knowledge Proofs

Universally-verifiable zero-knowledge proofs (ZKPs) can be used to ensure computation integrity and to prove that encrypted data are within given ranges. In Drynx, we choose to verify computation integrity by using the proofs for general statements about discrete logarithms, introduced by Camenisch and Stadler [55]. These proofs enable a verifier to check that the prover knows the discrete logarithms y_1 and y_2 of the public values $Y_1 = y_1 B$ and $Y_2 = y_2 B$ and that they satisfy a linear equation

$$A_1 y_1 + A_2 y_2 = A, \tag{1}$$

where A, A_1, A_2 are public points on $\mathcal{G}$. This is done without revealing any information about y_1 or y_2.
The input-range validation is done by relying on the proofs proposed by Camenisch and Chaabouni [56], with which we can prove that a secret message m lies in a given range $[0,u^l)$ with u and l integers, without disclosing m. The prover writes the base-u decomposition of its secret value m and commits to the u-ary digits by using the verifier signatures on these digits. The l created commitments prove to the verifier that $m \in [0,u^l)$. We present this proof, adapted to our framework, in Algorithm 1. Finally, both proofs can be made non-interactive through the Fiat-Shamir heuristic [57].

D. Interactive Protocols

Interactive protocols can be used to distribute the computations and the trust among multiple computing nodes CNs. In Drynx, each CN_i possesses a private-public key pair (k_i, K_i) where k_i is a uniformly-random scalar in $\mathbb{Z}_p$ and $K_i = k_i B$ is a point on $\mathcal{G}$. The CNs' collective public key is $K = \sum_{i=1}^{\#CN} K_i$. The corresponding secret key $k = \sum_{i=1}^{\#CN} k_i$ is never reconstructed such that a message encrypted by using K can be decrypted only with the participation of all CNs. An attacker would have to compromise all CNs in order to decrypt a message. As shown in Section V, to produce the intended results, Drynx protocols require the participation of all the CNs.

E. Blockchains

A *blockchain* is usually a public, append-only ledger that is distributively maintained by a set of nodes and serves as an immutable ledger [58], [59]. Its main applications are in cryptocurrencies [59], [60] but is also used in other domains, e.g., health care [61]. Data are bundled into blocks that are validated through the consensus [62], [63] of the maintaining nodes. Each block contains a pointer (i.e., a cryptographic hash) to the previous valid block, a timestamp, a nonce, and application-specific data. The chain of these blocks forms the blockchain.

F. Differential Privacy

Differential privacy is an approach for privacy-preserving reporting results on statistical datasets, introduced by Dwork [64]. This approach guarantees that a given randomized statistic, $\mathcal{M}(DS) = R$, computed on a dataset DS, behaves similarly when computed on a neighbor dataset DS' that differs from DS in exactly one element. More formally, (ϵ, δ)-differential privacy [65] is defined by $\Pr[\mathcal{M}(DS) = R] \leq \exp(\epsilon) \cdot \Pr[\mathcal{M}(DS') = R] + \delta$, where ϵ and δ are privacy parameters: the closer to 0 they are, the higher the privacy level is. (ϵ, δ)-differential privacy is often achieved by adding noise to the output of a function $f(D)$. This noise can be drawn from the Laplace distribution with mean 0 and scale $\frac{\Delta f}{\epsilon}$, where Δf, the sensitivity of the original real valued function f, is defined by $\Delta f = \max_{D,D'} ||f(DS) - f(DS')||_1$. Other mechanisms, e.g., relying on a Gaussian distribution, have also been proposed [66], [67].

G. Verifiable Shuffles

To randomly select a value from a public list of noise values and to ensure differential privacy, we rely on a verifiable shuffle [68], [69], [70], [71]. We implemented and use the verifiable shuffle of ElGamal pairs, described by Neff [69]. This protocol takes as input a list of χ ElGamal pairs $(C_{1,i}, C_{2,i})$ and outputs $(\bar{C}_{1,i}, \bar{C}_{2,i})$ pairs such that for all $1 \leq i \leq \chi$, $(\bar{C}_{1,i}, \bar{C}_{2,i}) = (C_{1,\upsilon(i)} + r''_{\upsilon(i)} B, C_{2,\upsilon(i)} + r''_{\upsilon(i)} \Omega)$, where $r''_{\upsilon(i)}$ is a re-randomization factor, υ is a permutation and Ω is a public key. The permutation υ is used to change the order of the ElGamal pairs and $r''_{\upsilon(i)}$ is used to modify the value of the ciphertext encrypting a message m such that its decryption still outputs m. As a result, an adversary not knowing the decryption key, υ and the $r''_{\upsilon(i)}$ is unable to link back any ciphertext $(\bar{C}_{1,i}, \bar{C}_{2,i})$ with a ciphertext $(C_{1,i}, C_{2,i})$. Neff provides a method for proving that such a shuffle is done correctly, i.e., that there exists a permutation υ and re-randomization factors $r''_{i,j}$ such that $output = \textsc{shuffle}_{\upsilon, r''_{i,j}}(input)$, without revealing anything about υ or $r''_{i,j}$. This is achieved by using honest-verifier zero-knowledge proofs, introduced by Neff [68], [69].

IV. System Overview

In this section, we describe the system and threat models, before presenting Drynx's functionality and security requirements.

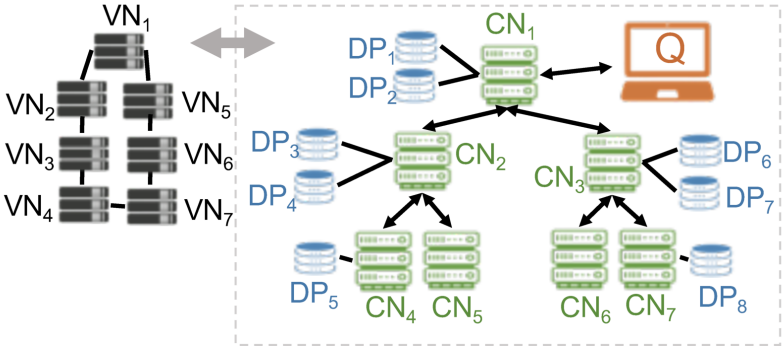


Fig. 1: A querier Q, Data Providers DP_i, Computing Nodes CN_i and Verifying Nodes VN_i.

A. System Model

The system model is represented in Figure 1. For simplicity, we describe here the logical roles in Drynx, and in Section VIII we discuss the fact that a physical node can simultaneously play multiple roles. A querier Q can execute a statistical query and the training and evaluation of a machine-learning model on distributed datasets held by DPs. The CNs collectively handle the computations in the system; i.e., from Q's perspective, they emulate a central server and provide answers to her queries. The verifying nodes' (VNs) role is to provide auditability; they collectively verify the query execution and immutably store the corresponding proofs. They enable an auditor, e.g., Q or an external entity, to easily verify (audit) the correctness of the query execution.

In Drynx's typical workflow, the query is defined by the querier Q and is then broadcast to the CNs and DPs. The DPs answer with their encrypted responses that are then collectively aggregated and processed by the CNs, before the result is sent to Q. We assume that the used data formats are sufficiently homogeneous among different DPs and that the DPs are able to interpret the queries, e.g., there is a common ontology of attributes and the query-language is agreed-on during system setup.

An exemplifying instantiation of this system model in the HDS scenario (Section III-A) would feature the CNs as universities that want to enable researchers (Q) to compute on data held by multiple hospitals (DPs). VNs can be independent or governmental institutions ensuring that data protection regulations are respected.

We assume that the system's topology and public information, e.g., public keys, are known by all entities. Authentication and authorization are out of scope of this paper and we briefly discuss them in Section VIII.

B. Threat Model

We assume a strong threat model:

- *Queriers*. They are considered malicious as they can try to infer information about the DPs from the queries end results or by colluding with other entities in the system.
- *Computing Nodes*. We consider an Anytrust model [72], which means that all Drynx's security and privacy guarantees (Section IV-D) are ensured, as long as at least one of the CNs is honest-but-curious (or plain honest).
- *Data Providers*. The DPs are considered malicious as they can try to produce an incorrect answer to a query in order to bias the final results. They can also collude with other nodes to infer information about other DPs or about a query end-results.
- *Verifying Nodes*. We assume that a threshold number of the VNs is honest. This threshold, e.g., $f_h = 2f+1$ out of $f_t = 3f+1$, where f_t is the number of VNs, is defined depending on the consensus algorithm [62], [63] that is used to ensure a correct and immutable storage of the proofs' verification results.

C. Functional Requirements

Drynx enables the computation on distributed datasets of any operation in the family of *encodable operations*. An *encodable operation* can be separated in two parts: the DPs' local computations and the collective aggregation. In the collective part, the computations are executed on encrypted data and are thus limited by the homomorphism in the used cryptographic scheme, e.g., additions and/or multiplications. DPs' computations are executed locally and are therefore not limited.

Definition 1. *An* encodable operation f *computed among* N DPs *is defined by:*

$$f(\bar{r}) \equiv \pi(\{\rho(\bar{r_i})\}_{i=1}^{N}),$$

in which the encoding ρ *is defined by*

$$\rho(\bar{r_i}) \equiv (\mathbf{V_i}, c_i),$$

where $\mathbf{V_i} = [v_{i,1},...,v_{i,d}]$ *is a vector of* d *values computed on a set of* $c_i = |\bar{r_i}|$ *records, where* $|.|$ *stands for cardinality.* $\bar{r}$ *is the set of all distributed datasets' records,* $\bar{r_i}$ *is the set of records that belong to* DP_i*, and* π *is a polynomial combination of the*

outputs of the encodings ρ. The encodings are defined as locally computed functions on the subsets ($\bar{r_i}$) of each DP_i. It is also possible to express an encodable operation as a recursive function:

$$f_k(\bar{r}) \equiv \pi(\{\rho(\bar{r_i}, f_{k-1}(\bar{r}))\}_{i=1}^{N}).$$

In Drynx, for any specific operation f, each DP_i creates an encoding ρ computed on its set of records $\bar{r_i}$. Then, π is executed in two parts: the CNs first aggregate all DPs' encodings outputs ($\sum_{i=1}^{N}\{\rho(\bar{r_i})\}$) and, if needed, the querier post-processes π on the aggregated result (e.g, if π involves information-preserving operations not executable by the CNs under homomorphic encryption).

We give here an instantiation of Definition 1 that enables the computation of the `average`, and in Section VII we show how an *encoding* can be instantiated to enable the computation of: `sum`, `count`, `frequency count`, `average`, `variance`, `standard deviation`, `cosine similarity`, `min/max`, `AND/OR` and `set intersection/union`, and the training and evaluation of `linear` and `logistic regression` models.

For example, if Q wants to compute the `average` (f) heart rate over multiple patients across hospitals (HDS (Section III-A)), each hospital (DP_i) answers with the encoding of its (encrypted) local sum of each patient's heart rate (h): $\rho(\bar{r_i}) \equiv ([\sum_{j=1}^{c_i} h_{i,j}], c_i)$. These encodings are then (homomorphically) added across all hospitals, and Q can (decrypt and) compute the global average by using $\pi = \sum_{i=1}^{N} v_{i,1} / \sum_{i=1}^{N} c_i$. We remark here that whereas ρ and π are application dependent, the workflow is common to all the possible operations.

Finally, in Drynx, an auditor can efficiently audit a query execution. Moreover, the proofs required for auditability are produced such that their creation does not affect the query runtime.

D. Security Requirements

Drynx must ensure:

- *Data confidentiality*. The data input by the DPs have to remain confidential at any time. Only Q is able to see the query answer.
- *DPs' privacy*. No entity is able to infer information about one single DP or about any individual storing his data in a DP's database.
- *Query Execution Correctness.* We consider the query execution to be correct when both *results robustness* and *computation correctness* requirements are met:
  - *Results robustness*. The query results are protected against strong outliers, either maliciously or erroneously input by the DPs.
  - *Computation correctness.* Any computation undertaken by the CNs is correctly executed.

V. Drynx Design

To overcome the limitations in existing works and meet the requirements presented in the previous section, we propose a novel system model in which we enable query auditability by introducing VNs. Additionally, Drynx provides multiple functionalities in a stronger threat model by relying on DPs that encode locally computed results proven to be within a certain range. It limits the trust in DPs by controlling that their results are in these pre-defined ranges. We propose a system that remains generic and practical while operating in a threat model stronger than existing works. We discuss now the design of this system.

In *Drynx's Security Design* (Section V-A), we show how we build Drynx to meet all its security requirements:

- In Section V-A1, we introduce a simple query-execution pipeline enabling Drynx's functionalities and protecting data confidentiality.
- In Section V-A2, we build upon the previously introduced query-execution pipeline and explain how to ensure DPs' privacy by introducing the new concept of a *neutral encoding*. This enables a DP to privately choose whether to answer a query. We also explain how Drynx handles bit-wise operations and maintains DPs' privacy. Finally, we introduce distributed differential privacy that is used to ensure that no entity infers information about a single DP or individual from the query end results.
- In Section V-A3, we show how we provide auditability in an efficient way by relying on a set of VNs. We describe how Drynx ensures results robustness by leveraging on range proofs and how all Drynx's computations can be verified by relying on proofs of correctness.

In *Drynx's Optimized Design* (Section V-B), we discuss how to optimize Drynx's performance:

- In Section V-B1, we present Drynx's full query-execution pipeline. We show how multiple parts of the query execution and verification can be run concurrently thus optimize Drynx's runtime.
- In Section V-B2, we introduce a tradeoff between security and performance by enabling a probabilistic verification of the query execution.

A. Drynx Security Design

We present Drynx core security architecture.

1) **Data Confidentiality:** First, we introduce a confidential distributed data-sharing system (Figure 2) that can run the same operations as Drynx, but only meets one of the security requirements: *data confidentiality*.

We describe the query execution protocol, and sketch the proof of confidentiality for this system. Afterwards, we describe how to enhance this construction to meet Drynx's other security requirements without breaking data confidentiality.

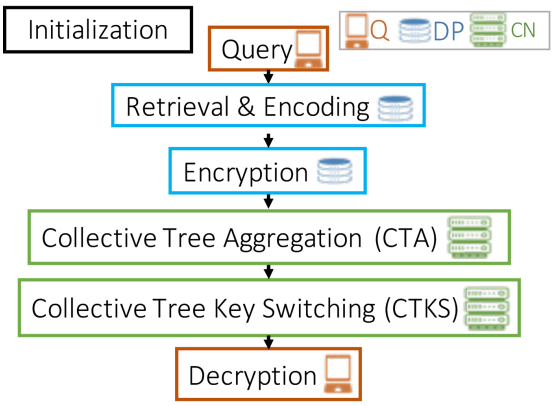


Fig. 2: Confidential System Query Execution.

1) *Initialization.* Each CN_i,DP_i and Q_i generates its own private-public key-pair (k_i,K_i). The CNs' public keys are then summed up in order to create K, the CNs' public collective key that is used to encrypt all the processed data.
2) *Query.* Q formulates the query that is broadcast in clear through the CNs to the DPs. Although the querier could directly communicate with the DPs, our choice simplifies the communication scheme and the synchronisation inside the system, as the CNs have to know the query and receive the DPs inputs to perform the computations in the remaining steps. The query defines the operation, the attributes on which the operation is computed, the participating DPs and (optionally) the filtering conditions. Drynx works independently of the query language. We illustrate its use with a SQL-like query to compute the `average` heart rate among patients for which data are held by n DPs:
   `SELECT average` *heart_rate* `ON` DP_1, ..., DP_n
   `WHERE` *patient_state* $=$ *'hypertensive'*
3) *Retrieval & Encoding*. The DPs compute their local answer by following ρ which is defined in the operation *encoding* (Definition 1). For this purpose, they first locally retrieve the corresponding data.
4) *Encryption*. The DPs encrypt their encoded answer under K and send the corresponding ciphertexts back to the CNs.
5) *Collective Tree Aggregation (CTA).* The CNs collectively aggregate all DPs' responses by executing a CTA protocol relying on the *Collective Aggregation* protocol defined in UnLynx [16]. The CNs are organized into a tree structure such that each CN waits to receive the aggregation results from its children and sums them up before passing the result on to its own parent.
6) *Collective Tree Key Switching (CTKS).* The CNs collectively convert the aggregated result, encrypted under K, to the same result encrypted under Q's public key K', without ever decrypting. This protocol (Protocol 1) is a new construction of the *Key Switching* proposed in UnLynx [16]. Conceptually, each CN partially decrypts m (i.e., the term $-(C_1)k_i$ in the computation in step 2) and re-encrypts it with Q's public key K' (i.e., the term $+\alpha_i K'$ in step 2).

Protocol 1 Collective Tree Key Switching (CTKS)

Input. $\mathrm{E}_K(m)=(C_1, C_2)=(rB,mB+rK)$, K'
Output. $\mathrm{E}_{K'}(m)=(C'_1,C'_2)=(r'B,mB+r'K')$

Protocol.

1. The root CN_1 sends C_1 down the tree to all CNs.
2. Each CN_i generates a secret uniformly-random nonce α_i and computes $w_{i,1}=\alpha_i B$ and $w_{i,2}=-(C_1)k_i+\alpha_i K'$
3. The CNs collectively aggregate (i.e., using CTA) all the $w_{i,1}$ and $w_{i,2}$.
4. CN_1 finally computes $(C'_1,C'_2)=(\sum w_{i,1},C_2+\sum w_{i,2})=(r'B,mB+r'K')$ where $r'=\sum\alpha_i$.

We improve the efficiency of $CTKS$ by changing the way the ciphertexts are transformed and by organizing the CNs in a tree structure, thus reducing its execution time. In this structure, multiple CNs can perform their local operations (3 scalar

multiplications and 1 addition) in parallel, and the CTA requires $\#CN-1$ aggregations and communications between the nodes. We show the computational complexity of all Drynx protocols in Table II.

7) *Decryption.* Q decrypts and decodes the query results.

Security Arguments. We show that, as long as one CN is honest, an adversary who controls the remaining CNs, DPs and Q cannot break data confidentiality. Without loss of generality, we assume that at least one DP is honest, as only in this case there is data to protect from the adversary. We sketch the proof by relying on the real/ideal simulation paradigm [73] and show that an adversary cannot distinguish a "real" world experiment, in which the adversary is given "real" data (sent by honest DPs), and an "ideal" world experiment, in which the adversary is given data (e.g., random) generated by a simulator. It can be shown that the DPs send encrypted data that are never decrypted before being aggregated and re-encrypted ($CTKS$) under Q's public key. Therefore, due to the cryptosystem's semantic security, the adversary cannot distinguish between a simulation and a real experiment. It can be seen that data confidentiality is thus ensured during end-to-end query execution:

In *Retrieval & Encoding*, the DPs operate only on their local data and no external data is seen by any malicious party. In *Encryption*, the DPs encrypt their responses with K and these responses are aggregated, still under encryption, in CTA. The (summed) ciphertexts cannot be decrypted unless all CNs collude, which is not possible as they follow an Anytrust model. Finally, in $CTKS$ (Protocol 1), a ciphertext is switched from K to Q's public key such that Q can decrypt:

- in $CTKS$ *Steps: 1-3.* The ciphertext is encrypted under K and thus cannot be decrypted without the collusion of all CNs.
- in $CTKS$ *Step: 4.* The ciphertext is always $(\tilde{C}_1,\tilde{C}_2)=(\tilde{r}B,mB+\tilde{r}K')$ where $\tilde{r}=\sum_{i=0}^{t}\alpha_i$ and $0\leq t\leq \#CN$ and can only be decrypted if the t CNs collude with Q, who is the intended recipient of the message.

2) **DPs' Privacy:** Drynx protects DPs' and individuals' privacy by ensuring that (a) each DP can privately decide whether to answer a query, (b) only the result of the operation, as defined by the operation *encoding*, is disclosed to Q, and (c) no entity can infer information about a single DP or individual.

a) **Neutral Response:** If a DP determines that a query can jeopardize its privacy, it can choose to not respond, or answer with a neutral response, thus hiding its refusal to participate in the query without distorting the query results. For this purpose we define *neutral response*:

Definition 2. *A DP_i sends a* neutral response *by defining its response encoding (Definition 1) by $\rho(\bar{r_i})\equiv(\mathbf{O},0)$, where $\mathbf{O}$ is the neutral vector such that $\mathbf{W}+\mathbf{O}=\mathbf{W}$ with $\mathbf{W}$ being any encoding vector; $c_i=0$ as DP_i computes on 0 records.*

In Section VII, we describe how a *neutral response* can be generated for each listed *encoding*.

Security Arguments. A DP not answering a query would suggest (leak) to other entities that this query is too sensitive for it. DPs' responses are always encrypted and, due to the indistinguishability property of the underlying cryptosystem, a *neutral response* is indistinguishable from a non-neutral one, thus effectively hiding the DP's refusal.

b) **Privacy-Preserving Bit-wise Operations:** In Drynx, DPs' responses are summed through the available additive homomorphism; if these responses are binary, the result of the sum can leak to Q more than the operation result. For example, when an `OR` operation is executed over a set of DPs, Q should only know if the answer is $true$ (1) or $false$ (0). Nevertheless, if the DPs' responses are naively summed, Q gets the number of DPs that answered '1' and '0'. To overcome this issue, we propose the *Collective Tree Obfuscation (CTO)* protocol, detailed in Protocol 2. For bit-wise operations, CTO is run between steps CTA and $CTKS$ of the query execution. In CTO, the CNs collectively obfuscate a ciphertext by multiplying it with a random secret.

CTO enables privacy-preserving bit-wise operations in Drynx as a '1' is obfuscated to a random value whereas '0' is preserved. To know the result of the operation, Q only checks if the final value is '0' or not.

Protocol 2 Collective Tree Obfuscation (CTO)

Input. $\mathrm{E}_K(m)=(C_1, C_2)=(rB,mB+rK)$
Output. $\mathrm{E}_K(sm)=(srB,smB+srK)$
Protocol.

1. Root CN_1 sends (C_1, C_2) down the tree to all CNs.
2. Each CN_i generates a secret uniformly random nonce s_i and computes $(\hat{C}_{i,1},\hat{C}_{i,2})=s_i\cdot(C_1, C_2)$
3. The CNs collectively aggregate (i.e., using CTA) all the $(\hat{C}_{i,1},\hat{C}_{i,2})$.
4. CN_1 obtains $\mathrm{E}_K(sm)=s\cdot(C_1, C_2)$ where $s=\sum s_i$.

Security Arguments. Protocol 2 does not hinder the confidentiality of m and indeed obliviously and statistically obfuscates m. The confidentiality relies on the cryptosystem's semantic security, as m remains encrypted during the whole protocol execution. A multiplicative blinding of m in $\mathbb{Z}_p$ is defined by $s\cdot m$, where s is a secret scalar value in $\mathbb{Z}_p$. The output of the CTO protocol is the encryption of $(\sum s_i)\cdot m$. We can rewrite $(\sum s_i)\cdot m$ by separating the contributions of the honest CNs h (at least one

CN due to our Anytrust model assumption) and malicious CNs e: $(\sum_{i\in h} s_i + \sum_{i\in e} s_i)\cdot m = (\sum_{i\in h} s_i)\cdot m + (\sum_{i\in e} s_i)\cdot m$. Even if an adversary knows $(\sum_{i\in e} s_i)\cdot m$, the other term $(\sum_{i\in h} s_i)\cdot m$ ensures a multiplicative blinding of m in $\mathbb{Z}_p$.

c) **Distributed Differential Privacy:** Drynx relies on the Collective Differential Privacy (CDP) protocol, introduced in Unlynx [16], to ensure differential privacy, and prevent information inference about some DPs and/or individuals from the query results. For completeness, we briefly present the CDP (Protocol 3) and refer to [16] for more details. The choice of parameters depends on the application's privacy policy and is out of the scope of this paper.

Protocol 3 Collective Differential Privacy (CDP)

Input. ϵ (defined in Section III-F), Δf: query sensitivity, and θ: quanta
Output. $\mathrm{E}_K(\hat{n}_1,...,\hat{n}_{\tilde{l}})$

Initialization
1. The distribution $LD = Laplace(0, \Delta f/\epsilon)$ is publicly agreed on.
2. LD is publicly sampled, using the quanta θ, to a list of $\tilde{l}$ noise values $\tilde{n_1},...,\tilde{n_{\tilde{l}}}$.

Protocol.
1. Each CN privately and sequentially shuffles $\tilde{n_1},...,\tilde{n_{\tilde{l}}}$, producing $\mathrm{E}_K(\hat{n_1},...,\hat{n_{\tilde{l}}})$.
2. First elements of $\mathrm{E}_K(\hat{n_1},...,\hat{n_{\tilde{l}}})$ are used as oblivious noise values and added to the query result.

Security Arguments. We observe that the list of noise values is verifiably generated from the differential privacy parameters and that all the CNs privately shuffle the values. This protocol's security is analyzed in details in UnLynx [16].

3) **Query Execution Correctness:** We first describe how Drynx provides auditability by enabling an efficient verification of the query execution correctness. The latter is achieved by guaranteeing results robustness and computation correctness. The first is ensured by limiting the DPs' values to be in a specific range (by means of range proofs) and the second by using ZKPs for all the CNs computations.

a) **Auditability:** To provide an efficient solution for the query verification, Drynx relies on a set of VNs that verify the query correctness in parallel to its execution and without affecting its runtime. After each operation, Q, the CNs and DPs create proofs of correct computations or value range that they sign with their private key (to provide authentication). Their signed proofs are sent to all the VNs. This enables an efficient query execution as the proof creation and verification are executed independently from it.

In order to implement this solution, we can rely on the distributed architecture of the VNs and can provide integrity and immutability by using a blockchain, i.e., the *proof blockchain*. This enables the public and immutable storage of both the query and its verification results. Moreover, it enables an efficient and lightweight verification of the query correctness. An auditor, e.g., Q, has only to request the block corresponding to the query, to verify the VNs signatures and to check the query verification results. We detail this in Protocol 4 and show an example of the *proof blockchain* in Figure 3.

Protocol 4 Query Verification

Query

Q:

1. Q signs and broadcasts the query to the VNs.

VNs:

1. Each VN verifies Q's signature.
2. Each VN deterministically derives the list of expected proofs for the query. It initializes a *query-proofs map* that stores the result of the verification for each proof: `true`, `false`, `not received` (before a predefined timeout).

Query Execution.

DP *or* CN*:*

1. A DP or CN executes an operation, then creates, signs and sends the corresponding proof to the VNs.

VNs:

1. Each VN verifies the prover's signature.
2. Each VN verifies the proof and stores the result in its *query-proofs map*.
3. Each VN stores the proof in its local (key, value)-database. The key is uniquely and deterministically derived from the query, the prover's ID and the proof type.

End of Query Execution (or timeout).

VNs:

1. One of the VNs (e.g., chosen in a round-robin fashion) gathers all VNs' *query-proofs maps*.
2. The same VN creates a block containing the *Query Unique ID*, the *Query* and all the *query-proofs maps*.
3. The block is sent around such that each VN checks that its *query-proofs map* and the query are correctly saved. If this is the case, the VN signs the block.
4. The VNs run a consensus algorithm such that a block signed by a threshold f_h of VNs is consistently added to the blockchain. Each VN keeps a local copy of the blockchain.

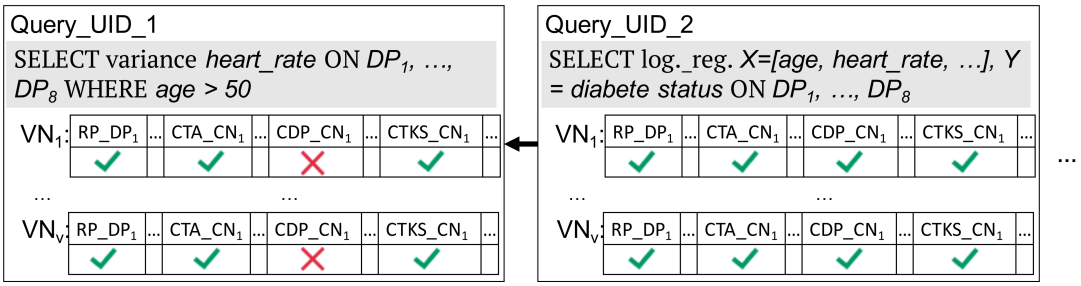


Fig. 3: *Proof blockchain*. Each block contains Query ID and content, and each VN's *query-proofs map*. RP is range proof.

Security Arguments. If an entity trusts a threshold f_h of the VNs, it can verify the query correct execution by checking the corresponding block in the *proof blockchain*. The verifier can check that f_h nodes agree on the correctness of the proofs. A block is created for every query, even if the proofs are wrong, thus enabling any entity to determine which parties were involved in incorrectly computed queries. Otherwise, as all the proofs are universally verifiable and stored by all VNs, an auditor, not trusting f_h of the VNs, can request the proofs from a subset of them and check the proofs by itself.

b) **Results Robustness:** If the querier defines a query with range boundaries on the DPs' values, the DPs are requested to create proofs of range by following the algorithm detailed in Algorithm 1. This algorithm is built by adapting the $[0,u^l)$-range proof scheme proposed by Camenisch et al. [56] to the Anytrust model. In this algorithm, the prover, i.e., DP, writes its secret value m in base-u and commits to the u-ary digits by using the CN_is' signatures on these digits ($A_{i,b}$ in Algorithm 1). The l created commitments complete the proof. To adapt this algorithm to the Anytrust model, the DP must compute multiple proof elements, i.e., c, $V_{i,j}$, $a_{i,j}$, by combining all CNs' signatures, i.e., Z_i, $A_{i,b}$. This ensures that the DP uses at least one CN's signature for which it does not know the underlying secret. The same transformation in [56] can be applied to generalize the proof to any range $[b_l,b_u)$.

Security Arguments. Both the correctness and the zero-knowledge property of the range proof are proven by Camenisch et al. [56].

These proofs are universally verifiable and sound in the Anytrust model. The latter comes from the fact that the elements depending on the CNs' secrets x_i are computed as a combination of all their public signatures. As at least one CN_i is honest-but-curious, one of the x_i is unknown (not revealed) to the DP (prover).

Algorithm 1 Input Range Validation in Anytrust Model

A DP proves that its secret $m \in [0,u^l)$, where u and l are two integers. $C_2 = mB + r\Omega$ corresponds to the right part of $\mathrm{E}_\Omega(m) = (C_1, C_2)$. $e()$ is a pairing function (bilinear map [56]) on an Elliptic Curve and H is a hash function.

Initialization:

Each CN_i picks a random $x_i \in \mathbb{Z}_p$ and computes $Z_i \leftarrow Bx_i$, $A_{i,b} \leftarrow B(x_i+b)^{-1}\ \forall b \in \mathbb{Z}_u$.
All Z_i and $A_{i,b}$ are made public.

Proof Creation:

DP computes value $c = H(B, C_2, \sum_i Z_i)$ and
for each $j \in \mathbb{Z}_l$ such that $m = \sum_j m_j u^j$ **do**
  Pick three uniformly-random values $s_j, t_j, v_j \in \mathbb{Z}_p$
  for each computing node CN_i **do**
    $V_{i,j} = A_{i,m_j} v_j$
    $a_{i,j} \leftarrow -s_j \cdot e(V_{i,j}, B) + t_j \cdot e(B,B)$
  end for
  $z_{v_j} \leftarrow t_j - v_j c\ (mod\ p)$ and $z_{m_j} \leftarrow s_j - m_j c\ (mod\ p)$
end for
DP picks $n \in \mathbb{Z}_p$ and computes $z_r = n - rc\ (mod\ p)$ and $D \leftarrow \sum_j Bu^j s_j + \Omega n$
DP publishes $proof = \{C_2, c, z_r, z_{v_j}, z_{m_j}, D, a_{i,j}, V_{i,j}\}\ \forall j \in \mathbb{Z}_l$ and $\forall i \in \{1,...,\#CN\}$.

Proof Verification:

Any entity can check that:
$D = C_2 c + \Omega z_r + \sum_j Bu^j z_{m_j}$ and $a_{i,j} = e(V_{i,j}, Z_i)c - z_{m_j} \cdot e(V_{i,j}, B) + z_{v_j} \cdot e(B,B)$, $\forall j \in \mathbb{Z}_l$ and $\forall i \in \{1,...,\#CN\}$.

c) **Computation Correctness:** In order to ensure the correctness of the query execution, each computation executed by a CN has to be proven correct.

- *Collective Tree Aggregation*. The CNs provide to-be-aggregated input ciphertexts and the resulting ciphertexts that constitute the ZKP.
- *Collective Tree Obfuscation*. The CNs produce an obfuscation proof by relying on Expression (1) in Section III-C. Each CN_i multiplies C by s_i to obtain the obfuscated ciphertext (C'_1, C'_2) with (a) $C'_1 = s_i C_1$ and (b) $C'_2 = s_i C_2$. For both equations, $y_1 = s_i$ is the discrete logarithm; we have the public values $A = C'_1$, $A_1 = C_1$ for (a) and $A = C'_2$, $A_1 = C_2$ for (b), which constitute the proof.
- *Collective Differential Privacy*. In this protocol, each CN sequentially executes a Neff shuffle and produces the corresponding ZKP of correctness described in Section III-G. This proof basically contains the input and output lists, the public key encrypting the ciphertexts, and commitment values.
- *Collective Tree Key Switching*. The CNs create the ZKP by applying Equation (1) in Section III-C, in which we have $y_1 = k_i$, $y_2 = \alpha_i$, the discrete logarithms of $k_i B = K_i$ and $\alpha_i B$, respectively. All points K_i, $\alpha_i B$, $A = w_{i,2}$, $A_1 = -rB$ and $A_2 = K'$ are made public and do not leak any information about the underlying secrets.

Security Arguments. We rely on proofs that are universally verifiable and zero-knowledge. They do not affect data confidentiality beyond what can be inferred from the proven facts themselves.

B. Drynx Optimized Design

We present Drynx's final query execution pipeline, before describing how the query verification's performance can be optimized.

1) **Full Query Execution Pipeline:** We show Drynx's full pipeline in Figure 4. Query execution and verification are executed concurrently and multiple steps of the query execution can be executed in parallel. The CNs aggregate each DP's response in CTA, as soon as they receive it. The noise generated from the CDP has to be added after all the results have been aggregated. However, if the differential privacy parameters are predefined, this protocol can be executed independently from the other steps or even pre-computed.

2) **Probabilistic Query Verification:** To improve the performance of the query verification, we enable a probabilistic verification of the proofs by the VNs. We show that this strategy still enables a verifier to detect a misbehaving entity with a high probability, yet considerably improves performance (see Section IX). A *proof* for a specific operation (e.g., $CTKS$ for a set of ciphertexts S) can have multiple *sub-proofs* (e.g., $CTKS$ for one ciphertext $C \in S$). One *proof* is considered incorrect if one or more of the *sub-proofs* is incorrect. We introduce the two thresholds T and T_{sub} that define the probability of verifying a single *proof* and a *sub-proof*, respectively. We modify the VNs' operations in step 2 of the *Query Execution* described in Protocol 4, by adding

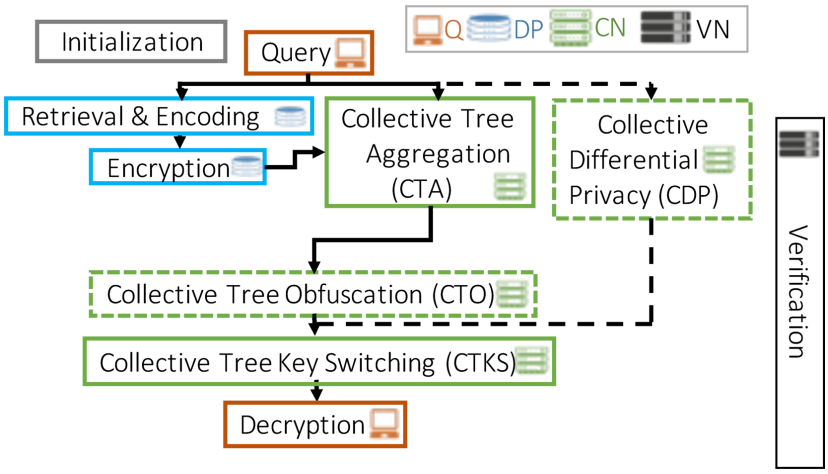


Fig. 4: Drynx's complete optimized query-execution. Arrows represent causal links. Steps without direct links can be executed independently and dashed steps are optional.

this probabilistic verification based on T and T_{sub}. Each VN stores all the *proof* it receives. It then generates a random value $r \in [0,1]$; if $r < T$, it starts the probabilistic verification of the *sub-proofs*. For each *sub-proof*, the same method is applied, using T_{sub}.

Security Arguments. The probabilistic verification does not necessarily compromise the security level of the system, given that the verification of each proof is redundantly done by each VN. A *proof* is verified with a probability $p_{ver} = 1-(1-T)^{N_{VN}}$, where N_{VN} is the number of VNs, and a *sub-proof* with a probability $p_{ver_{sub}} = 1-((1-T)+T(1-T_{sub}))^{N_{VN}}$. The probability that a *proof* or a *sub-proof* is verified by at least f_h nodes is

$$P_{f_h} = \sum_{i=f_h}^{N_{VN}} \binom{N_{VN}}{i} p^i (1-p)^{N_{VN}-i},$$

where p is either p_{ver} (for a *proof*) or $p_{ver_{sub}}$ (for a *sub-proof*). For example, if $N_{VN} = 7$, $T = 1$ and $T_{sub} = 0.3$, all the *proofs* are at least partially verified and each *sub-proof* is verified by $f_h = 5$ VNs with $P_{f_h} = 98.48\%$. Each *sub-proof* is thus verified by at least f_h of the VNs with a high probability. Due to the honesty assumption, a *sub-proof* is at least verified by one honest VN with a high probability. Moreover, the thresholds T and T_{sub} can be set to arbitrarily reduce the probability that one *sub-proof* is not verified by at least one honest node. Therefore, if all the VNs that participated in the verification agree on the result, the auditor knows the *proof* is correct, otherwise it can either choose to only trust some of the VNs or fetch all proofs and verify them itself, as all the proofs are universally verifiable. For example, an auditor can choose to verify only the proofs that were not checked by any of the VNs she trusts.

VI. Security Analysis

We employed only existing, peer-reviewed cryptographic schemes and discussed the composability of the security of the different blocks in previous sections. We corroborate these arguments with a brief summary of the security analysis.

- *Data confidentiality*. In Section V-A1, we sketched the proof for confidentiality in our simplified system and discussed in Section V-A how further design choices do not hinder confidentiality. In summary, data confidentiality is ensured as the data are always encrypted and no operation, e.g., ZKP creation, affects it.
- *DPs' privacy*. DPs can privately decide whether to answer a query, and differential privacy is ensured for the DPs and individuals, which protects them from potential inferences stemming from the release of end results. The latter is ensured in Drynx by blindly adding noise, sampled from a specific distribution, to the query end-results. As described in Section V-A2, this noise can be verified to be from a specific distribution (e.g., Laplacian) and no entity knows which noise value is added.
- *Results robustness*. This is ensured as all DPs' values can be verified to be within a certain range and all CNs' computations must be proven correct, as depicted in Section V-A3. By enforcing the generation of range proofs by DPs, we protect against strong outliers, maliciously or erroneously input, which can significantly distort the query results. DPs can still input incorrect values, but their influence on the final result is limited. We give an intuition on how robust a computation is against such behavior in Section IX-B.
- *Computation correctness.* The proofs of correct computations (Section V-A3) ensure that the DPs' answers are correctly aggregated (CTA) and that the remaining steps (CTO, $CTKS$, CDP) are correctly executed.

VII. ENCODINGS

We present a set of statistical computations that can be executed in Drynx. We then explain how to instantiate *encodings* (Definition 1) for the training of both linear and logistic regression machine-learning models. We adapt the logistic regression solution, proposed by Aono et al. [33], to our framework, thus enabling Q to train this model in a verifiable and privacy-preserving way, even in the presence of a strong adversary. Some of the encodings are adapted from the Corrigan-Gibbs and Boneh [26] system and improved upon.

Numerical Statistics. Table I lists a set of simple statistics that can be performed with Drynx. The `sum`, `mean`, `variance`, `std. deviation`, `cosine similarity (cosim)` and `R`^{2} operations are executed by requiring the DPs to send the result of their local and partial statistic computation. As an example, for `variance`, each DP_i locally computes the sum of the values (records) h_j that match the query, ($\sum_{j=1}^{c_i} h_j$) where c_i is DP_i's dataset cardinality, the square of those same values ($\sum_{j=1}^{c_i} h_j^2$) and generates $\rho(\bar{r_i}) = ([\sum_{j=1}^{c_i} h_j, \sum_{j=1}^{c_i} h_j^2], c_i)$. These values are independently aggregated among all DPs and the overall variance is computed by Q, after decryption, using the corresponding π (defined in Table I). For the `frequency count`, DPs are expected to send the vector $\mathbf{V_i}$ filled with the number of occurrences (fc) for specific values. The `cosine similarity` is computed between two vectors ϕ and $\bar{\phi}$, where each DP_i holds a subset of the coefficients of each vector.

Operat. (f)	π (on N DPs)	ρ ($\mathbf{V_i}$=$[v_{i,1},...,v_{i,d}]$,c_i)
sum	$\sum_{i=1}^{N} v_{i,1}$	($[\sum_{j=1}^{c_i} h_j]$, c_i)
mean	$\frac{\sum_{i=1}^{N} v_{i,1}}{\sum_{i=1}^{N} c_i}$	($[\sum_{j=1}^{c_i} h_j]$, c_i)
variance std. dev.	$\sigma^2 = \frac{\sum_{i=1}^{N} v_{i,2}}{\sum_{i=1}^{N} c_i} - (\frac{\sum_{i=1}^{N} v_{i,1}}{\sum_{i=1}^{N} c_i})^2$ $\sigma = \sqrt{\sigma^2}$	($[\sum_{j=1}^{c_i} h_j$, $\sum_{j=1}^{c_i} h_j^2]$, c_i)
AND/OR	$\sum_{i=1}^{N} v_{i,1} \stackrel{?}{=} 0$	($[R_j]$,c_i) or ($[b_j]$, c_i)
min/max	$l/rm_{\neq 0}(\sum_{i=1}^{N} v_{i,1},..., \sum_{i=1}^{N} v_{i,d})$	($[R_{j,1}$, ..., $R_{j,d}]$, c_i) or ($[b_{j,1}$, ..., $b_{j,d}]$, c_i)
frequ. count	$\sum_{i=1}^{N} v_{i,1},...,\sum_{i=1}^{N} v_{i,d}$	($[fc_{j,1}$, ..., $fc_{j,d}]$, c_i)
set int/un	$\sum_{i=1}^{N} v_{i,1},...,\sum_{i=1}^{N} v_{i,d}$	($[R_{j,1}$, ..., $R_{j,d}]$, c_i) or ($[b_{j,1}$, ..., $b_{j,d}]$, c_i)
cosim	$s(\phi,\bar{\phi}) = \frac{\sum_{i=1}^{N} v_{i,1}}{\sqrt{\sum_{i=1}^{N} v_{i,2}}\sqrt{\sum_{i=1}^{N} v_{i,3}}}$	($[\sum_{j=1}^{c_i} \phi_j \bar{\phi}_j, \sum_{j=1}^{c_i} \phi_j^2$, $\sum_{j=1}^{c_i} \bar{\phi}_j^2]$, c_i)
R^2	$1 - \frac{\sum_{i=1}^{N} v_{i,3}}{\sigma^2}$	($[\sum_{j=1}^{c_i} y_j$, $\sum_{j=1}^{c_i} y_j^2$ $\sum_{j=1}^{c_i} (y_j - \hat{y_j})^2]$, c_i)

TABLE I: Example set of *encoding* instantiations. All DPs *encoding*s (ρ) are then aggregated such that Q computes π at the end.

Bit-Wise Statistics. As depicted in Table I, bit-wise operations can be executed in two ways: Each DP_i either (1) sends a random encrypted integer R or (2) sends an encrypted bit b. For (1), in the `OR` (resp. `AND`) case, each DP_i is requested to send an encrypted integer $E_K(R_i)$, where $R_i = 0$ if the input is 0 (resp. 1), and a random positive integer otherwise. The `OR` (resp. `AND`) expression is $true$ (resp. $false$) if the sum $\sum R_i > 0$. Q obtains the final result by testing if the output is 0 or not. The result of this operation can be erroneous if $\sum R_i \equiv 0 \ mod(\#G)$, or in other words, if the order $\#G$ of the Elliptic Curve subgroup divides the sum of all DPs' random values. This happens only with a probability smaller than $1/(\#G - 1)$ (proof in Appendix B). This probability is close to 0 as $\#G$ is much bigger than the decryptable plaintext values, and can be further reduced by repeating the query. Alternatively, in (2) each DP_i has to send $b_{i,j} = 0$ or $b_{i,j} = 1$ encrypted value. This eliminates the error probability but requires more computations and proofs of correctness, as the DPs have to prove that their values are in $\{0,1\}$, and a CTO protocol (Section V-A2b) has to be executed to preserve privacy. The `min` (resp. `max`) is computed by applying the `or` operation element-wise among vectors $\mathbf{V_i}$. Each DP_i computes its local min (res. `max`) m_{DP_i} in a specified range, e.g., [0:100], which is represented by $\mathbf{V_i} = [b_{i,0},...,b_{i,100}]$. Each $b_{i,j} > m_{DP_i}$ (resp. $b_{i,j} < m_{DP_i}$) is encoded with a '1' (or random) and a '0' otherwise. The `min` (resp. `max`) across all DPs corresponds to the leftmost (resp. rightmost) position with a '1' in the vector resulting from the `OR` operation. Similarly, the `set intersection (resp. union)` is computed by using the `AND` (resp. `OR`) operation element-wise on the vectors $\mathbf{V_i}$.

Regression Models.

Linear Regressions. We assume a dataset distributed over the DPs with D features $x_1,...,x_D$ and a label value y such that $y \approx c_0 + c_1 \times x_1 + c_2 \times x_2 + ... + c_D \times x_D$. Drynx computes the least-squares linear fit over all the DPs by building a system

of $D+1$ equations that Q can use in order to compute the linear regression coefficients $c_0,c_1,c_2,...,c_D$:

$$\begin{pmatrix} n & \sum x_{\mu,1} & ... & \sum x_{\mu,D} \\ \sum x_{\mu,1} & \sum x_{\mu,1}^2 & ... & \sum x_{\mu,1}x_{\mu,D} \\ ... & ... & ... & ... \\ \sum x_{\mu,D} & \sum x_{\mu,1}x_{\mu,D} & ... & \sum x_{\mu,D}^2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ ... \\ c_D \end{pmatrix} \approx \begin{pmatrix} \sum y_\mu \\ \sum y_\mu x_{\mu,1} \\ ... \\ \sum y_\mu x_{\mu,D} \end{pmatrix} \tag{2}$$

where all the sums are between $\mu=1$ and $\mu=\sum_{i=1}^{N} c_i$. Each DP_i sends $\sum_{j=1}^{c_i} x_{j,\eta}$, $\sum_{j=1}^{c_i} x_{j,\eta}x_{j,\zeta}$, $\sum_{j=1}^{c_i} y_j$, $\sum_{j=1}^{c_i} y_j x_{j,\eta}$, $\forall \eta,\zeta \in \{1,2,...,D\}$, $\eta \neq \zeta$.

Logistic Regressions. We consider again a dataset of N_{da} records (distributed among the DPs) with a dimension D where each record $x^{(\mu)}=(1,x_1^{(\mu)},\cdots,x_D^{(\mu)}) \in R^D$ consists of D features and an offset term of 1, and is associated with a label $y^{(\mu)} \in \{0,1\}$. The original logistic regression cost function is

$$J(\theta)=\frac{1}{N_{da}}\sum_{\mu=1}^{N_{da}}\left[-y^{(\mu)}\log(h_\theta(x^{(\mu)}))-(1-y^{(\mu)})\log(1-h_\theta(x^{(\mu)}))\right]+lr_\theta,$$

where $h_\theta(x)=1/(1+\exp(\sum_{\eta=0}^{D}\theta_\eta x_\eta))$ and $lr_\theta=\frac{\lambda}{2N_{da}}\sum_{\eta=1}^{D}\theta_\eta^2$, λ is the L2-regularization parameter. $J(\theta)$ can be approximated by a linear function

$$J_a(\theta)=\left[\frac{1}{N_{da}}\sum_{\tau=1}^{k}\sum_{r_1,\ldots,r_\tau=0}^{D} a_\tau(\theta_{r_\tau}\cdots\theta_{r_\tau})A_{\tau,r_1,\ldots,r_\tau}-a_0\right]+LR_\theta,$$

by using the fact that $\log(\frac{1}{1+\exp(x)}) \approx \sum_{\tau=0}^{k} a_\tau x^\tau$, where $a_0,a_1,...,a_k$ can be chosen as the $k+1$ first coefficients of the Taylor expansion of $\log(\frac{1}{1+\exp(x)})$, or as the coefficients of the quadratic approximation that minimizes the area between the original function and its approximation. The $A_{\tau,r_1,\cdots,r_\tau}$ coefficients are defined by

$$A_{\tau,r_1,\cdots,r_\tau}=\sum_{\mu=1}^{N_{da}} a_{\tau,r_1,\cdots,r_\tau}^{(\mu)}=\sum_{\mu=1}^{N_{da}}(y^{(\mu)}-y^{(\mu)}(-1)^\tau-1)(x_{r_1}^{(\mu)}\cdots x_{r_\tau}^{(\mu)}),$$

where the $a_{\tau,r_1,\cdots,r_\tau}^{(\mu)}$ are computed and encrypted by the DPs before being collectively aggregated by the CNs.

Neutral Response. A neutral response for `and` and `set intersection` is $O=[1,...,1]$, and $O=[0,...,0]$ for other operations.

Optimized and Iterative Encoding Drynx can also be used in order to execute iterative processes, e.g., a k-means algorithm. In this case, each iteration can simply be mapped to a query sent to the system. An iterative process can also be used in order to optimize existing *encoding*s, such as the `min` and `max`. In their basic versions, these *encoding*s rely on a d-bit vector in which each bit represents a value in a predefined range of size $d=|b_u-b_l|$. This means that each DP sends d ciphertexts. This process can be optimized by using a binary-search iterative process as depicted in Protocol 5. In the *Range Reduction* step, each query only requires one ciphertext per DP and reduces by half the range of possible answers. This step is repeated until this range is reduced to a predefined size EL. It must be noted that the execution of other iterative processes would work in a similar way: For example, for a k-means algorithm [74], Q performs one iteration by executing one query that includes the centroids in clear; the DPs then assign their points to the closest centroid before aggregating their points by cluster; then, the same operation is repeated among all DPs by using Drynx typical query workflow and Q computes the new centroids. As in Protocol 5 and as described below, this algorithm leaks the intermediate results. We do not address the problem of hiding the intermediate results, e.g, by using differential privacy, in this work.

Protocol 5 Iterative Process (`max` example)

Input. Query = `max` in $ra=[b_l,b_u]$ and EL
Output. Max value
Range Reduction:
1: **while** $\#ra > EL$ **do**
2: Q sends `SELECT OR` $(\exists v \in [\lfloor \frac{(b_l+b_u)}{2} \rfloor, b_u])$ `ON` $DP_1,..,DP_n$
3: **if** query returns $true$ **then**
4: $ra=[\lfloor \frac{(b_l+b_u)}{2} \rfloor, b_u]$
5: **else**
6: $ra=[b_l, \lceil \frac{(b_l+b_u)}{2} \rceil]$
7: **end if**
8: **end while**

Final Step:
1: Q sends `SELECT MAX` $[b_l,b_u]$ `ON` $DP_1,...,DP_n$

Security Arguments. For all *encoding* and in each query, Q learns the elements of $\mathbf{V}$ (aggregated over all DPs) and the (approximate) number of samples considered c, as defined by *encoding*.

For the iterative process, in the *Range Reduction*, the DPs' answers remain confidential, but the range is sent in clear in each query thus revealed to other entities. Q controls the size of the range of possible values that is leaked by defining an entropy limit EL. In the *final step*, the `max` query is privately executed on the remaining range. This provides a tradeoff between performance and privacy (that we analyze in Section IX). The number of ciphertexts is lowered to $n = g + \lceil \frac{d}{2^g} \rceil$, $g = \lfloor log_2(\frac{d}{EL}) \rfloor$, which reduces the amount of computations and proofs by a factor $\frac{d}{n}$. For example, if Q wants to know the DPs' minimum value in $[0,1000)$ with $EL = 100$, the workload is reduced by a factor of 7.8 and the query leaks a range of 100 possible minimum values.

VIII. Discussion and Extensions

We illustrate multiple extensions for Drynx by relying on our use cases, HDS and PDS (Section III-A).

Modularity. Drynx is highly modular and some of its security features can be enabled or disabled, depending on the application. For example, if results robustness is not required, input-range validation can be omitted without hindering Drynx's execution and the remaining security guarantees are preserved. The same applies for DPs' privacy features, e.g., differential privacy.

For example, in HDS, each hospital (or DP) locally executes the query on multiple patient records and the range proofs can be omitted if the range of possible values is too broad or if the hospital is trusted to input correct values. Otherwise, the range boundaries have to be set accordingly. In this case, the querier has to use her knowledge on the attributes involved (e.g., age is between 0 and 150) and the information she has on the DPs' data (e.g., DPs have a maximum of X data samples) to define the ranges. In PDS, the ranges for the input values can be used to enforce tighter bounds (e.g., heart rate can only take values in [40,100] beats-per-minute) as each DP has one data record.

Drynx also enables the collective protection of data at rest by having DPs locally encrypt their data with the CNs' collective key K. This limits the flexibility of the system as DPs are then required to pre-compute all necessary inputs (e.g., the square root of the values to enable the computation of the `variance`) and the range proofs before entering the encrypted data in their databases. It also requires a fixed set of CNs, as only they can operate with that pre-encrypted data.

As mentioned before, Drynx's primary goal is to guarantee DPs' privacy and still enable the queriers to obtain the results of computations performed over multiple databases. For this, Drynx enables optional security and privacy features, such as differential privacy. These features can be enabled or disabled depending on the application requirements, hence enabling multiple trade-offs between security and privacy, performance and accuracy (see below).

Collusion Resistance. Each participant can play multiple roles without hindering Drynx's security. For example, in HDS, a hospital can be a DP and also play the role of a CN, to ensure its data confidentiality without having to trust any other hospital. It can also be a VN thus take part in the verification process.

Availability. Drynx's privacy and security guarantees hold even in the case where multiple CNs or DPs become unavailable. Any entity can leave or join the system without hindering Drynx's operation, as long as they are not involved in a query under execution. In the event of a CN becoming unresponsive during the query execution, the CTA and $CTKS$ steps cannot be finalized, as they both require the participation of all CNs. Therefore, in this case, the process is stopped and Q can request the same query by choosing another set of CNs, e.g., by excluding the faulty CN(s). An unresponsive DP only reduces the number of responses included in the statistic being computed and does not disrupt Drynx's process. Standard mechanisms, e.g., limiting the rate at which queries are accepted, can be implemented in Drynx to avoid DDoS attacks.

Accuracy. There are several aspects that can influence output precision in Drynx. (a) We first remark that the DPs' inputs to the system have to be approximated by fixed-point representation if they are floating values, as explained in Section III-B. (b) Drynx's encodings and query executions do not intrinsically hinder the accuracy of the computed results, as all operations are exact, as long as the target function is exactly *encodable*. In fact, it is worth noting that the encoding for the logistic regression training is built from an approximation of the original cost function.

Additionally, (c) the DPs can privately decide whether to answer a query; this choice can influence the final result. However, the number of samples considered in the computation, i.e., c_i in Definition 1, is always sent to Q, who can then observe if this number changed since her last query. It also enables her to take an informed decision on the statistical significance of the results, to accept them or not.

(d) Drynx can guarantee differential privacy by adding noise to the final result. In this case, Drynx returns approximate results, and the accuracy loss depends on the chosen privacy parameters and the executed operation. The choice of these parameters and the perturbation introduced in the results is thus orthogonal to this work.

Finally, (e) malicious DPs can try to distort the query result by inputting erroneous values. Drynx limits malicious DPs' influence on the final result by enabling the querier to restrict the range of possible inputs. This bounds the perturbation that some DPs can generate on the results. If the inputs were not bounded, one malicious DP could completely distort the final result by inputting extreme values. It is difficult to provide hard numbers for the accuracy of Drynx in the presence of malicious DPs, as it depends on many parameters such as the executed operation, the chosen input ranges, the number of DPs and data records. Nonetheless, in Section IX we show how the use of ranges limits the influence of malicious DPs in two examples.

Authentication/Authorization. Authentication and authorization fall out of the scope of this paper, but for the sake of completeness we briefly mention here that Drynx can integrate off-the-shelf solutions based on federated or distributed architectures [75], [76], [77].

IX. Performance Evaluation

We discuss our experimental setup and evaluate Drynx's performance. We show that it scales almost (in some cases better than) linearly with the number of CNs, VNs and DPs, and we compare Drynx against existing solutions. We also discuss multiple security, privacy and performance tradeoffs.

A. System Implementation

We implemented Drynx in Go [78], and our full code is publicly available [79]. We relied on Go's native crypto-library and on public advanced crypto-libraries [80]. For the implementation of the proofs' storage and verification, we use a skipchain [81], which is made of blockchain-like blocks that, to enable clients to efficiently navigate arbitrarily on the chain, also contain back-and-forward pointers to older and future blocks. We rely on a (private) permissioned blockchain [82], as in our examples HDS and PDS (Section III-A), the participants, i.e., researchers, patients or hospitals, have to be known and authorized. However, Drynx works independently of the blockchain type, and a permission-less blockchain can also be used in a less restrictive scenario. Drynx works independently of the used elliptic curve; we tested it on the Ed25519 [83] and bn256 elliptic curves [84]. Both curves provide 128-bit security, and we used bn256 by default as it enables pairing operations (required for range proofs). Our prototype is built as a modular library of protocols that can be combined in multiple ways. The communication between different participants relies on TCP with authenticated channels (through TLS).

B. System Evaluation

We used Mininet [85] to simulate a realistic virtual network between the nodes; we restricted the bandwidth of all connections between nodes to 100Mbps and imposed a latency of 20ms on all communication links. We evenly distributed the CNs, DPs, VNs and Q on a set of 13 machines that have two Intel Xeon E5-2680 v3 CPUs with a 2.5GHz frequency that supports 24 threads on 12 cores and 256GB RAM.

We begin our evaluation by studying how the different steps in Drynx's pipeline can be executed in parallel. We then show that Drynx's runtime only slightly increases when the number of records per DP grows (and the number of DPs remains constant).

In our **default setup**, we consider 6 ***CNs*** and 7 ***VNs*.** We set the proof verification thresholds $T = 1.0$ and $T_{sub} = 0.3$ and show, in Section IX-B1, the effect of these thresholds on Drynx's execution time. The joint use of these thresholds ensures that all the proofs are at least partially verified and that each *sub-proof* is verified by $f_h\ VNs$ with a probability of 98.5%. We show Drynx'runtime without the CDP protocol as CDP can be pre-computed or run in parallel with other steps. We notice that the CDP's runtime depends on the number of CNs and on the size $\tilde{l}$ of the list of noise values. This creates a tradeoff between privacy and performance as a greater $\tilde{l}$ provides a higher privacy level, as it reduces $\delta = 1/\tilde{l}$ but also increases the time to generate and shuffle the list of noise values. With a Laplacian distribution and $\tilde{l} = 100$, CDP's runtime is 2.9 seconds with an overhead of 8.1 seconds for the proof verification.

1) **Drynx Evaluation: Parallel Execution.** Figure 5 shows the runtime for training a *logistic regression* model. We use a randomly-generated dataset of 12 floating-point features and 600,000 records split among 12 DPs. We remark that the operations are verified in parallel to the query execution; this parallelization enables Q to obtain the query results as soon as it is computed (denoted by query execution dashed line). At the end of the verification process, an auditor can check the query by verifying the signature and the *query-proofs map* of the corresponding block in the *proofs blockchain*, which in this case takes 0.4 seconds. The blocks' sizes are small as they only contain the query and the corresponding *query-proofs map*; in this example one block is 56kB.

Scaling. We show how Drynx's execution time evolves with an increasing number of data records (Figure 6a), CNs and DPs (Figure 6b) and VNs (Figure 6c). Inspired by HDS and PDS, we simulate the computation of the heart-rate *variance* (values between [0,256)) over a set of distributed patients. In Figure 6a, we observe that Drynx scales better with (a) the number of records per DP (and fixed number of DPs) than (b) with the number of DPs; case (a) represents HDS, where a DP is an hospital with a database of multiple patients, whereas case (b) represents PDS, where each patient is a DP ($\#DPs = \#records$). This is because (a) enables the DPs to locally pre-aggregate their data, thus reducing the amount of proofs and computations. For Figures 6b and 6c and for the remaining part of the evaluation, we set the number of DPs to 10 per CN. In HDS, this could correspond to a use case in which some DPs are hospitals and the others are independent doctors sharing their data. We observe that Drynx's runtime increases with the number of DPs, CNs, and VNs. However, an increasing number of CNs and VNs also means a higher security level, as the trust is distributed among more entities.

Operations. Figure 7 shows Drynx's runtime for all the operations with a large integer range of $[0,2^{20}]$ for each of the DPs' inputs (the size of the DPs' inputs is shown below each operation). We observe that for all operations, the query execution

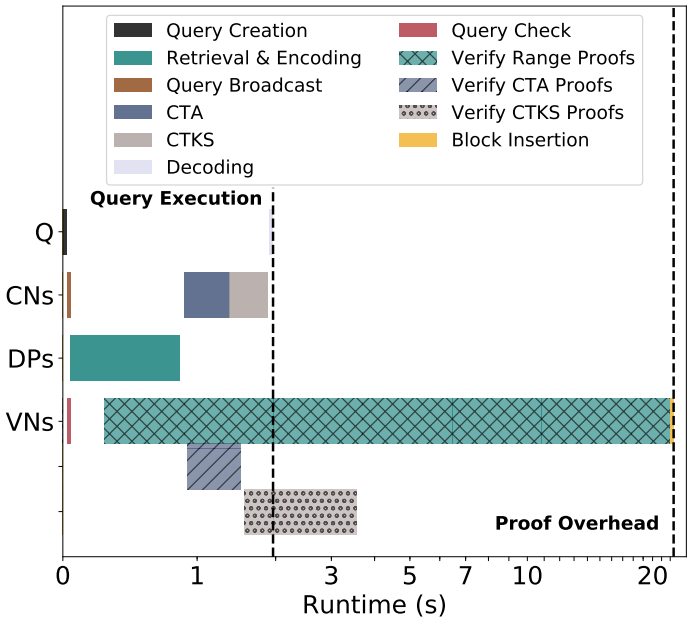
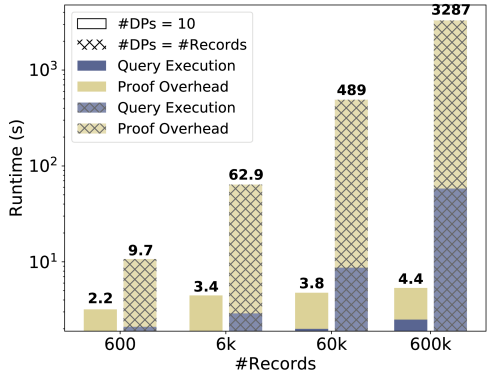
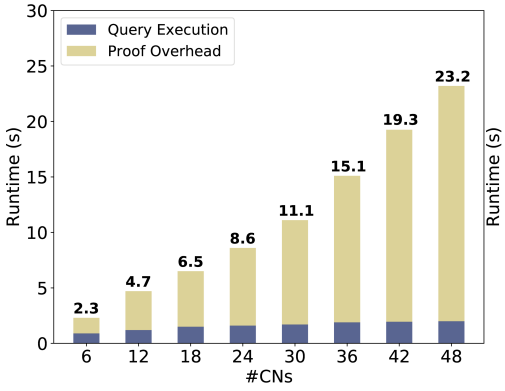


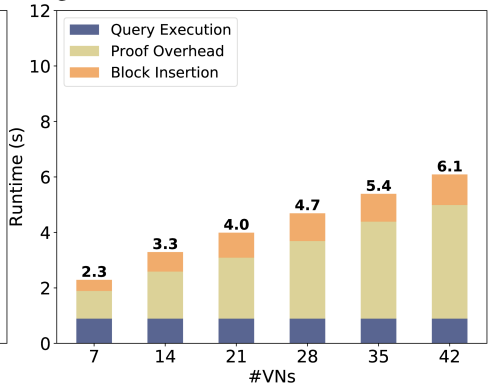
Fig. 5: `Log. reg. training`: 8 features, 6000 records, max. iter.: 25.



(a) `Variance`: increasing nbr. of records.



(b) `Variance`: increasing nbr. of CNs and DPs with 10 DPs per CN.



(c) `Variance`: increasing nbr. of VNs.

Fig. 6: Drynx's scaling.

time is always below 1.5 seconds; and the overhead incurred by the proofs verification increases with the size of the DPs' inputs. This is expected, as the larger the DPs' inputs become, the more ciphertexts there are for the system to process, and more proofs there are to verify. We also observe that bit-wise operations take more time when the DPs opt to send a bit value that is then obfuscated (using the CTO protocol).

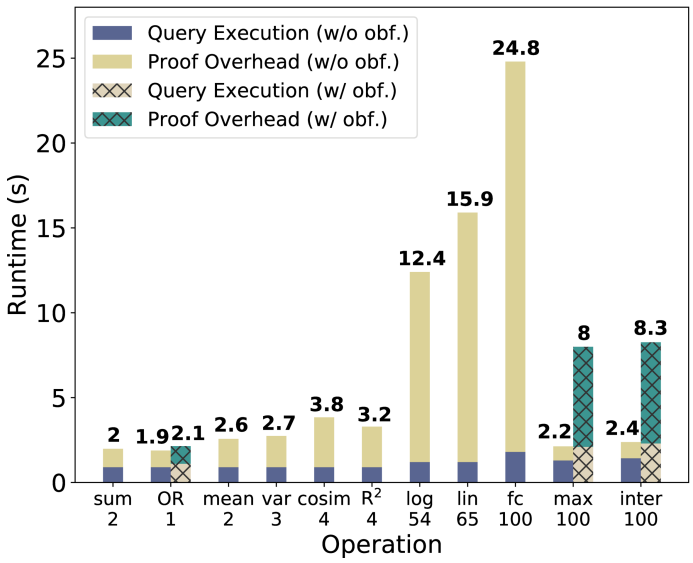


Fig. 7: Runtime for different operations with DPs' inputs sizes. Range $[0,2^{20}]$.

Verification Thresholds. In Figure 8, we show how the different thresholds on the proofs verification affect Drynx's performance with a *variance* query. It can be seen that sending the proofs (communication time is denoted by a dashed line) is the most time consuming part, and that reducing the thresholds reduces the verification time. For example, by having $T=1$ and $T_{sub}=0.2$, we effectively reduce the verification workload by a factor close to 0.8, and a *sub-proof* is still verified by $f_h=5$ of the VNs with a high probability (83.48%).

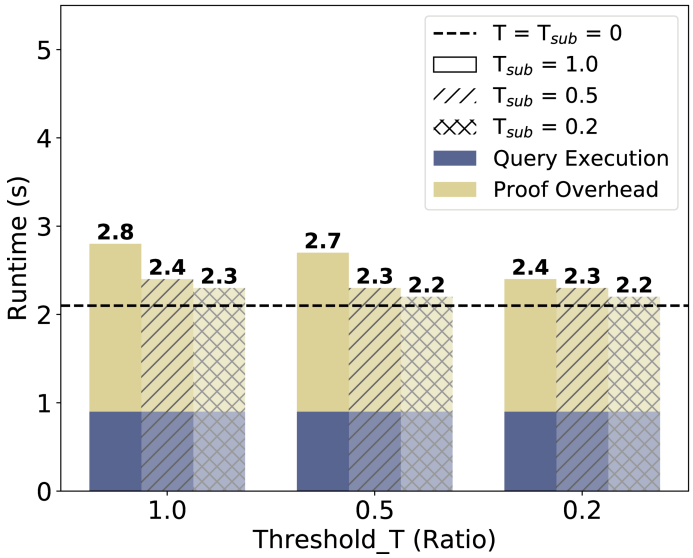


Fig. 8: `Variance`: proofs verif. thresholds.

Malicious DPs. By enforcing DPs' values to be within a specific range, Drynx limits the influence of malicious DPs on the computed statistic. We illustrate this in a simple and realistic example (using PDH from Section III-A) by computing the `average` heart rate over a dataset of 8922 hypertensive patients [86]. The real heart-rate values are limited to be between 40 bpm (beats per minute) and 100 bpm and, as presented by Lorgis et al. [86], the average value obtained among honest DPs is a_h =70 bpm with a 95% confidence interval of ±6 bpm. Each patient (DP_i) must send $(\mathbf{V_i},c_i)=([heart\_rate],count)$ (Definition 1), in which $heart\_rate$ has to be in $[40,100]$ and $count$ in $[0,1]$. In order to maximize the result's distortion, a malicious DP can send an extreme value, which is within the range bounds. We assume that all malicious DPs collude and send the same value $heart\_rate=e$, and that the computed average is given by $a_m=(h\cdot a_h+e\cdot d)/(h+c)$, where h and d are the numbers of honest and dishonest DPs, and c is the sum of c_i sent by malicious DPs. The relative error is $|1-(a_m/a_h)|$.

We remark that a malicious DP can maximize this error with a valid input by sending $([100],0)$. In Figure 9, we observe that with 1% of malicious DPs for the range [40,100], the highest relative error is 1.44%. This error corresponds to 1 bpm, still in the 95% confidence interval. We observe similar results when the cosine similarity is computed in the same settings. For this example, we also present the worst-case scenario in which the cosine similarity computed on the honest DPs is 1 and the malicious DPs input extreme values from the range of accepted values to reduce the similarity. As shown in Figure 9, these numbers highly depend on the chosen bounds. Even if many other factors influence this error (e.g., the computed operation and the distribution of the values), it shows that Drynx can limit the power of malicious DPs.

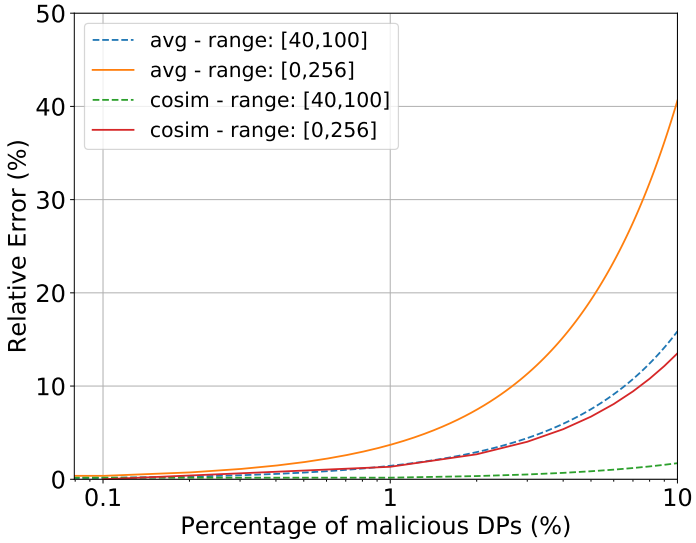


Fig. 9: `Average` and `cosim`: influence of malicious DPs.

Iterative Queries. Figure 10 depicts how Drynx's runtime can be reduced by using multiple queries to execute a `min/max` operation in a binary-search style. This represents a tradeoff between privacy and performance, as each iterative query is sent in clear, leaking the interval where the min/max value is. We assume that Q sets the entropy limit $EL = 100$, in other words, another entity in the system can learn that the `min/max` is in an interval of at least 100 values. The precise value is kept private. We observe that the execution time is not improved when the range is small, but is greatly reduced when the range grows, reaching an execution time reduction of almost 96% at a range size of 100,000.

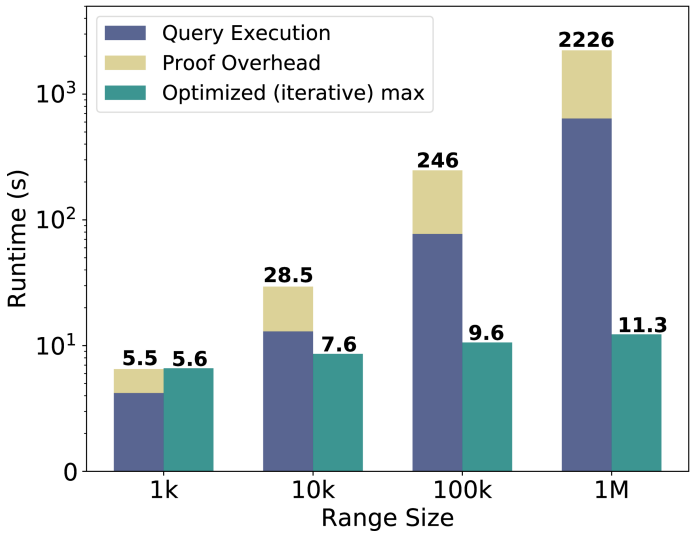


Fig. 10: `max` (iterative): increasing range size.

Communication. Figure 11 depicts Drynx's runtime evolution with respect to both the communication delay and bandwidth capacity with a heart rate *variance* query. We remark that when the latter is reduced by a factor 100, the runtime increases by a factor 2 or 3. This shows that our system is more sensitive to communication delay than bandwidth capacity.

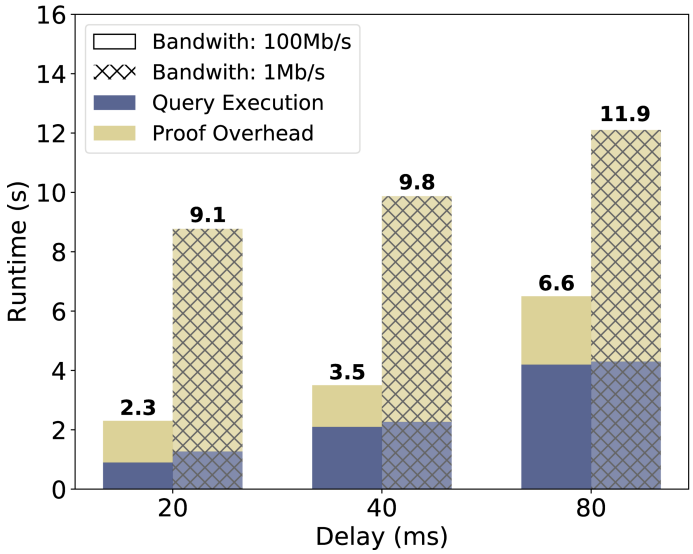


Fig. 11: `Variance`: runtime w.r.t. network bandwidth and delay.

Bandwidth. In Table II, we present the computation and bandwidth complexities for 1 ciphertext (i.e., 2 points (2p) on the elliptic curve, 2p = 64 bytes) per DP. We use DP, VN, and CN as the numbers of corresponding entities in the system. s is the size of the Schnorr signature [57] ($s=96$ bytes), h is the hash size ($h=32$ bytes), l comes from the range $[0,u^l)$ for the range proofs ($u^l=16^2, l=2$), pap is a pairing point's size ($pap=384$ bytes) and n is the number of values that are used in the CDP ($n=100$). We do not include the computational complexity for the local computations executed by the DPs and CNs. We refer to Neff's work [69] for the complexity of the verifiable shuffle (VS). We observe that when the number of CNs and VNs increases, the computational, bandwidth and storage costs increase for all the steps. As having more CNs or VNs improves the security and the distribution of the workload in the system, it creates a tradeoff between security, efficiency, and scalability.

	DPs answer	*DPs* answer with RV	*CTA*	CTO	*CTKS*	*CDP*
QE C.C.	-	-	(CN-1) · *A*	(CN-1)·*A* + (2·CN)·*SM*	(CN-1)·*A* (3·CN)·*SM*	CN · VS
QE band.	DP · 4p (1.28kB)	DP · 4p (1.28kB)	CN · 4p (0.768kB)	2 · CN · 4p (1.536kB)	CN · 6p (1.344kB)	CN · 4p · n (76kB)
QV band. and storage	DP ·VN ·(s+4p) (15.68kB)	DP · VN · (s + (7 + 2l + 2l · CN)p + l · CN · pap + h) (2.43MB)	CN · V N · (s + 4p) (9.048kB)	CN ·VN ·(s+8p+h) (16.128kB)	CN ·VN ·(s+10p+h) (18.816kB)	(CN +1)·VN ·(s+ 13np + h) (2.04MB)

TABLE II: Bandwidth and Storage costs. DP, VN, CN = nbr. of entities; RV = Range Valid.; QE=query exec.; C.C.=comput. complexity; QV=query verif.; *A*=ciphertext addition; *SM*= scalar multi.; *VS*= verif. shuffle

2) **Comparison with Existing Works:** We supplement the related work's overview, described in Section II, by presenting here a qualitative and quantitative comparison with multiple systems that are Drynx's closest related works. We compare Drynx against SMCQL [10], UnLynx [16], Prio [26], Boura et al. [31], Aono et al. [33], Kim et al. [13] and Gazelle [32]. In Table 12a, we show that Drynx provides several functionalities in a strong threat model and achieves results that can rival with other secure and dedicated approaches, notably in the training of logistic regression models as depicted in Figure 12a. Drynx performs as well or better than its two closest related works, UnLynx and Prio, and provides better security guarantees.

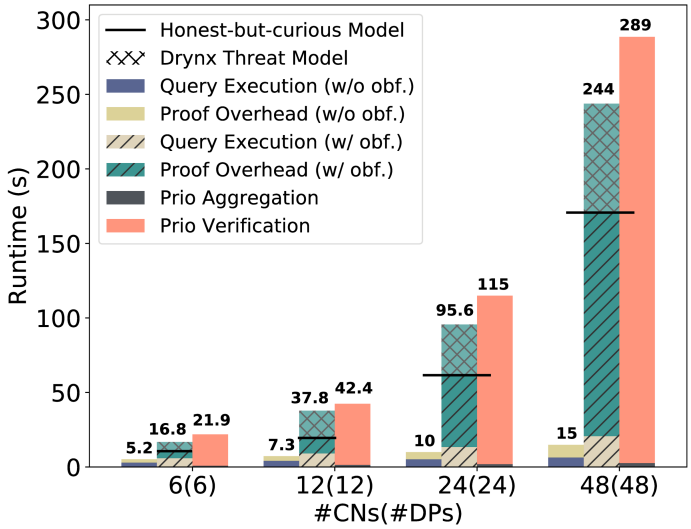
We observe that solutions based exclusively on secret sharing and garbled circuits, namely SMCQL [10], Prio [26] and Boura et al. [31], offer multiple or advanced functionalities but fail to provide proofs of correct executions. Systems solely based on homomorphic encryption (HE), namely UnLynx [16], Aono et al. [33], Helen [36] and Kim et al. [13], are limited in the functionalities they offer. Furthermore, Aono et al. [33] and Kim et al. [13] rely on data centralization. Gazelle [32] combines HE and garbled circuits and enables complex evaluations of neural networks, but does not protect DPs' privacy or provide computation correctness. Contrarily, Drynx enables multiple operations while distributing trust, computations, and data storage, and it provides strict security guarantees in a stronger adversarial model.

We quantitatively compare Drynx to Unlynx [16] and Prio [26], which are, to the best of our knowledge, the closest prior works. Drynx's query execution time for the `sum` is faster than UnLynx, as we improved the $CTKS$ protocol by enabling its execution in a tree fashion, thus reducing its execution complexity from $O(\#CN)$ to $O(log(\#CN))$. Unlike UnLynx, Drynx

enables the verification of DPs' value ranges, which, for the computation of a `sum`, adds an overhead of only 0.6 seconds (out of a total time of 2 seconds, as depicted in Figure 7). However, Drynx enables a faster scalable verification of proofs by an auditor. After the proofs are verified and the results stored in the *proof blockchain*, an auditor can simply request and verify the corresponding block, which in this case takes approximately 0.4s. In Unlynx, an auditor has to request the proofs from each entity and verify them by itself, which takes 1.4s.

		SMCQL	UnLynx	Boura et al.	Prio	Aono et al.	Kim et al.	Gazelle	Helen	Drynx
System Properties	Distribution of Trust	X	✓	✓	✓	X	X	2-party	✓	✓
	Distrib. Comput. / Stor.	✓	✓	✓	✓	X	X	2-party	✓	✓
	Modular architecture	X	X	✓	✓	X	X	X	X	✓
Security Guarantees (assuming Drynx's threat model)	Data Confidentiality	X	✓	✓	✓	✓	✓	✓	✓	✓
	DPs' Privacy	X	✓	X	X	X	X	X	~	✓
	Comput. Correctness	X	✓	X	X	X	X	X	✓	✓
	Results' Robustness	X	X	X	X	X	X	X	✓	✓
Functionalities Accuracy|AUC	Statistics	✓	X	X	✓	X	X	X	X	✓
	Lin. Reg. Training	X	X	✓	✓	~	~	X	✓	✓
	Log. Reg. Training SPECTF [55] Pima [48] LBW [38] PCS [51]	-	X	✓	~	✓ 75.4|0.76 75.4|0.87 - -	✓ - - 69.3|0.66 69.1|0.75	X	X	✓ 74.8|0.73 77.5|0.83 70.2|0.73 75.1|0.81
	Neural Networks	X	X	X	X	X	X	✓	X	X

(a) Solutions Comparison. For log. reg., we split the datasets [87], [88], [89], [90] among 10 DPs before standardization, scale factor $= 10^2$ for fixed-point represent., learning rate 0.1; 80% train., 20% test.



(b) Comparison with Prio for a *min* query.

Fig. 12: Drynx's comparisons.

Prio [26] relies on secret-shared non-interactive proofs that are created by the DPs to prove the correctness of their inputs to the system and that are collectively verified by the CNs. Even though both systems have similar functionalities, Prio provides input-range verification and computation correctness only when all the CNs are honest-but-curious. We adapted the Gorrigan-Gibbs prototype implementation [91] of Prio to a similar deployment environment as Drynx so that both use the same communication settings, thus enabling a fair comparison. In Figure 12b, we compare Prio's runtime in an illustrative example by using the `min` operation on the range $[0,1000)$ with increasing number of CNs and DPs, against multiple settings of Drynx. This figure shows that Drynx significantly outperforms Prio when computing `min` without using obfuscation (CTO) hence accepts a small probability of error $(1/(\#\mathcal{G}))$ and avoids the need for range proofs. If we use obfuscation, Drynx scales similarly as Prio, but it must be noted that Drynx performs its operations in a stronger threat model. When used in Prio's threat model (delimited by a black line), Drynx is about two times faster. This is because each range proof can be sent and verified by a single VN as all VNs are considered honest-but-curious under Prio's threat model.

X. Conclusion

We have proposed Drynx, a novel system that enables a querier to compute statistics and train machine-learning models on distributed datasets in a strong adversary model where no entity is individually trusted. Drynx provides query-execution auditability and ensures the end-to-end confidentiality of the data. It protects the privacy of the data providers and relies on an immutable and distributed ledger to provide efficient correctness verification and proofs storage. Drynx is highly modular, offering configurable tradeoffs between security, privacy, and efficiency. Finally, Drynx enables privacy-preserving computations of widely-used statistics on sensitive and distributed data, thus offering features that are absolutely needed in crucial areas such as user-behavior analysis or research for personalized medicine.

References

[1] "Big Data Privacy is a Bigger Issue Than You Think." https://www.techrepublic.com/article/big-data-privacy-is-a-bigger-issue-than-you-think (25.06.2018).
[2] "GDPR," https://www.eugdpr.org (25.07.2018).
[3] "A new data breach may have exposed ... every American adult," https://tinyurl.com/ydz7jpdk (4.02.2019).
[4] "Equifax Breach," https://tinyurl.com/y9h4pgsk (4.02.2019).
[5] V. Bindschaedler, R. Shokri, and C. A. Gunter, "Plausible deniability for privacy-preserving data synthesis," *VLDB*, vol. 10, no. 5, 2017.
[6] X. Hu, M. Yuan, J. Yao, Y. Deng, L. Chen, Q. Yang, H. Guan, and J. Zeng, "Differential Privacy in Telco Big Data Platform," *VLDB*, vol. 8, no. 12, 2015.
[7] N. Johnson, J. P. Near, and D. Song, "Towards Practical Differential Privacy for SQL Queries," *VLDB*, vol. 11, no. 5, 2018.
[8] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing," in *SOSP*. ACM, 2011.
[9] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," in *VLDB*, vol. 6, 2013.
[10] J. Bater, G. Elliott, C. Eggen, S. Goel, A. Kho, and J. Rogers, "SMCQL: Secure Querying for Federated Databases," *VLDB*, vol. 10, no. 6, pp. 673–684, 2017.
[11] R. Chen, A. Reznichenko, P. Francis, and J. Gehrke, "Towards Statistical Queries over Distributed Private User Data." in *NSDI*, vol. 12, 2012, pp. 13–13.
[12] K. A. Jagadeesh, D. J. Wu, J. A. Birgmeier, D. Boneh, and G. Bejerano, "Deriving Genomic Diagnoses without Revealing Patient Genomes," *Science*, vol. 357, no. 6352, pp. 692–695, 2017.
[13] M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang, "Secure Logistic Regression Based on Homomorphic Encryption: Design and Evaluation," *JMIR*, 2018.
[14] L. Melis, G. Danezis, and E. De Cristofaro, "Efficient Private Statistics with Succinct Sketches," *NDSS*, 2015.
[15] J. L. Raisaro, J. Troncoso-Pastoriza, M. Misbach, J. S. Sousa, S. Pradervand, E. Missiaglia, O. Michielin, B. Ford, and J.-P. Hubaux, "Medco: Enabling Secure and Privacy-Preserving Exploration of Distributed Clinical and Genomic Data," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2018.
[16] D. Froelicher, P. Egger, J. S. Sousa, J. L. Raisaro, Z. Huang, C. Mouchet, B. Ford, and J.-P. Hubaux, "UnLynx: A Decentralized System for Privacy-Conscious Data Sharing," *PoPETS*, vol. 2017, no. 4, pp. 232–250, 2017.
[17] D. B. Baker, J. Kaye, and S. F. Terry, "Governance through privacy, fairness, and respect for individuals," *eGEMs*, vol. 4, no. 2, 2016.
[18] X. Dong, J. Yu, Y. Luo, Y. Chen, G. Xue, and M. Li, "Achieving an Effective, Scalable and Privacy-Preserving Data Sharing Service in Cloud Computing," *Computers & security*, vol. 42, pp. 151–164, 2014.
[19] X. Liu, Y. Zhang, B. Wang, and J. Yan, "Mona: Secure Multi-Owner Data Sharing for Dynamic Groups in the Cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1182–1191, 2013.
[20] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious Multi-Party Machine Learning on Trusted Processors," in *USENIX Security Symposium*, 2016, pp. 619–636.
[21] D. Bogdanov, L. Kamm, B. Kubo, R. Rebane, V. Sokk, and R. Talviste, "Students and Taxes: A Privacy-Preserving Study using Secure Computation," *PoPETS*, vol. 2016, no. 3, pp. 117–135, 2016.
[22] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A Framework for Fast Privacy-Preserving Computations," in *European Symposium on Research in Computer Security*. Springer, 2008, pp. 192–206.
[23] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans, "Privacy-Preserving Distributed Linear Regression on High-Dimensional Data," *PoPETS*, vol. 2017, no. 4, pp. 345–364, 2017.
[24] R. Shokri and V. Shmatikov, "Privacy-Preserving Deep Learning," in *Proceedings of the 22nd ACM SIGSAC CCS*, 2015.
[25] H. Yang, W. Shin, and J. Lee, "Private information retrieval for secure distributed storage systems," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 12, pp. 2953–2964, 2018.
[26] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, Robust, and Computation of Aggregate Statistics." in *NSDI*, 2017, pp. 259–282.
[27] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-Preserving Ridge Regression on Hundreds of Millions of Rs," in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 334–348.
[28] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan, "Big Data Analytics over Encrypted Datasets with Seabed." in *OSDI*, 2016, pp. 587–602.
[29] M. Du, Q. Wang, M. He, and J. Weng, "Privacy-preserving indexing and query processing for secure dynamic cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 9, pp. 2320–2332, 2018.
[30] P. Mohassel and Y. Zhang, "SecureML: A System for Scalable Privacy-Preserving Machine Learning," in *38th IEEE Symposium on Security and Privacy*, 2017.
[31] C. Boura, I. Chillotti, N. Gama, D. Jetchev, S. Peceny, and A. Petric, "High-precision privacy-preserving real-valued function evaluation," *FC '18*.
[32] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A Low Latency Framework for Secure Neural Network Inference," *arXiv preprint arXiv:1801.05507*, 2018.
[33] Y. Aono, T. Hayashi, L. Trieu Phong, and L. Wang, "Scalable and Secure Logistic Regression via Homomorphic Encryption," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. ACM, 2016.
[34] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018.

[35] Q. Jia, L. Guo, Z. Jin, and Y. Fang, "Preserving model privacy for machine learning in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[36] W. Zheng, R. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2019, pp. 915–929.

[37] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC CCS*, 2016.

[38] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *arXiv preprint arXiv:1712.07557*, 2017.

[39] K. Chaudhuri and C. Monteleoni, "Privacy-preserving logistic regression," in *Advances in neural information processing systems (NIPS)*, 2009.

[40] M. Pathak, S. Rane, and B. Raj, "Multiparty differential privacy via aggregation of locally trained classifiers," in *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[41] M. Kim, J. Lee, L. Ohno-Machado, and X. Jiang, "Secure and differentially private logistic regression for horizontally distributed data," *IEEE Transactions on Information Forensics and Security (TIFS)*, 2019.

[42] B. Jayaraman, L. Wang, D. Evans, and Q. Gu, "Distributed learning without distress: Privacy-preserving empirical risk minimization," in *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[43] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *ACM Conference on Computer and Communications Security (CCS)*, 2015.

[44] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *IEEE Symposium on Security and Privacy (SP)*, 2017.

[45] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine learning*, 2011.

[46] Z. Huang, R. Hu, Y. Guo, E. Chan-Tin, and Y. Gong, "Dp-admm: Admm-based distributed learning with differential privacy," *IEEE Transactions on Information Forensics and Security (TIFS)*, 2019.

[47] B. Jayaraman and D. Evans, "Evaluating differentially private machine learning in practice," in *USENIX Security Symposium*, 2019.

[48] "GA4GH," https://genomicsandhealth.org (30.11.2018).

[49] J. V. Selby, A. C. Beal, and L. Frank, "The patient-centered outcomes research institute (pcori) national priorities for research and initial research agenda," *Jama*, vol. 307, no. 15, pp. 1583–1584, 2012.

[50] "SPHN," https://www.sphn.ch/en.html (29.10.2018).

[51] "HealthLNK," https://tinyurl.com/y7dqhws6 (29.01.2019).

[52] "Apple Watch Heart Monitoring," https://tinyurl.com/y7ctnauc (29.01.2019).

[53] "P4MI," http://p4mi.org (29.01.2019).

[54] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[55] J. Camenisch and M. Stadler, "Proof Systems for General Statements about Discrete Logarithms," *Technical report/Dept. of Computer Science, ETH Zürich*, vol. 260, 1997.

[56] J. Camenisch and R. Chaabouni, "Efficient Protocols for Set Membership and Range Proofs," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2008, pp. 234–252.

[57] A. Fiat and A. Shamir, "How to Prove Yourself: Practical Solutions to Identification and Signature Problems," in *Advances in Cryptology—CRYPTO'86*. Springer, 1986, pp. 186–194.

[58] L. Kokoris-Kogias, L. Gasser, I. Khoffi, P. Jovanovic, N. Gailly, and B. Ford, "Managing Identities using Blockchains and CoSi," in *HotPETs*, 2016.

[59] S. Nakamoto, "Bitcoin: A Peer-To-Peer Electronic Cash System," 2008.

[60] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[61] T.-T. Kuo, H.-E. Kim, and L. Ohno-Machado, "Blockchain distributed ledger technologies for biomedical and health care applications," *Journal of the American Medical Informatics Association*, vol. 24, no. 6, pp. 1211–1220, 2017.

[62] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[63] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus in the lens of blockchain," *arXiv preprint arXiv:1803.05069*, 2018.

[64] C. Dwork, "Differential Privacy," *Encyclopedia of Cryptography and Security*, pp. 338–340, 2011.

[65] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*. Springer, 2006, pp. 265–284.

[66] A. Ghosh, T. Roughgarden, and M. Sundararajan, "Universally utility-maximizing privacy mechanisms," *SIAM Journal on Computing*, vol. 41, no. 6, pp. 1673–1693, 2012.

[67] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[68] C. A. Neff, "A Verifiable Secret Shuffle and its Application to E-Voting," in *Proceedings of the 8th ACM conference on Computer and Communications Security*. ACM, 2001, pp. 116–125.

[69] ——, "Verifiable Mixing (Shuffling) of ElGamal pairs," *VHTi Technical Document*, 2003.

[70] S. Bayer and J. Groth, "Efficient zero-knowledge argument for correctness of a shuffle," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 263–280.

[71] J. Groth, "A verifiable secret shuffe of homomorphic encryptions," in *International Workshop on Public Key Cryptography*. Springer, 2003, pp. 145–160.

[72] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, "Scalable Anonymous Group Communication in the Anytrust Model," DTIC Document, Tech. Rep., 2012.

[73] Y. Lindell, "How to simulate it–a tutorial on the simulation proof technique," in *Tutorials on the Foundations of Cryptography*. Springer, 2017, pp. 277–346.

[74] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[75] "OpenID Connect," https://openid.net/connect/ (04.09.2019).

[76] "OAuth 2.0," https://oauth.net (04.09.2019).

[77] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing," in *25th USENIX Security Symposium*, 2016, pp. 279–296.

[78] "Go Programming Language," https://golang.org (25.06.2017).

[79] "Drynx Implementation," https://github.com/ldsec/drynx (19.11.2018).
[80] "DEDIS Research Lab at EPFL, Advanced crypto library for the Go language," https://github.com/DeDiS/crypto (12.08.2018).
[81] K. Nikitin, E. Kokoris-Kogias, P. Jovanovic, N. Gailly, L. Gasser, I. Khoffi, J. Cappos, and B. Ford, "CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds," in *26th USENIX Security*, 2017, pp. 1271–1287.
[82] G. Danezis and S. Meiklejohn, "Centrally Banked Cryptocurrencies," *Proceedings of the 24rd Network and Distributed System Security Symposium*, 2016.
[83] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-Speed High-Security Signatures," *Journal of Cryptographic Engineering*, 2012.
[84] P. S. Barreto and M. Naehrig, "Pairing-Friendly Elliptic Curves of Prime Order," in *International Workshop on Selected Areas in Cryptography*. Springer, 2005.
[85] "Mininet," http://mininet.org (14.08.2018).
[86] L. Lorgis, M. Zeller, P. Jourdain, J. Beaune, J.-P. Cambou, B. Vaisse, B. Chamontin, and Y. Cottin, "Heart rate distribution and predictors of increased heart rate among French hypertensive patients with stable coronary artery disease. data from the lhycorne cohort," *Archives of cardiovascular diseases*, 2009.
[87] "SPECTF," https://archive.ics.uci.edu/ml/datasets/SPECTF+Heart (14.04.2018).
[88] "Pima Indians Diabetes Dataset," https://tinyurl.com/y8o3x8me (14.04.2018).
[89] "Prostate Cancer Data," https://tinyurl.com/ycsc8f9d (14.03.2018).
[90] "Low Birth Weight Dataset," https://tinyurl.com/yd6mclh6 (21.07.2018).
[91] "Prio Implementation," https://github.com/henrycg/prio (1.07.2018).

APPENDIX A
TABLE OF SYMBOLS

Symbol	Description
HDS, PDS	Hospitals & Patients Data Sharing
$\mathcal{G}, B, p$	Elliptic curve; base point on $\mathcal{G}$, prime
$\mathrm{E}_\Omega(m) = (C_1,C_2)$ $=(rB, mB+r\Omega)$	ElG encrypt. of m under key Ω, nonce r
K	CNs pub. coll. key
(k_i, K_i)	CNs CN_i priv., pub. key
A, A_1, A_2, Y_i, y_i	ZKPs pub. (uppercase), discrete log.
Q, DP, N	Querier, Data Provider, $\#DP$
CN, VN	Computing & Verifying Node
f_h	Threshold of honest VNs
$\pi, \rho, \bar{r_i}$	linear combi., *encoding*, records
$\mathbf{V_i} = [v_{i,1},...,v_{i,l}], c_i$	vector, count
$CTA, CTO, CTKS$	Coll. Tree Aggr., Obfusc., Key Switch.
$w_{i,1}, w_{i,2}$	CN_i's contribution in $CTKS$
α_i, s_i	CN_i secret random nonce
$CDP, (\epsilon, \delta, \theta)$	Coll. Diff. Privacy & params.
$[b_l, b_u]$, $[0, u^l)$	Range, default range
$x_i, (A_{i,j}, Z_i, H, V_{i,j}, a_{i,j})$	Range proof priv., pub. values
T, T_{sub}	Proofs and sub-proofs verif. thresh.
N_{da}, N_i, D	Tot. & DP_i #records, dataset dim.
$p_{ver}, p_{ver_{sub}}$	proof, sub-proof
P_{f_h}	prob. of f_h VNs verif.

TABLE III: Table of Recurrent Symbols.

APPENDIX B
ERROR PROBABILITY

In Section VII, we notice that the result of bit-wise operations, when DPs are requested to answer with random values R_is, can be erroneous with a probability smaller than $1/(\#G-1)$. We demonstrate here this result and provide an expression for the probability of error P_n where n is the number of DPs.

$$
\begin{aligned}
P_n &= P(\sum_{i=1}^{n} R_i = 0) = \sum_{a=0}^{\#G-1} P(\sum_{i=1}^{n} R_i = 0 | \sum_{i=1}^{n-1} R_i = a) \cdot P(\sum_{i=1}^{n-1} R_i = a) \\
&= P(\sum_{i=1}^{n} R_i = 0 | \sum_{i=1}^{n-1} R_i = 0) \cdot P(\sum_{i=1}^{n-1} R_i = 0) \\
&+ \sum_{a=1}^{\#G-1} P(\sum_{i=1}^{n} R_i = 0 | \sum_{i=1}^{n-1} R_i = a) \cdot P(\sum_{i=1}^{n-1} R_i = a) \\
&= \sum_{a=1}^{\#G-1} P(\sum_{i=1}^{n} R_i = 0 | \sum_{i=1}^{n-1} R_i = a) \cdot P(\sum_{i=1}^{n-1} R_i = a) \\
&= P(R_n = -a) \cdot \sum_{a=1}^{\#G-1} P(\sum_{i=1}^{n-1} R_i = a) \\
&= \tfrac{1}{\#G-1} \cdot \sum_{a=1}^{\#G-1} P(\sum_{i=1}^{n-1} R_i = a) = \tfrac{1}{\#G-1} \cdot (1 - P_{n-1}).
\end{aligned}
$$

We have $P_n = \frac{1}{\#G-1} \cdot (1 - P_{n-1}) \leq \frac{1}{\#G-1}$ and $P_n = \sum_{i=2}^{n} (-1)^i \cdot (\frac{1}{\#G-1})^{i-1}$.