d'Artagnan: A Trusted NoSQL Database on Untrusted Clouds

Rogério Pontes*, Francisco Maia†, Ricardo Vilaça‡
HASLab - High-Assurance Software Lab
INESC TEC & U. Minho, Portugal
Email: {*rogerio.a.pontes, †francisco.a.maia, ‡ricardo.p.vilaca}@inesctec.pt

Nuno Machado
Teradata & INESC TEC
Email: nuno.a.machado@inesctec.pt

***Abstract*—Privacy sensitive applications that store confidential information such as personal identifiable data or medical records have strict security concerns. These concerns hinder the adoption of the cloud. With cloud providers under the constant threat of malicious attacks, a single successful breach is sufficient to exploit any valuable information and disclose sensitive data. Existing privacy-aware databases mitigate some of these concerns, but sill leak critical information that can potently compromise the entire system's security.**

This paper proposes d'Artagnan, the first privacy-aware multi-cloud NoSQL database framework that renders database leaks worthless. The framework stores data as encrypted secrets in multiple clouds such that *i)* a single data breach cannot break the database's confidentiality and *ii)* queries are processed on the server-side without leaking any sensitive information. d'Artagnan is evaluated with industry-standard benchmark on market-leading cloud providers.

I. Introduction

Motivation. To this day, commercial databases still store and process data with mediocre security guarantees. Despite the existence of data access control safeguards [1], a single breach is usually sufficient to compromise the entire system [2]. Moreover, with the shift from private infrastructures to cloud environments, databases are far more exposed to daily attacks. In 2018 alone, cloud services were attacked 1.5 million times every day [3], with a successful data breach costing on average 3.86 million dollars [4]. This cost is expected to rise in the future, with an average data breach reaching the 150 million dollars in 2020 [5].

To minimize the database's vulnerability to attacks, sensitive data should never be stored or processed as plaintext. Unfortunately, query processing over ciphertexts poses a long-standing problem. Recent efforts have tackled this challenge with novel property-preserving encryption (PPT) schemes that sacrifice security for efficient secure processing [6]. The trade-off consists in leaking some partial information, such as the equality or order [7] of ciphertexts, to enable the database engine to process queries without fully disclosing the plaintexts. State-of-the-art privacy-preserving databases such as CryptDB [8] and SQL Server Always Encrypted [9] follow this approach.

However, privacy-aware databases are no panacea, as the same security guarantees that make PPT schemes attractive end up compromising the user's confidentiality. For example, a common attack vector consists in performing frequency analysis on datasets without a uniform distribution. Notably, prior work has shown that, depending on the dataset, more than 50% of the data can be disclosed by correlating information from different columns [6]. Inference attacks are also effective at disclosing sensitive data with more than 90% accuracy from a single database snapshot encrypted with PPT schemes [10].

Our Approach. This paper tackles the limitations of privacy-aware databases with a novel approach that makes no security compromises. In a nutshell, we propose d'Artagnan, a secure multi-cloud NoSQL database framework that decentralizes information by encrypting data in secrets and storing them in independent databases, each hosted on a different cloud provider. A single secret does not leak any information and the original plaintext value can only be decrypted if the majority of the secrets are obtained by a single entity (e.g.: client application). Furthermore, with these secrets d'Artagnan can process any query on the server-side by evaluating secure cryptographic protocols. An immediate result of such system is that a data breach of a single cloud is inconsequential and, for any significant information to be obtained, the majority of the clouds that constitute d'Artagnan must be compromised. Clearly, the effort required to break the system grows with the number of cloud providers used.

At the core of d'Artagnan are secret sharing schemes and secure multi-party protocols (SMPC) [11]. Secret sharing schemes encrypt data in multiple secrets while SMPC protocols enable a set of independent parties (clouds) to evaluate functions over such secrets. These cryptographic techniques have been overlooked by SQL and NoSQL databases as a practical alternative to PPT schemes even though they can assure security guarantees unmatched by state-of-the-art privacy-aware databases. One of the hindering factors that prevents the widespread use of such protocols is the highly-specialized cryptographic knowledge necessary to integrate them in a real-world system. Furthermore, coordinating the multiple parties of an SMPC protocol to process database queries raises its own set of challenges.

Contributions. Our contributions address this gap with the design of a high-level, modular and extensible multi-cloud database framework that encrypts data in secrets and process queries with SMPC protocols. The system hides the details of the underlying cryptographic protocols and the distributed execution of secure queries under a standardized NoSQL API. Note that we chose to focus on key-value NoSQL systems to

distill d'Artagnan's architecture in a set of core components applicable to any database, including SQL databases. In summary, our contributions are:

- A novel multi-cloud NoSQL database framework (d'Artagnan) that decentralizes confidential data over multiple independent parties and leverages SMPC protocols to process queries. The framework is designed with a modular and extensible architecture that enables the integration of distinct SMPC protocols tailored fit to application specific use-cases. Furthermore, the framework is open-sourced [1].
- A NoSQL database prototype built on the d'Artagnan framework that leverages Apache HBase as underlying database and the state-of-the-art SMPC protocols to securely process queries over encrypted data.
- A thorough evaluation of the prototype under different environment including a real-word deployment on the market-leading cloud providers (namely Google, Amazon, Azure and Digital Ocean). The experiments shows that d'Artagnan's lowest overhead is 39%.

Layout. Section II introduces the fundamental concepts of NoSQL databases and SMPC. Section III defines d'Artagnan's security model. Section IV presents the architecture and Section V describes how every component in the system interacts to securely process queries. Section VI describes the prototype implementation. Section VII presents and discusses the experimental evaluation. Finally, Section VIII overview the related work and Section IX concludes the paper.

II. Background

This section presents key concepts on key-value NoSQL databases and SMPC protocols that are crucial to understand d'Artagnan's architecture.

A. NoSQL databases

Key-value NoSQL databases are d'Artagnan's storage backend. These databases are suitable for applications that handle eventually consistent data, support lack of database-wide transactions and require a flexible data scheme [12]. Conceptually, data is stored in named multi-dimensional maps, similar to hash tables, where values are indexed by a unique key and a column. The map data structure does not follow a static schema and is dynamically adjusted as new rows are inserted. Clients interact with the database to store or process data through an interface similar to the following [13]:

- **Put**$(Table, Column, Key, Value)$ - Given a table, store a value indexed by the row key and column.
- **Update**$(Table, Column, Key, Value)$ - Given a table, update the record indexed by the row key and column with the specified value.
- **Get**$(Table, Column, Key)$ - Retrieve the value of a table indexed by a row key and a column.
- **Delete**$(Table, Column, Key)$ - Delete the value of the table indexed by a row key and a column.
- **Scan**$(Table, Start, Stop)$ - Retrieve the records of a table whose index is greater than or equal to the identifier $Start$ and lower than the identifier $Stop$.
- **Filter**$(Table, Condition)$ - Given a table, retrieve all the records that validate the condition predicate. The condition can specify columns, propositional formulas, and regular expressions.

Generally, NoSQL databases trade strict consistency and transactional support for a highly scalable, distributed architecture [12], [14]. This trade-off is key to design "shared nothing" databases that scale horizontally by partitioning data into *shards*, subsets of table rows uniquely defined by a first and last row key. Shards are created as a table grows, replicated and balanced between a pool of *computing nodes*. Every node manages a subset of the database shards and directly answers client queries without requiring a proxy master node. In case of a node failure, the remaining live nodes can take over shards while a new node is added to the cluster.

B. Secure Multi-Party Protocols

Secure multi-party protocols enable a set of independent parties to jointly compute *any* function f on their private inputs without disclosing information besides an agreed upon output [11]. Currently, there are two main approaches to build an SMPC protocol: secret sharing and garbled circuits [11]. Both rely on compiling functions into secure circuits composed of addition and multiplication gates [11]. The compiled circuits can be viewed as one-way functions that specify a secure protocol to be run by the parties to obtain a common result. However, garbled circuits are limited to only two participating parties and require expensive public-key operations [15]. As such, d'Artagnan focus on secret sharing SMPC protocols that use simple arithmetic operations and can support an arbitrary number of parties.

Secret Sharing enables the construction of SMPC protocols for an arbitrary number of parties [16]. Consider an extended version of the classical Yao's millionaire problem where the parties are n millionaires that wish to know who is the richest without ever revealing their wealth to each other. In this scenario, every party starts by encrypting its private input (e.g.: the millionaire wealth) with a (n,t)-secret sharing scheme that creates n shares in an arithmetic field [17]. Every individual share is a seemingly random ciphertext that has no relevant information by itself. The original private inputs can only be recovered if more than t shares are brought together in a single party. After a party encrypts its private input in multiple shares it sends a single share to every party participating in the protocol. Once the shares are exchanged, the parties start to evaluate an agree upon circuit to jointly compute the function result. The evaluation of every circuit gate takes secret shares as inputs and outputs new secret shares that hide a secret result value. The output of a gate is the input of the next gate until the entire circuit is evaluated. Since shares are values of an arithmetic field, addition gates can be locally computed by the parties without requiring any communication [18]. However, multiplication gates require parties to communicate

[1]https://dbr-haslab.github.io/tools/dartagnan

and exchange shares to reach a correct secret result. When every gate is evaluated, the output of the final gate returns a single share of the function result. Parties exchange their resulting share with each other to disclose and learn the final function result.

Besides confidentiality, SMPC protocols ensure additional security properties such as: *fairness*, *output correctness*, *output delivery* and *identifiable aborts*. These properties provide additional guarantees to the parties if a malicious adversary is present. The goal of the adversary is to corrupt honest parties, parties that correctly follow a protocol, in an attempt to compromise a protocol confidentiality or execution. For instance, in the millionaire's problem, a corrupted adversary may attempt to disclose another party's wealth or prevent some parties to learn the final result. Informally, the properties ensured by SMPC protocols are defined as follows [19]:

- **Fairness** - Corrupted parties only receive a protocol output if all honest parties also receive their outputs.
- **Output Correctness** - Every party receives a correct output.
- **Output Delivery** - Corrupted parties cannot prevent honest parties from receiving their output.
- **Identifiable Aborts** - Honest parties receive identification of corrupted parties.

III. Threat Model

Before presenting d'Artagnan's architecture we describe the framework's security guarantees. d'Artagnan protects the user's confidentiality by decentralizing data between multiple clouds, each hosting an individual database. In a n-cloud system, d'Artagnan encrypts sensitive values with a (n,t)-secret sharing scheme and stores each share on a single database. These schemes ensure that *data at rest*, data not processed by queries, remains secure as long as no more than t clouds are compromised. Even if a subset of the cloud providers leak multiple database snapshots, its proven to be impossible to decrypt the original values [17].

Besides protecting data at rest, d'Artagnan's goal is to protect user's confidentiality during query processing. In this case, d'Artagnan security guarantees rely on the properties and trust model of SMPC protocols to protect data from external attackers and malicious insiders. An external attacker is anyone who attempts to infiltrate a cloud provider or eavesdrop on the communication of a protocol execution. A malicious insider is any cloud or database administrator that uses privileged access to read or leak data. Both adversaries can have different behaviors after successfully corrupting a cloud provider. For instance, an adversary may either decide to actively corrupt a protocol execution by modifying the circuit gate evaluation, and risk detection, or act cautiously and only read available data while attempting to break the secrets in a different infrastructure. Depending on its behavior, the adversary is modeled as follows [11]:

- **Semi-honest adversary** has access to a corrupted party internal state and reads every message received or sent from/to a party. However, the adversary does not modify in any way the protocol execution.
- **Malicious adversary** corrupts a fixed set of parties and actively attempts to break a protocol confidentiality by modifying the corrupted parties' behavior.

d'Artagnan is designed to leverage different SMPC protocols to protect user's data against these adversaries. Consequentially, the system's security guarantees acquire the same properties as the protocols used. Furthermore, depending on the adversary the protocols ensure slightly different security properties. Let us first consider a client that entrusts the cloud providers to protect an application from external attacks. The client's main concern is preventing a semi-honest malicious adversary, a malicious insider, from accessing the databases' data. Against this adversary, d'Artagnan leverages SMPC protocols that guarantee not only the user's confidentiality during query processing but also fairness, output correctness and output delivery, even if every cloud has a malicious insider [20].

The threat escalates when the cloud providers' infrastructures are not safe from an external attacker. In this case, the attacker's attempts to corrupt multiple clouds providers. For instance, the attacker explores different attack vectors such as SSL/TLS exploits (e.g: Heartbleed) [21], Hypervisor attacks [22] and even hardware faults such as Meltdown [23]. If the attacker successfully corrupts a cloud provider, it gains the same control as a malicious insider. In this model, d'Artagnan can securely process queries with SMPC protocols that ensure fairness, output correctness, output delivery and identifiable aborts if the attacker cannot successfully corrupt the majority of the cloud providers [16]. However, without an honest majority the system is limited to protocols that only ensure confidentiality [16], [11]. Nonetheless, d'Artagnan has a significant advantage over state-of-the-art privacy-aware databases in both models. It forces the adversary to spend far more resources than those necessary to break into a single cloud deployment.

IV. Architecture

d'Artagnan's architecture, depicted in Figure 1, coordinates independent key-value databases, hosted at different cloud providers, to create a logical NoSQL database capable of processing queries over encrypted data. This decentralized approach prevents a single cloud provider or a database vulnerability from corrupting and compromising the entire system's security guarantees. When taken to the limit, it is entirely possible to have a system deployment where the first cloud (Party 1) is a BigTable database on Google's Cloud, the second cloud (Party 2) hosts a Dynamo DB in Amazon AWS and the remaining clouds host entirely different databases.

From a high-level perspective, the framework operates across a *trusted domain* and an *untrusted domain*. The *trusted domain* is the system's entry-point where the client application resides sheltered from attacks. This domain, possibly a private trusted infrastructure, is where sensitive data is encrypted before being outsourced to the cloud. The *untrusted domain*

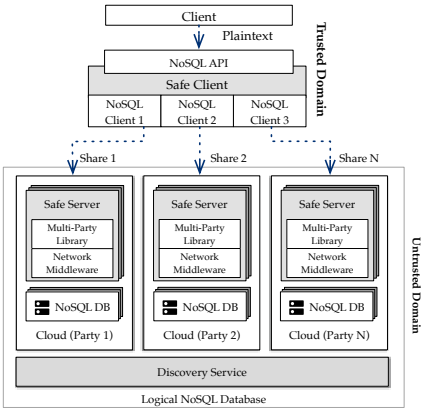

Fig. 1. d'Artagnan architecture has two parts: a *Safe Client* and a *Logical Database*. The Safe Client resides on a trusted domain and intercepts client requests. The Logical Database consists of independent NoSQL databases hosted at independent cloud providers. Cloud providers are considered an untrusted domain. In both parts, the gray boxes refer to d'Artagnan components and white boxes represent unmodified third-party components.

is where all the data storage and query processing take place. This domain has two or more clouds, each playing the role of a SMPC protocol party. Every party (cloud) hosts an autonomous key-value NoSQL database which has no knowledge of the remaining parties. Without d'Artagnan's components, the party's databases are nothing more than a storage system that holds encrypted data. This section describes these components and their role on the framework architecture.

A. Trusted domain

The first component is the **Safe Client**, a privacy-preserving layer between the trusted and untrusted domain. This component has two main goals, abstract the multiple underlying NoSQL databases and protect sensitive data. The first goal is accomplished by exposing an high-level NoSQL API, as defined in Section II-A, that enables applications to use d'Artagnan as a single NoSQL database. The high-level queries are transformed into multiple low-level requests made to the untrusted domain. However, before any query is sent to the untrusted domain, the `Safe Client` encrypts sensitive user information. The client can specify, on a column basis, the SMPC protocols used to process queries. For instance, a client can choose to protect a subset of table columns with protocols that ensure *confidentiality* and *fairness* guarantees while another subset is protected with protocols that just ensure *confidentiality*. To ensure different security guarantees each protocol implements a different secret sharing scheme. As such, the `Safe Client` delegates the encryption and decryption of data to external software libraries that implement the following API:

- **Encode**$(secret, n, k) \rightarrow [shares]$ – Encode a secret in n shares such that only k shares are required to decode the secret. This function returns a list of shares.
- **Decode**$([shares]) \rightarrow secret$ – Given a list of shares, decode them and return the secret.

B. Untrusted Domain

The **Safe Server** component is the processing engine of the untrusted domain. This component is a distributed layer with multiple nodes, at least one node per party, that intercepts high-level `Safe Client` requests and converts them into secure operations. The conversion process depends on the client request, the security model and the underlying databases, but it ensures that every client key-value NoSQL query is converted into a sequence of database-specific requests and SMPC protocols that have a semantically identical functionality with additional security guarantees. Overall, the `Safe Server` initializes the necessary resources and leverages every available component, the `Multi-Party Library`, `Network Middleware`, `Discovery Service` and the underlying database to securely process client requests. A detailed description of the query transformation process and interaction of each component is presented in Section V.

The **Multi-Party Library** component contains the implementation of the secure multi-party protocols. This component abstracts the details of protocol implementations from the `Safe Server` with a high-level interface. Let $p = \{s_{p0}, s_{p1}, \ldots s_{pn}\}$ and $k = \{s_{k0}, s_{k1}, \ldots s_{kn}\}$ be two secret shared values and $p \circ k$ a comparison operation such that $\circ \in \{=, <, >, \leq, \geq\}$. The multi-party library API supports the following operation set $\bullet$:

- **Equal**(s_{pi}, s_{ki})
- **LessThan**(s_{pi}, s_{ki})
- **GreaterThan**(s_{pi}, s_{ki})
- **LessThanOrEqualTo**(s_{pi}, s_{ki})
- **GreaterThanOrEqualTo**(s_{pi}, s_{ki})

such that $p \circ k \equiv s_{pi} \bullet s_{ki}$. This interface has a set of core operators capable of satisfying the high-level key-value NoSQL API. However, this set is extensible and can support additional application-specific functions.

This library also defines a clear boundary between the database execution and the SMPC protocols' implementations. It encapsulates the details of the NoSQL database from the protocol implementations, enabling expert cryptographers to integrate new protocols without having to write database-specific logic. All of the necessary context to integrate new protocols is provided by an execution environment that contains a `Safe Server` party ID and a `Network Middleware` client. The party ID determines how the library evaluates a protocol circuit and the `Network Middleware` client enables the parties to exchange shares.

The **Network Middleware** component establish a mesh network between every party and ensures the protocols exchange shares correctly. The parties can evaluate protocols and communicate via the following network interface:

- **Send**$(playerID, share)$ – send a share to a player.
- **Receive**$(playerID)$ – receives a share from a player.

This simple interface can support a wide range of SMPC protocol implementations and abstracts the `Multi-Party Library` from concurrent protocol executions. At any time,

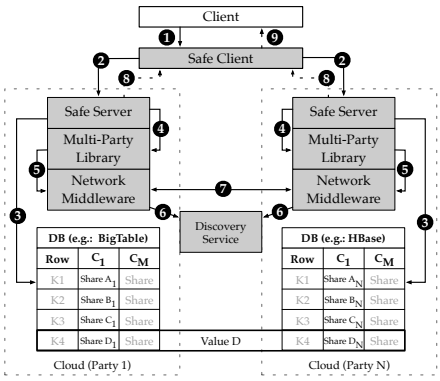

Fig. 2. Interaction of d'Artagnan components to process a client request.

a single `Safe Server` node can process multiple concurrent requests and start parallel SMPC protocols in the `Multi-Party Library`. If concurrent protocol executions have to send shares from party A to party B, the `Network Middleware` multiplexes the shares sent by party A and forwards them to the correct protocol execution environment in party B. With this approach, the protocols in the `Multi-Party Library` only need to implement the circuit's logic without having to deal with concurrency issues.

Besides concurrent protocol execution, the `Network Middleware` also handles the execution of protocols in a dynamic setting. The set of key-value databases that play the role of parties in a protocol is not static and can be added or removed as necessary. For instance, the set of available parties can increase to improve the system availability and performance while ensuring the same security guarantees. As such the `Network Middleware` adapts to the dynamic set of parties and routes the shares to the correct participants by leveraging the `Discovery Service`.

The **Discovery Service** component keeps track of the `Safe Server's` nodes status and location. Every time a node goes online in any of the clouds it notifies the `Discovery Service` with a payload. The payload contains meta-data regarding the node, such as the node's IP address and the database tables it manages. When a node goes offline, it reports its new status to the service. None of the information regarding the shared secrets is kept on the `Discovery Service` and the connections between the nodes are made directly with the `Network Middleware` without using the `Discovery Service` as a proxy. d'Artagnan makes no assumptions on the actual implementation of the `Discovery Service`. Since the meta-data contains no sensitive information, it can be stored on a distributed NoSQL database to avoid a single point of failure and deployed in any public or private infrastructure.

V. Secure Query processing

This section describes how d'Artagnan securely process queries and how every component interacts using a **Filter** operation as example. Figure 2 depicts an illustrative deployment scenario with N clouds to be used throughout this section. The first cloud hosts a BigTable database and the N cloud hosts an HBase database. In this example, d'Artagnan stores a Table T with M columns, from column C_1 to column C_M. In this table only the columns store sensitive data and the row keys are simple plaintext identifiers (e.g: $K1$ and $K2$ in Figure 2).

Consider a client plaintext **Filter**$(T, C_1 == D)$ request on table T to search for every record where the column C_1 has the value D. The table can have several records matching this condition, but, for simplicity purposes, let us consider that only the record with key $k4$ (last row of table T) has value D. Every value stored in the tables is encrypted in shares, with each database storing a single secret. The `Safe Servers` task is to find its party's secrets of the matching rows and send it to the `Safe Client`. Upon receiving all secrets, the `Safe Client` can disclose the result and send it to the client.

The first step in d'Artagnan's execution flow consists in protecting the client's plaintext **Filter** (Figure 2-❶). Upon receiving the request, the `Safe Client` has three main tasks. First, it checks the client's security requirements to select the secret sharing scheme to use. Secondly, it encrypts the search value D in N shares with the proper secret sharing scheme. Finally, it generates N secure **Filters**, each containing a single secret instead of the client's plaintext values. Afterwards, the filters are sent in parallel (Figure 2-❷) to the cloud providers, one filter per party. It's worth noting that the encrypted shares are always random and not deterministic. As such, the shares generated in the `Safe Client` for the value D are different from the shares stored in the databases for the same value. The only way the database can process the incoming request is with an SMPC protocol.

Every party's `Safe Server` intercepts the secure queries and start an execution flow to process the secure **Filter**. The flow is identical in every party and executed in parallel by each party's `Safe Server`. Thus, we describe the execution flow from the viewpoint of party 1 depicted in Figure 2. Upon intercepting the request, the `Safe Server` scans the database (BigTable) in batches. In this example there is only a single batch, from $K1$ to $K4$. Afterwards, it creates an execution environment containing its player ID (1) and a `Discovery Service` client. With the execution environment, the `Safe Server` is ready to find the records that satisfy the **Filter** condition and starts an **Equal** protocol (Figure 2-❹) with the shares stored in column C_1.

The `Multi-Party Library` starts to participate in the protocol execution by evaluating the **Equal** encrypted circuit (Figure 2-❺). Eventually, every party's `Safe Server` reaches this point and also starts to participate in the protocol by invoking the same function with their own shares. The compiled **Equal** circuit is composed of multiplication gates that requires parties to exchange shares during the circuit evaluation. However, before any share can be sent, the `Network Middleware` contacts the `Discovery Service` to store a payload signaling its location and the request it's processing (Figure 2-❻). This information enables each party's `Safe Server` to find its peers in the computation. For instance, if party 1 has to send a share to party N, the `Network Middleware` asks for the IP address of the `Safe Server`

in party N that is processing an **Equal** protocol on table T. In a real execution, additional information is required since there can be multiple concurrent protocols being executed. However, for the `Multi-Party Library` these details are abstracted by the `Network Middleware`. When the parties learn each other's locations, the party's `Network Middleware` establish a communication channel and exchange the necessary shares to evaluate the protocol (Figure 2-❼).

The protocol evaluation either successfully completes or aborts. Protocols abort if the implementation ensures *fairness* guarantees and an active attack is detected. Moreover, a protocol with *identifiable aborts* [19] returns the party ID of the corrupted party. In this scenario, the information is sent to the client that decides the proper course of action. A successful protocol evaluation returns the rows satisfying the **Filter** request. In the example, it's the row with key $K4$ depicted in Figure 2. An attacker also learns this information if a party is corrupted. Even though this information by itself is not sufficient to break the system's confidentiality, it's an open problem that can be addressed with protocols that ensure oblivious execution [24]. The server-side processing ends with every `Safe Server` sending the shares of the rows that satisfy the protocol to the `Safe Client` (Figure 2-❽) which discloses the original row values and returns the correct result to the client (Figure 2-❾).

VI. Implementation

We implemented a complete and fully-functional d'Artagnan prototype. This prototype supports Apache HBase, a scalable, open-source NoSQL relational database [25]. HBase stores data in a multi-dimensional map similar to the NoSQL data model previously presented in Section II. However, tables are partitioned automatically by the system in multiple shards. A shard is a storage unit of a subset of consecutive tables rows that enables the system to scale horizontally by balancing the workload between the database's computing nodes.

A single HBase deployment is a distributed system with two types of nodes: a single Master node and multiple RegionServers (computing nodes). The Master stores meta information regarding cluster configuration and database tables. The RegionServers stores and processes all of the database data. Each RegionServer hosts a set of shards and each shard can only be served by a single RegionServer. The query execution is handled directly between clients and RegionServers.

d'Artagnan's components are implemented in Java. The `Safe Client` prototype provides the same API as the HBase client but manages multiple HBase client connections, one for each party. As described in Section IV, this component transforms plaintext HBase queries into secure HBase requests. However, the `Safe Client` prototype contains a pool of threads that executes the secure requests in parallel. The results of the secure requests are decrypted and aggregated in a single final result forwarded to the client application.

The `Safe Server` component is integrated as an HBase coprocessor, a feature that enables developers to extend RegionServers behavior with plugins. The coprocessors intercept every request sent to a RegionServer and have access to an internal database API capable of modifying the RegionServers core behavior. This approach brings secure query processing closer to the data by removing a network hop between the `Safe Server` nodes and the parties databases.

In this prototype a `Safe Server` node is instantiated per RegionServer. Since each HBase database cluster can have more than a single RegionServer, the role of a single SMPC party is in fact played by multiple `Safe Server` nodes. To manage the multiple nodes as a single party, the `Safe Server` nodes store a payload on the `Discovery Service`, implemented as a Redis [26] database. The payload contains the address of each `Safe Server` node, the party it belongs to and a unique request identifier for each client query. This information enables the `Network Middleware` to discover which `Safe Server` nodes are evaluating an SMPC protocol and establish a mesh network. Even when one or more parties have multiple `Safe Server` nodes, each processing concurrent requests, the `Network Middleware` is able to route shares between the nodes participating in a protocol. The information stored on the `Discovery Service` reveals no information on the actual shares as the communication between the parties is made directly through TCP channels without using the `Discovery Service` as a proxy. Furthermore, all of the data stored on the `Discovery Service` can be cryptographically signed to prevent a malicious attacker from corrupting the information.

The `Multi-Party Library` is implemented in Java and currently supports the protocols proposed by Bogdanov *et al.* [18]. These protocols are among the most efficient in the state-of-the-art and are one of the few protocols applied in the industry to protect critical information [15]. This protocol suit is optimized for three independent parties with a single dealer and ensures confidentiality against a semi-honest adversary.

d'Artagnan inherits the security guarantees of the underlying secure multi-party protocols. As such, this prototype protects the user's confidentiality against malicious insiders and external attackers that do not modify the protocol execution. Although the current prototype does not support protocols that protects the user's confidentiality against a malicious adversary, the integration of such protocols does not pose additional challenges. The current framework can easily leverage new protocols such as the SPDZ Protocol Suite [27] which detects the presence of a corrupted party. Supporting these protocols only requires writing two wrappers: the wrapper for the encode and decode functions in the `Safe Client`; the protocols wrappers for the `Multi-Party Library`. The remaining components, the `Safe Server`, `Network Middleware` and `Discovery Service` are orthogonal to the underlying protocol implementations and require no modifications.

VII. Evaluation

This section presents the evaluation of d'Artagnan's prototype in two experimental settings, including a complete system deployment on public cloud providers. Furthermore, it demonstrates the current performance bottleneck of one of

the most efficient SMPC protocols with an industry-proven benchmark.

A. Experimental Setup

Methodology. The evaluation measures d'Artagnan throughput, operations per second (OP/s), and latency in two different settings. The first is a controlled setting of a fully distributed deployment designed to establish the system's performance baseline. The second scenario validates the prototype with a real-world deployment on the leading cloud providers: Google Cloud Platform [28], Microsoft Azure [29], Digital Ocean [30] and Amazon AWS [31]. HBase is the baseline used in both scenarios.

Use case. We use a synthetic use case of a medical clinic with an appointments table. The table contains the columns: *Physician ID*, *Patient ID*, *Date*, *Type* and *Institution ID*. Records in this table contain private data that must be protected. The information stored on the table reveals the location (*Institution ID*), *Type* (e.g.: Cardiology, Oncology) and date of an appointment. Furthermore, it leaks the relation between a physician and a patient. To protect this information, columns are encrypted with an additive (3,1)-secret sharing scheme. The table records are indexed by a numeric identifier (Key) stored as plaintext. These identifiers reveal no information and are only used to access the data.

Benchmark. For a standard evaluation across environments and NoSQL databases we use the industry-proven Yahoo! Cloud Serving Benchmark (YCSB) [32]. This benchmark has six standard workloads (A-F), each with a different read-write ratio and value distribution. Following the approach commonly seen in related literature [33], [32], we omit the results of workload C as they are identical to the results obtained in workload B. The operators on these workloads only process the plaintext keys and do not measure the overhead of SMPC. However, the workloads are essential to evaluate d'Artagnan's overhead of protecting data with a secret sharing scheme and issuing concurrent requests to multiple clouds. Furthermore, it establishes a direct comparison with a standard HBase deployment without any secure processing.

We designed a new workload G to measure the overhead of SMPC protocols. The workload simulates a clinic use case where the staff frequently browses through the daily appoints but sporadically schedules or updates a new appointment. The workload filters through the $Type$ of appoints using SMPC protocols. The request distribution of every workload is presented in Table I.

One important aspect to the entire evaluation is the value distribution on the Appointments table and how the values for the NoSQL operations are chosen by the YCSB benchmark. On both scenarios the table identifiers are integer values that increase monotonically with every new row. Each table column value is sampled from a uniform distribution of the data type. The same applies for the input values of the **Put**, **Update** and **Get** operators. The **Scan** and **Filter** operators choose a starting key from a uniform distribution and iterate over every row until the last table row. The operator read-modify-write (**R-M-W**) presented in Table I is the composition of a **Get** and **Update** operation.

Workloads	Get	Update	Insert	Scan	R-M-W	Filter
A	50%	50%	-	-	-	-
B	95%	5%	-	-	-	-
D	95%	-	5%	-	-	-
E	-	-	5%	95%	-	-
F	50%	-	-	-	50%	-
G	40%	20%	20%	-	-	20%

TABLE I
BENCHMARK WORKLOADS. EACH WORKLOAD ISSUES A REQUEST WITH THE NOSQL API PRESENTED IN SECTION II. THE OPERATOR $R-M-W$ IS THE COMPOSITION OF A GET AND UPDATE OPERATION.

B. Controlled Setting

Experimental set-up. d'Artagnan's prototype was deployed on a private infrastructure divided in three independent clusters. Each cluster is in itself a distributed HBase deployment with 2 Region Servers, each one with a `Safe Server`. In total, this deployment consisted of 10 nodes, 3 per HBase cluster plus the client machine with the YCSB benchmark that contains the `Safe Client`. Every node had an i3 CPU 4 cores at 3.7 GHz, 8 GB of main memory and a 128 GB SSD. Hosts were connected over a shared Gigabit Ethernet network with an average latency of 0.3 ms.

YCSB workloads. Figure 3 presents the results of the YCSB workloads with the Appointments table containing 1 million rows divided between 20 shards. Each plot has latency versus throughput curves that depict the systems scalability with an increasing number of clients, from 1 to 256 in a logarithmic scale base 2. Each dot ($\times$, $\circ$) in a plot represents an experiment with a different number of clients. Experiments consists of 3 hour runs.

From workload A to F d'Artagnan prototype scales with the number of clients but starts to reach a plateau with 32 clients as requests latency increase at a higher rate than the throughput. For instance, on Workload A d'Artagnan prototype has a throughput increase of 10% from 32 clients to 256 clients but the latency increases 87%. Both d'Artagnan prototype and HBase follow a similar pattern across the workloads with a higher throughput on read-intensive workloads. The highest throughput reached by both systems is found on workload B with d'Artagnan prototype peaking around 17 KOP/s and HBase at 49 KOP/s. With a maximum overhead of 2.88$\times$ the baseline, d'Artagnan's overhead is acceptable considering that for each client, the `Safe Client` has to encrypt every column value with a secret sharing scheme, send three concurrent requests, one per cloud, and wait for all parties to process the request. On the maximum load of 256 concurrent clients, d'Artagnan has 768$\times$ more requests to manage than the baseline.

Workload G on Figure 3 is the first workload to measure the impact of SMPC protocols. Even though only 20% of the workload operations are filters that require SMPC, the protocols have a significant impact on the system throughput.

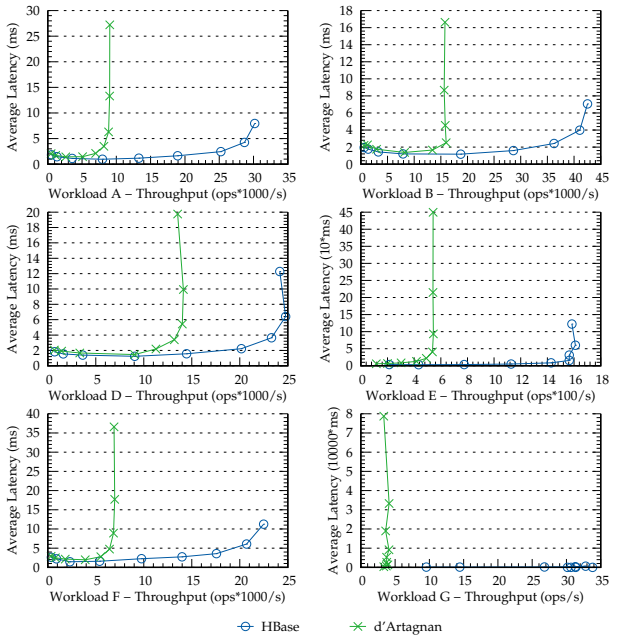

Fig. 3. d'Artagnan and baseline (HBase) performance with the YCSB Workloads. The benchmarks consider a dataset with 1 Million rows with an increasing number of concurrent clients, from 1 to 256 in a logarithmic scale. The dots (×, ○) in the plot represent an experiment with the increasing number of clients.

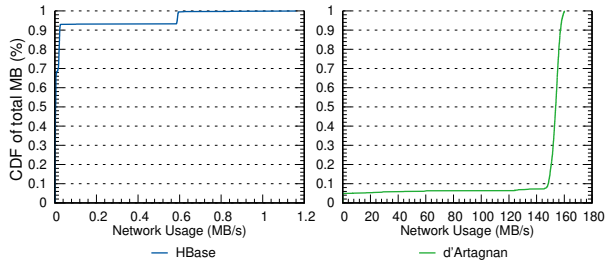

Fig. 4. Baseline (HBase) and d'Artagnan cumulative distribution function (CDF) of the network usage on workload G with a single client.

d'Artagnan prototype is limited to 5 OP/s and reaches an average latency of 78 seconds with 32 clients. HBase scales with the increasing number of clients and peaks at 33 OP/s with an average latency of 10 seconds with 32 clients. With the highest throughput of both systems, d'Artagnan prototype is 6.6× slower than HBase. Even with the specialized SMPC protocols, d'Artagnan prototype main bottleneck is the network bandwidth used to evaluate the multiplication gates of the encrypted circuits, as depicted with the cumulative distribution function (CDF) in Figure 4. The presented CDFs' results were collected during the execution of Workload G with a single client on both systems. A single database client is sufficient to saturate the network bandwidth with just 10% of the d'Artagnan network usage bellow 140 MB/s. Computational resources are not a critical factor as the CPU usage in every experiment, even when evaluating protocols with 256 concurrent clients, never rises above 30%.

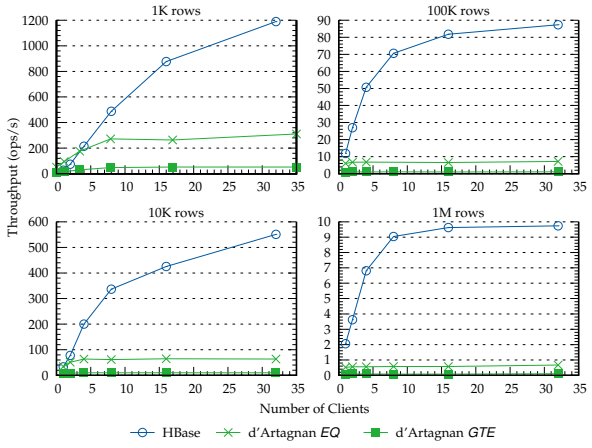

Fig. 5. d'Artagnan and baseline (HBase) performance on YCSB workloads with 100% Filters. These plots present both systems throughput with an increasing data set and clients (1,2,4,8,16,32).

Multi-party protocols. We also evaluated the SMPC protocols throughput isolated from any other operations. In particular, we evaluate two encrypted circuits: *Equality* (EQ) and *GreaterThanOrEqualTo* (GTE). These circuits are the backbone of the protocols implemented on the prototype as they enable the database to answer the NoSQL API defined in Section II. The evaluation measured the protocols performance with different Appointments table size, ranging from 1000 records to 1M. Furthermore, it assesses the protocols scalability for each dataset with an increasing number of clients, from 1 to 32 in a logarithmic scale base 2. The number of clients was restricted to 32 as d'Artagnan prototype reaches a saturation point. The table and request value distributions followed the same approach as the previous evaluation. The baseline was HBase's Equal filter.

Figure 5 shows the evaluation results with throughput versus number of clients curves. Overall, both systems throughputs decrease as data size increases, but only the baseline scales with the number clients. For the smaller data set, 1K rows, d'Artagnan prototype EQ protocol peaks at 310 OP/s. In contrast, the baseline has a maximum of 2431 OP/s for the same filter operation. Still on the smaller datasets, the GTE protocol reaches a maximum throughput of 52 OP/s for 32 clients. On the larger datasets, 100 K and 1 M, d'Artagnan prototype has a consistent overhead of 99% compared to the baseline as the system cannot scale with the increasing number of clients. Similar to Workload G, the main bottleneck is the network. On the smallest datasets the network usages ranges on average from 4 MB/s to 90 MB/s as the number of clients increase. After 10 K the bandwidth becomes saturated with just a few clients.

C. Multi-cloud deployment

A multi-cloud deployment requires a careful analysis of the cloud providers' location and the interconnecting network. The most important aspects are the distance between the third-

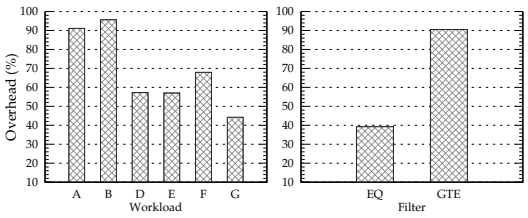

Fig. 6. d'Artagnan overhead against baseline (HBase) for the YCSB workloads and filters on a multi-cloud deployment for 32 clients and an Appointments table with 100 K rows.

party infrastructures and their distance to the `Safe Client`. Ideally, the client machine should be a private infrastructure located in the same city as the cloud servers to minimize the requests latency. However, this is not often possible either because the private infrastructure is far from any of the cloud provider servers, as is our case, or because the cloud providers do not have datacenters at the client's geographical area. As such, we present a scenario that illustrates a realistic use-case where deployment on a single city is not available but the `Safe Client` is in a private infrastructure near different cloud providers.

Experimental Set-up. The entire deployment consisted of 4 nodes, three independent HBase servers hosted on Google Cloud, Microsoft Azure and Digital Ocean, and the YCSB benchmark client hosted on Amazon AWS. The nodes were spread out through European countries and were selected to minimize the latency and maximize the available bandwidth. Google's HBase server was located on Frankfurt, Azure's was on Holland and Digital Ocean's was on Belgium. The client machine was also located on Frankfurt. The latency between nodes ranged from 1 ms to 12 ms and the bandwidth from 1 Gbps to 3 Gbps. d'Artagnan's servers were hosted on machines with 4 vCPUs, an SSD Disk and at least 10 GB of main memory. The client machine had 1 vCPU allocated, an SSD Disk and 1 GB of main memory.

This scenario follows a similar approach to the controlled environment but adjusts the appointments table size to 100 K rows. This adjustment was made to simulate a realistic use case where critical data is stored on an untrusted cloud [33]. As the system's scalability is presented in the controlled environment, this evaluation only considers the YCSB workloads and SMPC protocols for 32 clients. Experiments consists of 3 hour runs.

Figure 6 presents d'Artagnan prototype results as the overhead percentage in relation to the baseline, an HBase server on Microsoft Azure. All YCSB workloads follow the same distribution as in the controlled environment. d'Artagnan prototype throughput on workload A and B has a maximum overhead of 95%, peaks at 797 OP/s on workload A while the baseline reaches the 10 KOP/s on workload B. On the workloads D, E and F the overhead is slightly smaller and never surpasses the 43%. On Workload G the prototype peaks at 21.48 OP/s and the baseline at 38.53 OP/s, an overhead of 44%. The EQ protocol has the lowest overhead of 39% in contrast to HBase.

Overall, both experimental settings show that the d'Artagnan main source of overhead are the SMPC protocols. Even though these protocols are among the most efficient in the state-of-the-art the network bandwidth used to evaluate the multiplication gates decreases the overall system's throughput. However, the system's performance is acceptable for privacy-sensitive application without real-time performance requirements. In a realistic deployment with cloud providers, the system has an overhead as low as 39%. Furthermore, the current performance is not a hard-limit as novel, ground-breaking SMPC protocols have broken the 1 Billion gates per second barrier [34] as well as achieved global-scale secure computation [35]. The proposed framework can support additional protocols to tailor the performance for specific application requirements.

VIII. Related Work

Privacy-aware databases. Secure database systems make up a wide spectrum of solutions that provide trade-offs between privacy, efficiency, query capabilities and scalability. CryptDB [8] was the first system to support ciphertext processing of complex SQL queries with a modest overhead by integrating distinct cryptographic techniques in a single onion-layer storage model. This system protects values under increasingly stronger layers of encryption that limit the database engine query capabilities. The most outer layer leaks no information about the plaintext data, however, inner layers leak the order between values to enable the process queries. Monomi [36] expands CryptDB's query execution capabilities with new cryptographic techniques and a novel query planner that delegates most computation to the backed server. Unlike these systems, d'Artagnan is not limited to encryption schemes that leak some information to process queries. The system can process any complex query without compromising the user's confidentiality with SMPC protocols [11]. Furthermore, while a single successful attack on a privacy-aware database could compromise the system, such attack has little impact on a d'Artagnan system.

Multi-Party Systems. Multi-party protocols are mostly used to process special functions over data stored in independent database systems [18]. However, the protocols are not part of the databases and are implemented as a separate system. Wong *et al.* [37] proposed the first SQL database that relied on secure *two-party* computation protocols. This system is limited to two-party system where one of the parties is the database backend and the other is the database client. This approach requires the client application to do more processing than d'Artagnan and shares a similar problem to privacy-aware databases. A single successful attack on the cloud provider hosting the database server is sufficient to compromise the system. Furthermore, as a two-party system that leverages secret sharing it requires the client to store part of the secrets necessary for reconstructing the original values.

IX. Conclusion

This work presents d'Artagnan, a novel NoSQL database framework that deals with the inherent challenges of managing

multiple clouds, each with an independent database, to create a single logical secure database. Furthermore, the framework securely processes queries with SMPC protocols. A prototype is implemented from the architecture and validated with an industry standard-benchmark on a realistic deployment in the market-leading cloud providers. To achieve this, d'Artagnan can be configured with multiple privacy-preserving techniques, which include SMPC protocols. d'Artagnan's prototype was evaluated with state-of-the-art benchmarks and deployed in market-leading cloud providers. The results show d'Artagnan's practicality and effectiveness. Additionally, d'Artagnan's shown overhead in performance is not an hard-limit as novel research continues to improve SMPC protocols' performance and these can be easily integrated to the system.

ACKNOWLEDGMENTS

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia within project: UID/EEA/50014/2019. This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia with the grant: SFRH/BD/142704/2018.

REFERENCES

[1] E. Bertino and R. S. Sandhu, "Database security - concepts, approaches, and challenges," *IEEE Transactions on Dependable and Secure Computing*, pp. 2–19, 2005.

[2] E. Saleh, A. Alsa'deh, A. Kayed, and C. Meinel, "Processing over encrypted data: Between theory and practice," *SIGMOD Rec.*, pp. 5–16, 2016.

[3] J. Walker, "Microsoft's cloud is attacked 1.5 million times every day." [Online]. Available: http://www.digitaljournal.com/tech-and-science/technology/microsoft-s-cloud-is-attacked-1-5-million-times-every-day/article/494602

[4] I. . P. Institutre, "2018 cost of a data breach study: Global overview." [Online]. Available: https://www.ibm.com/security/data-breach

[5] S. Smith, "Cybercrime will cost businesses over 2 trillion by 2019." [Online]. Available: https://www.juniperresearch.com/press/press-releases/cybercrime-cost-businesses-over-2trillion

[6] F. B. Durak, T. M. DuBuisson, and D. Cash, "What else is revealed by order-revealing encryption?" Cryptology ePrint Archive, Report 2016/786.

[7] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *Proceedings of the 28th Annual International Conference on Advances in Cryptology: The Theory and Applications of Cryptographic Techniques*, 2009, pp. 224–241.

[8] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: Protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011, pp. 85–100.

[9] Microsoft, "Always encrypted (database engine)." [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine?view=sql-server-2017

[10] V. Bindschaedler, P. Grubbs, D. Cash, T. Ristenpart, and V. Shmatikov, "The tao of inference in privacy-protected databases," *PVLDB*, vol. 11, no. 11, pp. 1715–1728, 2018. [Online]. Available: http://www.vldb.org/pvldb/vol11/p1715-bindschaedler.pdf

[11] O. Goldreich, *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[12] R. Cattell, "Scalable SQL and NoSQL Data Stores," *SIGMOD Rec.*, vol. 39, no. 4, pp. 12–27, May 2011. [Online]. Available: http://doi.acm.org/10.1145/1978915.1978919

[13] J. Melton, *Database Language SQL*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 103–128. [Online]. Available: https://doi.org/10.1007/978-3-662-03526-9_5

[14] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, p. 4, 2008.

[15] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security*, ser. ESORICS '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 192–206.

[16] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. ACM, 1987, pp. 218–229.

[17] A. Shamir, "How to share a secret," *Commun. ACM*, pp. 612–613, 1979.

[18] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson, "High-performance secure multi-party computation for data mining applications," *Int. J. Inf. Secur.*, pp. 403–418, 2012.

[19] R. Cohen and Y. Lindell, "Fairness versus guaranteed output delivery in secure multiparty computation," *J. Cryptology*, vol. 30, no. 4, pp. 1157–1186, 2017.

[20] Y. Lindell and B. Pinkas, "Secure multiparty computation for privacy-preserving data mining," *IACR Cryptology ePrint Archive*, vol. 2008, p. 197, 2008. [Online]. Available: http://eprint.iacr.org/2008/197

[21] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman, "The matter of heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, ser. IMC '14. New York, NY, USA: ACM, 2014, pp. 475–488.

[22] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, "Eliminating the hypervisor attack surface for a more secure cloud," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 401–412.

[23] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[24] M. Keller and P. Scholl, "Efficient, oblivious data structures for mpc," in *Advances in Cryptology – ASIACRYPT 2014*, P. Sarkar and T. Iwata, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 506–525.

[25] HBase, "HBase." [Online]. Available: https://hbase.apache.org

[26] Redis, "Redis." [Online]. Available: https://redis.io

[27] H. Shacham and A. Boldyreva, Eds., *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, ser. Lecture Notes in Computer Science, vol. 10992. Springer, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-96881-0

[28] Google, "Google cloud plaform." [Online]. Available: https://cloud.google.com

[29] Microsoft, "Microsoft azure." [Online]. Available: https://azure.microsoft.com

[30] D. Ocean, "Digital ocean." [Online]. Available: https://www.digitalocean.com

[31] Amazon, "Aws." [Online]. Available: https://aws.amazon.com

[32] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, 2010, pp. 143–154.

[33] R. Krahn, B. Trach, A. Vahldiek-Oberwagner, T. Knauth, P. Bhatotia, and C. Fetzer, "Pesos: policy enhanced secure object store," in *EuroSys*. ACM, 2018, pp. 25:1–25:17.

[34] T. Araki, A. Barak, J. Furukawa, T. Lichter, Y. Lindell, A. Nof, K. Ohara, A. Watzman, and O. Weinstein, "Optimized honest-majority mpc for malicious adversaries — breaking the 1 billion-gate per second barrier," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 843–862.

[35] X. Wang, S. Ranellucci, and J. Katz, "Global-scale secure multiparty computation," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: ACM, 2017, pp. 39–56.

[36] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," *Proc. VLDB Endow.*, 2013.

[37] W. K. Wong, B. Kao, D. W. L. Cheung, R. Li, and S. M. Yiu, "Secure query processing with data interoperability in a cloud database environment," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 2014, pp. 1395–1406.