2016 IEEE 35th Symposium on Reliable Distributed Systems Workshops

SafeRegions: Performance evaluation of multi-party protocols on HBase

Rogério Pontes, Francisco Maia, João Paulo, Ricardo Vilaça
HASLab, INESC TEC & University of Minho

***Abstract*—On-line applications and services are now a critical part of our everyday life. Using these services typically requires us to trust our personal or company's information to a large number of third-party entities. These entities enforce several security measures to avoid unauthorized accesses but data is still stored on common database systems that are designed without data privacy concerns in mind. As a result, data is vulnerable against anyone with direct access to the database, which may be external attackers, malicious insiders, spies or even subpoenas.**

Building strong data privacy mechanisms on top of common database systems is possible but has a significant impact on the system's resources, computational capabilities and performance. Notably, the amount of useful computation that may be done over strongly encrypted data is close to none, which defeats the purpose of offloading computation to third-party services.

In this paper, we propose to shift the need to trust in the honesty and security of service providers to simply trust that they will not collude. This is reasonable as cloud providers, being competitors, do not share data among themselves. We focus on NoSQL databases and present SafeRegions, a novel prototype of a distributed and secure NoSQL database that is built on top of HBase and that guarantees strong data privacy while still providing most of HBase's query capabilities. SafeRegions relies on secret sharing and multi-party computation techniques to provide a NoSQL database built on top of multiple, non-colluding service providers that appear as a single one to the user. Strikingly, service providers, individually, cannot disclose any of the user's data but, together, are able to offer data storage and processing capabilities. Additionally, we evaluate SafeRegions exposing performance trade-offs imposed by security mechanisms and provide useful insights for future research on performance optimization.

***Keywords*-HBase, Secure databases, Multi party computation**

I. Introduction

Convenience and significant economical savings are spurring many enterprises and end users to move their data and applications to third-party cloud services. This naturally implies trusting that these services employ the necessary mechanisms to ensure that their data is kept private. However, such trust is misplaced. In fact, recent studies have shown that a large amount of the data stored in third-party infrastructures is unprotected against the prying eye of a system administrator, a government subpoena or even an external attacker [1]. Notably, leveraging cloud database storage and computational capabilities while ensuring data privacy is still an open research challenge.

An immediate approach to achieve stronger data privacy guarantees is to strongly encrypt the data before uploading it to third party infrastructures. However, this solution raises several issues. For instance, if a traditional symmetric cypher is applied to every database entry many of the query capabilities of that database become unusable since data properties, such as order, are lost with the encryption scheme. As a result, the database becomes nothing more than a storage service and computation over the data must be made at the user side where data can be decrypted and processed. Needless to say that this defeats the purpose of using the storage and processing capabilities of a third-party infrastructure. This limitation can be partially solved by privacy-aware databases such as CryptDB that leverages multiple encryption schemes and rewrites database queries at the client side so that useful computation can be done over protected data [2]. However, this approach has been shown to still leak a significant amount of sensitive information [3].

This paper explores an alternative solution for providing a privacy-aware database. The core idea behind this solution is to divide data management and processing amongst multiple service providers with independent infrastructures. By doing so, no provider is able to access the original data or extract any kind of useful knowledge from the protected data, while the client still has a fully-functional NoSQL databases. Rooted on this idea, we propose a solution that relies on secret sharing to divide sensitive data into a set of secrets that are stored across several domains that are not expected to collude. Each secret reveals nothing about the data and can be seen as a piece of a puzzle. With a single piece the puzzle cannot be solved but, after every single piece is put into the right place the puzzle is complete and the original data is recovered. Additionally, the proposed solution relies on secure multi-party protocols (MPC) to leverage distributed computation on top of the secrets without leaking any information. To sum up, by combining secret sharing and multi-party protocols our proposal achieves both private data storage and private processing.

The main contribution of the paper is SafeRegions, a system built on top of the HBase NoSQL database that resorts to secret sharing and MPC. SafeRegions leverages the concept of a virtual cloud database composed by multiple independent untrusted cloud infrastructures, a concept that was previously discussed for secure storage systems [4, 5]. In more detail, our prototype resorts to different HBase clusters deployed on independent infrastructures and on a proxy that abstracts clients from this distributed setup, thus presenting SafeRegions to clients' applications as a regular HBase-like NoSQL database.

The paper is structured as follows: an introduction to secret sharing, multi-party protocols and HBase is presented in Section II. Section III presents SafeRegions' architecture and discusses its components. Section V presents the results and an analysis of SafeRegions prototype's evaluation. Section VI discusses related work while Section VII concludes the paper.

II. Building Blocks

SafeRegions combines secret sharing and MPC protocols with HBase to provide a privacy-aware NoSQL database solution. Next we discuss each of these building blocks in more detail.

A. Secret Sharing and MPC

Secret sharing schemes consider two types of entities, a *dealer* D and a set of *players* $P = \{p_1, \ldots, p_n\}$. The purpose of a dealer is to transform a value v in a set of n *secrets* and store on each player a single secret. Each secret by itself does not leak any information about v which conforms to our privacy needs. Value v can be reconstructed only when all secrets are grouped together [6].

The purpose of secret sharing is to ensure the privacy of stored data. This way, this technique is not intended to provide processing capabilities over the secrets. Secure multi-party protocols solve this limitation by calculating an arbitrary function over a set of private inputs, such as the generated secrets, without leaking any sensitive information. While many protocols have been proposed, there are few practical implementations, which has lead us to focus on the Sharemind protocols [7].

In Sharemind, values are protected with additive secret sharing over a finite field $\mathbb{Z}_{2^n}$. Additive secret sharing enables arithmetic calculations to be performed over the finite field which is crucial to compare values through secure protocols. Namely, two protocols are proposed, an *Equality* protocol and a *GreaterThan* protocol. The protocols are bound to three parties. With a lower number of parties, privacy cannot be ensured while an higher number would only increase the overhead of computation.

The Sharemind protocols follow the following scheme. There is one Dealer that, for each input value, generates three secrets (1 in Figure 1). Each secret is stored in one of three players (2 in Figure 2). To give an overview of a protocol execution (depicted in Figure 2), let us consider that the Dealer wants to search for a certain value v is stored in the system. In order to do so, the Dealer generates new secrets from v and sends one of them to each player requesting a comparison protocol execution (1 and 2 in Figure 2). Each player, in order to execute the protocol, follows a set of secret generation and secret exchanging steps with other players (3 in Figure 2). These computation and message exchanging steps can be repeated. Next, each player returns its computation result to the Dealer (4 in Figure 2). Finally, the Dealer can extract the computation result from the combination of the three secrets (5 in Figure 2). In this case the result is one if the value v was found in the system and zero otherwise.

Trust Model. The previous protocols are proven to be secure under the passive (honest-but-curious) model. This model assumes an attacker capable of gaining access to the infrastructure of at most a single party, before the execution of any protocol. The attacker can see all the secrets held by the party, as well as all the messages exchanged by such party. However, the attacker is not capable of interfering in any way on the output of the computation nor see any of the messages exchanged between other entities [7]. Furthermore the protocols are proven to be universally composable, meaning that every protocol on the framework can be composed and remain secure.

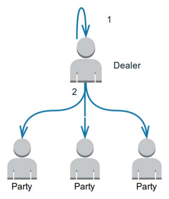

Figure 1. Store operation.

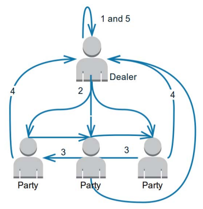

Figure 2. Search operation.

B. HBase

HBase is an open-source, widely used NoSQL data store that offers high efficiency and scalability [8]. This data store is heavily inspired by Google BigTable [9] and offers a powerful and scalable data model. Similarly to relational databases, information is stored on tables. However an HBase table, contains two levels of columns: column families and column qualifiers. Column families are defined on table declaration and group multiple column qualifiers. As such the column qualifiers are always associated to a column family and are only defined on value insertion. In the case a column family does not contain a column qualifier, the new qualifier is dynamical added. Each row on a table has an unique identifier and the cells of the table can contain empty values. Table I contains a model of an HBase table with one column family, Name, and two column qualifiers (First and Last Name).

As tables grow in size, HBase can scale dynamically by partitioning a table horizontally, creating multiple regions. Each region of a table holds a subset of the rows and is stored on a single RegionServer. These servers do most of the computation in HBase and are the backbone of HBase scalability. The reminder of the computation is performed by the HBase master. This entity manages the cluster in a master/slave architecture and is the entry point of the system. Every client operation must first interact with the

Identifier	Name	
	First Name	Last Name
1412	John	Alistar
2231	James	Smith
6262	Jane	

Table I
EXAMPLE OF AN HBASE TABLE

Master that redirects the clients to a RegionServer that can handle the request.

The HBase client API is composed by *PUT*, *GET*, *DELETE* and *SCAN* operations. The PUT and DELETE are the main operators used to insert, update or delete a record. A simple PUT operation requires declaring a row identifier, a column family and a column qualifier to be inserted/updated. On the other hand, a DELETE only requires specifying a row id in order to remove the corresponding row. By default the GET operator returns every column value from a single row of a table. However, all these operations can be enhanced to only specify certain column families or/and column qualifiers to be inserted, deleted or retrieved. Finally the SCAN operator returns the set of rows whose id is between a minimum and maximum value. Similarly, a SCAN can return all rows and corresponding columns or a filter may be used to discard unwanted records. For instance it is possible to define a SCAN operation that only returns the rows from Table I where the value of column *First Name* is "John".

The implementation/behavior of the previous operations can be modified/extended without breaking the API or changing the database source code due to the concept of Coprocessors. Coprocessors enable developers to load custom code for each operation that is executed by the database. While there are several types of Coprocessors, this paper focuses on the Endpoint Coprocessors. These coprocessors allow adding new operations that can be used by an HBase client application as a regular operation and are essential to provide the MPC protocols on SafeRegions in a generic and transparent manner.

The vanilla HBase system does not provide any mechanisms to protect stored data's privacy. In the next sections we present SafeRegions, a novel solution to ensure private storage and computation on top of HBase.

III. SYSTEM DESIGN

The main idea of *SafeRegions* is to use additive secret sharing to encrypt users' data and then to leverage the Equality and GreaterThan MPC protocols to provide secure HBase operations. These two protocols are essential since most computation done in HBase requires equality or order comparison of values. However, there are two major challenges that must be addressed to achieve a privacy-aware HBase MPC solution. Firstly, the previous protocols require three independent computation parties, which in our case are three HBase clusters. Secondly, we want to still offer a unified interface to clients, which implies enhancing HBase client to abstract this distributed deployment and the novel security mechanisms while providing the same API as the vanilla HBase client. In Figure 3 we present an high level overview of the SafeRegions architecture. Along this section we briefly describe its three main components: client, computation parties and communication middleware.

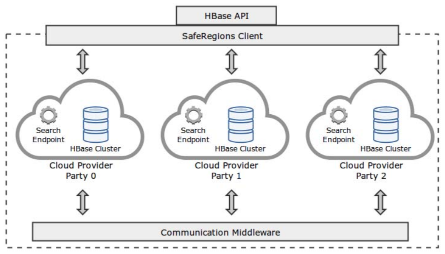

Figure 3. SafeRegions Architecture

A. SafeRegions Client

The client component is the entry point to the SafeRegions system and provides the same API as the vanilla HBase client. This component abstracts all the complexity of protecting users' data (secrets generation and decoding) and the communication with the three HBase clusters.

The client is also responsible for orchestrating the interaction with the different computation parties to process user's requests. It is the client component that transforms user's data into protected data to be stored and queried in the same way as the Dealer in Figure 2.

In order to query protected data, the client queries each one of the HBase clusters that then perform the necessary MPC protocols on the locally stored secrets. In more detail, this request can be made by issuing parallel remote-procedure calls (RPC) calls to each HBase SearchEnpoint *coprocessor*. Each HBase cluster processes the request resorting to its local storage, and returns as a response a vector with the resulting secrets of performing the desired MPC protocolos. With the three vectors, the client can merge the correct secrets and rebuild the information contained in the query response. Lets take as example a *GET* operation over protected row identifiers. The secure client must encode, with secret sharing, the identifier value being requested by the user and issue requests to the HBase clusters using these secret shared identifiers. Resorting to the *Search* operator, each cluster checks for a corresponding row in their local storage. If found, the cluster will reply to the client component with the corresponding row. Finally, the client component is responsible for decoding the answers and replying back to the user with the actual queried data.

B. Computational Parties

In order to support the MPC protocols discussed in Section II-A, we consider three computational parties in our architecture that correspond to the players in Figure 2. In SafeRegions these components are three HBase clusters. Each cluster is responsible for one of the three secrets generated per data piece. Naturally, this implies that these clusters are structurally similar. They share the same table structure and will hold the same amount of data.

However, the clusters are necessarily deployed in independent infrastructures and can be uniquely identified by a number *ID*. This identification is important for the client to store and collect secrets from the distinct clusters. For instance, some data A is transformed into three secrets A_1, A_2 and A_3 to be stored in cluster 1, 2 and 3. Similarly, if A_1 is stored in $TableOne$ at cluster 1, then A_2 is stored in an identical table at cluster 2 and so forth.

In addition to maintaining an identical structure across HBase clusters, MPC protocols requires communication between the clusters. This is achieved with a *coprocessor* endpoint, which we call *SearchEndpoint*. The *SearchEndpoint* does not require any modification to the architecture neither to the implementation of the HBase core mechanisms. Nevertheless, the *SearchEndpoint* plays a central role in the architecture. Each RegionServer must contain this endpoint in order to expose the secure MPC protocols as a RPC to be used by the client. Additionally, the *SearchEndpoint* relies on a communication middleware for exchanging secrets across parties (clusters), that is further described next.

Upon the reception of a query request, the endpoint starts a MPC protocol for every record stored in that region server. The actual implementation of the protocol is contained in a MPC library that performs all the necessary computation (secret generation) and uses uses the communication middleware to exchange messages across parties (clusters). The validity of secrets computed with the MPC library is verified at the endpoint that is also responsible for replying back to the client with the query response.

C. Communication Middleware

By default HBase does not support communication across RegionServers. However, SafeRegions requires communication across distinct HBase clusters. To achieve this, we introduce a communication middleware between the RegionServers to handle the exchange of messages across parties. In order to support MPC protocols, the communication middleware offers two essential primitives. A $send(secret, party)$ primitive that delivers a secret to a target party in a non-blocking fashion and a $receive(party)$ primitive that waits for incoming messages. Moreover, the communication middleware ensures that the messages sent from one party to another arrive in order and that it is possible to always know the source party that sent the message. These two properties are needed for correctly supporting MPC protocols.

IV. Implementation

The system architecture proposed in the previous section leaves open implementation decisions that impact the inner working of the SafeRegions system. Starting with maintaining different HBase structurally similar, there is an immediate implementation detail that must be addressed, which is using secrets as row identifiers. In detail, if secrets are used to protect HBase row identifiers, then these cannot be mapped directly to the protected HBase schema as identifiers because secrets generated by the MPC library do not have deterministic content (due to MPC randomness). This means that it becomes impossible to match identical rows in different HBase clusters and to update, retrieve or delete a specific row in SafeRegions. To solve both problems, an extra column is added to the HBase table for storing the protected identifiers (secrets). Then, the row identifiers of each HBase table are replaced with virtual identifiers managed by the SafeRegions client. These identify uniquely each row and are identical across different clusters. This virtual identifier does not leak any sensitive information and allows matching rows and the corresponding secrets across different HBase clusters.

For each MPC protocol execution request, our current coprocessor implementation performs a simple sequential iteration over every record stored in the corresponding RegionServer and uses our implementation of the Sharemind protocols to execute the necessary computation. When this computation is finished, two of the RegionServers send the generated secrets to a third RegionServer. This third RegionServer combines the resulting secrets from every record and discovers what are the HBase records that match the NoSQL query being executed. This information is then sent to the SafeRegions client that retrieves from each HBase cluster the needed records (protected as secrets) and combines them to decode the original values. This last step does not compromise the privacy of stored values and is an optimization of the standard Sharemind protocol in order to lower the computation and number of messages/data received at the SafeRegions client.

Finally, Sharemind protocols were implemented in Java since there is not a freely available implementation. The communication middleware is implemented using Java NIO.

V. Evaluation

In order to evaluate the performance of adding privacy to common HBase operations such as, creating a table, inserting records, and retrieving records, we performed a set of experiments. The experiments ran on a cluster of servers equipped with an Intel i3 CPU with four cores at 3.7 GHz, 8 GB of RAM and a 128GB SSD. Each machine ran Ubuntu 14.04 and the SafeRegion Cluster was built using HBase 0.98 in standalone mode. The results obtained consist of one run of half an hour.

Finally, in the experiments we chose to preserve the privacy of HBase identifiers with secret sharing. This decision was taken because the tested operations perform computation over the identifiers and not across the column values. This way, to understand the overhead of our solution identifiers must be protected while values can be left in clear text.

We started by evaluating the latency for creating a

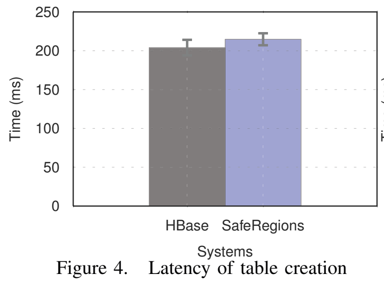

Figure 4. Latency of table creation

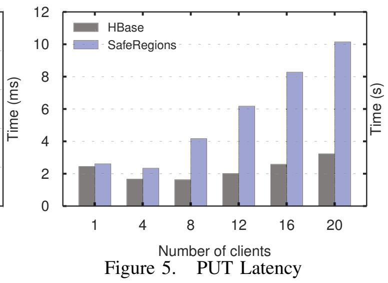

Figure 5. PUT Latency

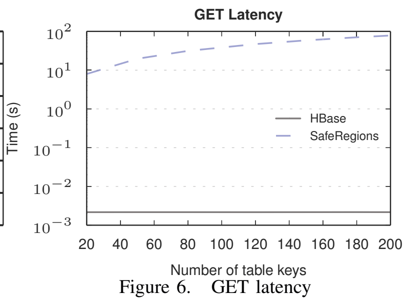

Figure 6. GET latency

table on vanilla HBase and on the SafeRegions system. SafeRegions system introduces an overhead of 4.9% in comparison to the vanilla HBase as depicted in Figure 4. As expected, this overhead steams from adding a column family in the table create statement and from issuing three concurrent requests to each HBase cluster.

We then proceeded to evaluate the overhead of PUT operations on both systems. In these experiments the client is only inserting new keys. Since updates are not being executed, no MPC protocol has to be performed. This way, we are evaluating the overhead of creating three secrets, and issuing three parallel PUT operations, one for each cluster. The evaluation benchmark simulated multiple clients by resorting to independent threads.

As can be seen in Figure 5 with a single client, the latency overhead of PUT operations is 7%, however as the number of clients increases the latency overhead also increases, being the highest overhead value approximately 220%. This significant increase on latency comes from spawning one thread per simulated client, each issuing three additional concurrent PUT requests (1 per cluster). In fact when there are n clients, there are $n * 3$ PUT operations to be made and $n * 3$ threads. For instance, when there are 20 clients, our benchmark has 60 PUT operations and threads being executed concurrently. On the other hand, the generation of random numbers by the clients has small impact in performance. In detail, we use the Uncommons maths library [10] that takes on average 0.016 ms to generate a random number.

Finally, we have evaluated the overhead incurred by executing a MPC equal protocol. To evaluate this cost, we ran a benchmark that pre-populated multiple rows on the system and then performed several GET operations. Each GET requires the equal protocol to be performed, as described in section IV, in order to retrieve the correct row identifier. From Figure 6, with a logarithmic scale, it is possible to verify that the MPC equal protocol incurs a significant overhead, while the default HBase can perform an operation with a latency of milliseconds, the SafeRegions operation is in the orders of seconds. While the latency of the vanilla HBase has a constant 2 ms latency with the increase of rows, the SafeRegion solutions sees an increase on latency. For two hundred keys the latency reaches the 78 seconds on average. This significant increase is due to the equal protocol that exchanges multiple messages between the parties, for instance, with 8 byte keys it requires 458 messages per comparison. Furthermore the MPC protocols requires a comparison with every record on the table, which is also a costly operation.

VI. Related Work

Bringing privacy guarantees to the database-as-a-service model (DBaaS) is a field still in expansion with multiple paths being pursued. In 2002, NetDB2's challenged the database community to explore several open challenges in databases [11]. One of the issues discussed was data privacy, which was tackled with symmetric or asymmetric encryption at the client side to protect users' data before being stored in a remote database.

The previous proposal requires queries computation to be placed on the client side, which defeats the purpose of leveraging the cloud's computational resources. In order to shift some of the computation away from the client, multiple systems proposed to use a trusted third-party service to handle the communication and computation between the client and the database service provider [12, 13, 14]. These solutions still depend on trusting a third-party entity.

In CryptDB a proxy mediates the communication between the client and database while rewriting queries to leverage computation over protected data [2]. Data can be encrypted with different schemes, deterministic encryption, order preserving encryption or Homomorphic encryption, with each one enabling different types of queries and having different types of security guarantees. In the case of deterministic encryption it has been shown that cryptDB is susceptible to frequency analysis attacks [3]. Monomi builds on top of cryptDB and improves the performance of query processing by choosing in anticipation the most appropriate encryption scheme for each database column. Also query plans are used to decide which parts of the query are processed in the untrusted service and which are executed on the proxy/client [15].

Wai *et al.* takes a different approach to existing systems by removing the proxy and sharing the computation between the client and the server [16]. Not only is the architecture different but also the encryption schemes. This system is based on secure MPC while sensitive data is encrypted using secret sharing which encodes the information in two secrets. One secret is stored on the client while the other is stored at the server. With secret sharing and secure MPC SQL operators can be pipelined in a query, unlike in CryptDB where the operators that can be executed are bound by the encryption scheme.

The existing solutions focus mostly on SQL databases and are not concerned with distributed databases that may provide better performance and scalability. Our approach is similar to Wai *et al.* since it also applies MPC protocols. However, all queries are processed in the untrusted servers, thus requiring minimal computation at the client side. Furthermore we do not rely on the client to store meta-data or secrets *i.e.*, all data is stored on the HBase clusters.

VII. Conclusion

This paper introduces SafeRegions, a novel system that combines secret sharing and MPC to provide privacy-aware data storage and computation in NoSQL. Our prototype resorts to the widely used HBase NoSQL data store and shows that it is possible to provide the full HBase API for clients while ensuring that their data is stored in a completely private fashion. In fact, by spreading the data (secrets) into multiple HBase clusters, even if one of these clusters becomes compromised, there is not any leakage of sensitive information.

Like any other security mechanism, secret sharing and MPC introduce a performance penalty in SafeRegions. This is a necessary tradeoff, that is discussed in our experimental evaluation section, in order to have stronger security guarantees.

As future work, many design and implementation improvements are still possible. For instance, the secrets can be calculated in parallel and the implementation of the MPC protocols can be improved with batching or additional parallelization.

To conclude, privacy in NoSQL comes always associated with a performance/functionality cost. However, we predict that in the near future, novel solutions and optimizations will be proposed to tackle the information privacy challenge that is now a global concern.

VIII. Acknowledegements

The research leading to these results has received funding from different sources. Rogério Pontes is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme within project (POCI-01-0145-FEDER-006961), and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) as part of project UID/EEA/50014/2013. The remainder authors received funding from the European Union's Horizon 2020 - The EU Framework Programme for Research and Innovation 2014-2020, under grant agreement No. 653884.

References

[1] T. B. Pedersen, Y. Saygın, and E. Savaş, "Secret sharing vs. encryption-based techniques for privacy preserving data mining," 2007.

[2] 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*, ser. SOSP '11, 2011, pp. 85–100.

[3] I. H. Akin and B. Sunar, "On the difficulty of securing web applications using cryptdb," in *Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on*, Dec 2014, pp. 745–752.

[4] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "Depsky: Dependable and secure storage in a cloud-of-clouds," in *Proceedings of the Sixth Conference on Computer Systems (EuroSys)*, 2011.

[5] H. Tang, F. Liu, G. Shen, Y. Jin, and C. Guo, "UniDrive: Synergize Multiple Consumer Cloud Storage Services," in *Proceedings of the 16th Annual Middleware Conference (Middleware)*, 2015.

[6] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[7] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson, "High-performance secure multi-party computation for data mining applications," *International Journal of Information Security*, pp. 403–418, 2012.

[8] "Hbase." [Online]. Available: https://hbase.apache.org/

[9] 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 Trans. Comput. Syst.*, vol. 26, no. 2, pp. 4:1–4:26, Jun. 2008.

[10] "Uncommons maths." [Online]. Available: http://maths.uncommons.org/

[11] "Providing database as a service," in *Proceedings of the 18th International Conference on Data Engineering*, ser. ICDE '02, 2002, pp. 29–.

[12] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring data storage security in cloud computing," in *Quality of Service, 2009. IWQoS. 17th International Workshop on*, July 2009, pp. 1–9.

[13] N. Jefferies, C. Mitchell, and M. Walker, *Cryptography: Policy and Algorithms: International Conference Brisbane, Queensland, Australia, July 3–5, 1995 Proceedings*, 1996, ch. A proposed architecture for trusted third party services, pp. 98–104.

[14] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, N. Mishra, R. Motwani, U. Srivastava, D. Thomas, J. Widom, and Y. Xu, "Vision paper: Enabling privacy for the paranoids," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, ser. VLDB '04, 2004, pp. 708–719.

[15] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," *Proc. VLDB Endow.*, vol. 6, no. 5, Mar. 2013.

[16] 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*, ser. SIGMOD '14, 2014, pp. 1395–1406.