PIUDI: Private Information Update for Distributed Infrastructure

Shubham Raj, Snehil Joshi and Kannan Srinathan
Centre for Security, Theory and Algorithmic Research, International Institute of Information Technology, Hyderabad, India

Keywords: Privacy, Blockchain, Private Information Retrieval, Private Information Retrieval-Writing, Distributed Database, Packed Secret Sharing, Privacy Enhancing Technology.

Abstract: Encrypted data is susceptible to side-channel attacks like usage and access analysis. Techniques like Oblivious-RAM (ORAM) and privacy information retrieval and writing aim to hide clients' access pattern while accessing encrypted data on a distrusted server. However, current techniques are constructed for a single server model making them unsuitable and inefficient for contemporary distributed architectures. In our work, we address this problem and provide a solution to private information update using packed secret sharing. Our protocol, named "Private Information Update for Distributed Infrastructure" *PIUDI*, aims to mitigate the attacks to which *PIR-Writing* protocols are more susceptible in a distributed environment. Our scheme is secure in presence of up to $t+k-1$ compromised parties where k is the size of the data set. We also provide an analysis of our protocol for computational efficiency and gas cost in blockchains.

1 INTRODUCTION

Encryption is one of the primary measures used to safeguard sensitive data stored in databases(Popa et al., 2011; Popa et al., 2014; Papadimitriou et al., 2016). However, inference and log analysis attacks pose significant threats to the privacy and security of encrypted data(Grubbs et al., 2017; Lacharité et al., 2018). For example, an attacker could use traffic analysis to infer when and how often they parties communicate, and also distinguish between different types of encrypted data, such as emails, media files etc. Additionally, service providers can run analysis on the access patterns over a client's encrypted data to extract vital information.

Inference based methods involve attempting to extract sensitive information by observing patterns in the access and operations over encrypted data. These kind of attacks are highly dangerous and can often compromise the privacy of individuals and organizations. Due to efficiency concerns, a majority of current protocols unintentionally expose data access patterns(Papadimitriou et al., 2016). *Log analysis*,meanwhile, exploits database logs to gain unauthorized access to sensitive information and find correlations between encrypted columns via frequency attacks(Zolotukhin et al., 2014). The attackers use these correlations to get insights about sensitive data based on the type of data stored in the database or mapping the data to publicly available data-sets(Dwork et al., 2017). Thus attackers can often gain access to valuable information even without breaking the encryption.

In this paper, we address the challenges posed by these attacks. Our focus is majorly on mitigating *correlation and frequency attacks.* Correlation attacks involve finding correlations between different columns in the database. By analyzing the correlations, attackers can often infer sensitive information that they would not otherwise have access to. Frequency attacks, on the other hand, involve analyzing the frequency of particular data values. Attackers can often infer sensitive information by analyzing the frequency of different operations as well as the frequency of access queries of any type on data values, even if the data itself is encrypted.

In current cryptography literature, there are mainly two methods for concealing a client's access patterns: *Oblivious RAM (ORAM)* and *Private Information Retrieval (PIR)*. ORAM's traditional approach involves organizing data in a way that ensures that the client never accesses the same part twice, without an intermediary process that removes the correlation between block locations(Goldreich and Ostrovsky, 1996)(Stefanov et al., 2018). While ORAMs had low communication complexity and do not require any computation on the server, sometimes the client may have to download and reorganize the entire

Raj, S., Joshi, S. and Srinathan, K.
PIUDI: Private Information Update for Distributed Infrastructure.
DOI: 10.5220/0012087900003555
In *Proceedings of the 20th International Conference on Security and Cryptography (SECRYPT 2023)*, pages 425-432
ISBN: 978-989-758-666-8; ISSN: 2184-7711

database, which is not practical.(Islam et al., 2012). In contrast to ORAM, Private Information Retrieval *(PIR)* hides the specific query being made, regardless of any previous queries. *PIR* uses homomorphic encryption and does not hide a sequence of accesses, but instead each access individually. The downside is that the server needs to compute over the entire database for each query, which can be impractical for large databases.

In addition to all the above issues, current techniques have only been proposed for single server problems and distributed systems have generally been overlooked. As such, using them for a distributed database will require direct replication of the techniques at each instance of the database itself. This makes these techniques inefficient and difficult to scale, and therefore impractical, in the presence of a large number of instances. They also do not take into cognizance data sharding techniques. Moreover, data sharding is commonly used in large-scale applications that require the ability to store and process vast amounts of data in a distributed environment. Sharding is a technique that involves partitioning a large database into smaller, more manageable subsets called shards. Each shard contains a subset of the data and can be stored on separate servers. This technique can improve the scalability and performance of the database by allowing multiple servers to process data simultaneously. The current *PIR-Writing* techniques do not provide an efficient mechanism to handle the case for data sharding as well. Our work aims to solve these problems.

1.1 Relevant Work and Motivation

The problem of Private Information Retrieval (*PIR*) has been studied extensively in the field of cryptography and computer science. *PIR* protocols allow a client to retrieve data from a database without revealing which item was accessed. This is particularly useful when dealing with sensitive information that needs to be kept private against access pattern analysis. The first *PIR* protocol was proposed in the seminal work of (Chor et al., 1998), and since then, most of the research in this area has been done to develop more efficient and secure protocols(Gasarch, 2004).

PIR solved the problem of obliviously reading the data, as it ensured that the client's privacy was protected. However, the problem of *PIR-writing* was still relevant due to the statistical and inference-based attacks on access patterns and database and system logs. Such attacks revealed information about the client's queries, even if the specific data accessed remains private. This posed a significant challenge in the design of *PIR* systems, as ensuring both data privacy and query privacy is crucial to protect clients' sensitive information.

Boneh et al. proposed the first *PIR-Writing* protocol with sublinear communication complexity, which uses a bilinear-pairing based cryptosystem(Boneh et al., 2007). Lipmaa et al. (Lipmaa and Zhang, 2010) then came up two new *PIR-Writing* protocols. The first *PIR-Writing* protocol is based on the cryptocomputing protocol PrivateBDD of Ishai and Paskin(Ishai and Paskin, 2007). The second protocol is based on a fully-homomorphic cryptosystem. Both these approached use computational assumptions depending on the hardness of the underlying problem upon which the cryptosystem is based. Our protocol relies on perfectly secure schemes. Even though our *PIR-Writing* scheme is homomorphic, it does not use homomorphic encryption for the private computational operations which makes it quite beneficial when computing happens over expensive environments like public and permission-less blockchains.

1.2 Contributions

In our paper, we propose a novel protocol, "Private Information Update for Distributed Infrastructure" ($PIUDI$), that uses a combination of data sharding and secret sharing to mitigate aforementioned attacks as well as improve scalability for practical applications.

- Our protocol is incredibly communication efficient, utilising packed secret sharing to encode multiple data elements into a single polynomial.
- It is also computation efficient which in the context of blockchains, saves gas costs.
- It is also more scalable in a distributed setting : as the number of the database instance increases, our protocol makes sure that encoded data set increases by a constant rate.
- Our protocol also supports batch updates, which greatly simplifies the process of updating data in both blockchain networks and distributed databases and can lead to significant improvements in efficiency.

We also provide a detailed analysis of our proposed solution, including a formal security proof and a comparative analysis with existing protocols, thereby demonstrating the effectiveness of our protocol.

1.3 Organization of the Paper

The first section begins with an introduction, motivation of our work and the literature survey. It also outlines contributions of our work. The second section defines the the communication and adversarial models, cryptographic assumptions,and underlying schemes we have utilized in our work. This provides the reader with the necessary background knowledge and technical details for our protocol. The third section describes our proposed protocol in detail, and its variations. In the fourth section, we provide formal definitions of security and present a detailed proof of our protocol's security guarantees against different attacks. The next section presents a performance evaluation of our protocol and a comparative analysis against existing protocols. The next section describes potential use cases of our protocol in practical scenarios. Finally, the conclusion summarizes the contributions of our research and provides a discussion of the limitations of our study and suggestions for future research.

2 PRELIMINARIES

2.1 Communication and Adversary Model

In this paper, we will be examining the *stand-alone setting*, which is characterized by a *synchronous network* and *perfectly private channels* between all parties involved in the protocol. The stand-alone setting restricts our analysis to only a single protocol execution, as opposed to a repeated execution of the protocol with changing participants.

Furthermore, we assume *static corruptions* in which the set of corrupted parties is fixed ahead of time and remains constant throughout the execution of the protocol as well as *semi-honest adversaries*, i.e., those who follows the protocol correctly, but may attempt to learn information outside their purview without actually deviating from the protocol in any way. All parties also have a probabilistic polytime bound on their computational power.

2.2 Shamir Secret Sharing

Shamir Secret Sharing (SSS)(Shamir, 1979) is a cryptographic technique that allows a secret to be split into multiple shares and distributed among a group of participants in such a way that only a predetermined number of shares are required to reconstruct the original secret. This technique has found widespread use in a variety of applications, including secure communication, key management, and data storage.

2.3 Packed Secret Sharing

The Packed Shamir secret sharing scheme proposed in 1992(Franklin and Yung, 1992) is an extension of the original Shamir secret sharing scheme introduced by Shamir in 1979. This variant enables the sharing of a group of secrets using a single Shamir sharing, which is a more efficient and convenient approach. Specifically, if we have a vector x in a finite field F_k, then we can create a degree-d packed Shamir sharing denoted as $[x]_d$, where d is a value between $k-1$ and $n-1$. To reconstruct the original sharing, at least $d+1$ shares are required, and any $d-k+1$ shares are independent of the underlying secrets. The packed secret sharing has linear homomorphism as well has multiplicative properties.

Consider a field F and let $\alpha_1,...,\alpha_n$ be n distinct elements in F. Let pos = $(p_1,p_2,...,p_k)$ be another k distinct elements in F. Suppose we wish to share a vector x = $(x_1,...,x_k) \in F_k$ among k parties such that each party receives a share of the vector. We can achieve this by constructing a degree-d $(d \geq k-1)$ packed Shamir sharing of x, which is a vector $(w_1,...,w_n)$ satisfying the following conditions:

There exists a polynomial $f() \in F[X]$ of degree at most d such that $f(p_i) = x_i$ for all $i \in 1,2,...,k$. For all $i \in 1,2,...,n$, $f(\alpha_i) = w_i$, where the $i-th$ share w_i is held by party P_i. In other words, the polynomial $f()$ is used to encode the vector x and the shares $w_1,...,w_n$ are used to distribute the encoded vector among the parties. This allows each party to reconstruct their share of the vector using their share and the shares of other parties.

2.3.1 Packed Secret Sharing Protocol *(PSS)*

Let x be a vector that we want to share such that x = $(x_1,...,x_k) \in F$ such that the protocol can tolerate up to t adversaries. Let pos = $(i_1,...,i_k)$ be other field element such that pos $\in F$ and they are the index to encode the secret vector x.

1. Let dealer select a random polynomial $f_s()$ "of degree at most $d = k-1+t$
2. Encode $x_j \in x$ in the polynomial $f_s()$ as $f_s(i_j) = x_j$ $\forall j = 1,...,k$
3. Distribute $f_s(w_i)$ to n parties such that $w_i \in F$ $\forall i = 1,...,n$

Lemma 2.1. *Suppose we have a secret vector x and a random polynomial f_s, as defined previously. If we*

select a subset of shares, containing no more than t shares, then the distribution of those shares is unrelated to x. On the other hand, if we gather at least $k+t$ shares, we can use them to recover x.

Proof. To prove the first statement, let's consider the shares $f_s(1)$ through $f_s(t)$, without loss of generality. Using Lagrange interpolation, we can create a polynomial h with a maximum degree of $d = k-1+t$, such that $h(w_1)$ through $h(w_t)$ are all zero, and $h(i_j)$ = -x_j for $j = 1$ through k. This means that for each polynomial $f_s()$ that shares the secret vector x, there is exactly one polynomial $f_s(x)+h(x)$ that shares the all-zero vector and generates the same first t shares. Since the choice of polynomials is random and uniform, we can conclude that the distribution of the t shares is the same for all secret vectors, namely the distribution resulting from sharing the all-zero vector. The last statement on reconstruction is straightforward and follows from Lagrange interpolation. □

2.3.2 *(PSS)* Notations

Let x be the vector of secrets we want to share using polynomial f_a. Let y be the vector we want to securely add to vector x and we share y using polynomial f_b.

- ENCODE(x, f_a) : $[x, f_a]_d = (f_a(w_1), ..., f_a(w_n))$
- ENCODE(y, f_b) : $[y, f_b]_d = (f_b(w_1), ..., f_b(w_n))$
- ADD(x, y) : $[x, f_a]_d + [y, f_b]_d = [x+y, f_a+f_b]_d$

We can define the multiplicative properties of the share in the same way

- MUL(x, y) : $[x, f_a]_d * [y, f_b]_d = [x*y, f_a*f_b]_{2d}$

We will only focus only on the additively homomorphic property of the packed secret sharing scheme to keep the details of our protocol simple for analysis.

2.4 Sharding

Data sharding is a technique used to partition a large database into smaller, more manageable chunks. Each such chuck of data is called a shard. Sharding can help distribute the load of database queries across multiple servers, allowing for faster and more efficient retrieval of data. This is particularly important for large-scale databases that require high performance and low latency(Bagui and Nguyen, 2015)(Luu et al., 2016).
Sharding can be implemented in various ways, including range-based sharding, hash-based sharding, and directory-based sharding. In range-based sharding, data is partitioned based on a specific range of keys, such as timestamps or alphabetical characters. In hash-based sharding, data is partitioned based on a hashing function, which distributes data evenly across shards. Directory-based sharding involves using a central directory to map data to specific shards.
In this paper, we will shard a set of database fields into smaller subsets such that every subset contains at least one field which has a higher access rate.

3 PROTOCOL

3.1 Overview

We propose three variations of our protocol to cover the different kinds of use-cases as the efficiency will differ widely depending on their applications in different cases. While the basic structure of the protocol will remain largely similar, there will be modifications to it to make it more efficient for each scenario. Accordingly, our security definition will also vary for each case.
All our cases assume a client who wants to maintain a database *DB* protected by a *PIR-Writing* protocol which has N records of C columns each, to be replicated across m servers in some capacity. It is assumed that up to t servers can collude with a semi-honest adversary to learn more information about the files that have been accessed by the client in *DB*. We have three different scenarios:

1. *Column Hiding* : The client wants to hide the column that was updated in a record. The adversary can learn which record was updated but it can not tell the specific column in that record that was changed. An example would be the engagement metrics for a YouTube channel where the adversary will know that some values were updated about the channel, but not the exact fields.
2. *Row Hiding* : The client wants to hide the record that was updated. The adversary can learn which column was updated but it can not tell the specific record that was changed. A useful scenario would be updating an employee's salary so adversary will know that someone's salary changed but will not know for how many employees or for whom.
3. *Database Hiding* : The client wants to hide any kind of update information. The adversary should neither learn the record nor the column that was updated. This can be the case of extremely sensitive data like healthcare data that can be used to draw inferences about both an individual or a wider population.

3.2 Base Protocol

We first present the base version of our protocol. The variations are all derived from it and maintain the same level of security.

Consider D to be a database that follows a tabular structure and $x = (x_1, ..., x_k)$ as a set of values that is a part of one of the columns of this database. The individual elements of x are owned by different clients. The objective of this protocol is to ensure that in case any element from x is updated, no entity can obtain information about which particular element has been updated. This ensures that privacy of the individual elements in x is maintained.

PIUDI - PIR-Writing Protocol:
Common Input. A distributed database with n instances.
Database Initialization. The database instances have been initialised with shares of zero such that $[0, f_a]_d \longleftarrow$ ENCODE(0, f_a)
Client's Input. n shares of a vector x of size k
Database Output. Updated database

1. Client chooses a random polynomial f_b of degree d such that $d = k - 1 + t$ where k is the size of vector x
2. $[x, f_b]_d \longleftarrow$ ENCODE(x, f_b) such that n $\longleftarrow$ size($[x, f_b]_d$)
3. Client distributes the shares to the n instances of the database and adds to every initialised element at the respective database instance : $[0, f_a]_d + [x, f_b]_d \longleftarrow ADD(0, x)$

PIUDI - PIR-Writing Batch Update Protocol:
Common Input. A distributed database with n instances.
Database State. The database instances have shares of a vector x such that $[x, f_a]_d \longleftarrow$ ENCODE(x, f_a)
Client's Input. n shares of a vector y of size k such that a client wants to add each element of y to each element of x at the respective vector positions: $[y, f_b]_d$ $\longleftarrow$ ENCODE(y, f_b)
Database Output. Updated database after performing the following operation: ADD(x, y) $\longleftarrow [x, f_a]_d + [y, f_b]_d = [x + y, f_a + f_b]_d$

1. Client chooses a random polynomial f_b of degree d such that $d = k - 1 + t$ where k is the size of vector y
2. $[x, f_b]_d \longleftarrow$ ENCODE(y, f_b) such that n $\longleftarrow$ size($[y, f_b]_d$)
3. Client distributes the shares to the n instances of the database and adds to every initialised element at the respective database instance : $[x, f_a]_d + [y, f_b]_d \longleftarrow ADD(x, y)$

3.3 Protocol Variations

3.3.1 Column-Hiding PIUDI Protocol

The adversary should not learn which attribute/column was updated for a particular record in the database.

Common Input. Same as base protocol
Database Initialization. Same as base protocol
Client's Input. Same as base protocol
Database Output. Same as base protocol

1. Client chooses a random polynomial f_b of degree d such that $d = k + t$ where the vector x is a vector containing a row of the table and $k = c$ is the number of columns in the database.
2. To update a value, $[x, f_b]_d \longleftarrow$ ENCODE(x, f_b) such that n $\longleftarrow$ size($[x, f_b]_d$)
3. Client distributes the shares to the n instances of the database and adds to every initialised element at the respective database instance : $[x, f_a]_d + [y, f_b]_d \longleftarrow ADD(x, y)$

This results in all record values in our updated row being refreshed with new shares across all the servers, while the other rows will remain unchanged. Adversary can not guess which value change caused this change in the row.

3.3.2 Row-Hiding PIUDI Protocol

This protocol ensures that the adversary can not learn which row/record was updated corresponding to a particular attribute/column update in the database.

Common Input. Same as base protocol
Database Initialization. Same as base protocol
Client's Input. Same as base protocol
Database Output. Same as base protocol

1. Client chooses a random polynomial f_b of degree d such that $d = k + t$ where the vector x is the column to be updated and $k = c$ is the number of rows/records in the database.
2. To update a value, $[x, f_b]_d \longleftarrow ENCODE(x, f_b)$ such that n $\longleftarrow$ size($[x, f_b]_d$)
3. Client distributes the shares to the n instances of the database and adds to every initialised

element at the respective database instance : $[x, f_a]_d + [y, f_b]_d \longleftarrow ADD(x, y)$

This results in all column values corresponding to our column change being refreshed with new shares across all the servers, while the other columns will remain unchanged. Adversary can not guess which value change caused this change in the column.

3.3.3 Cell-Hiding PIUDI Protocol

This protocol ensures that the adversary can not learn which column was updated for a particular record in the database. This is achieved by keeping the vector x as the entire database of size $r \times c$. The rest of the protocol is the same as the base protocol. This results in all the columns for all the records being refreshed with new shares across all the servers. While this is more inefficient than the other two variations, cell-hiding protocol is still more efficient than previous *PIR-Writing* protocols in a distributed setting.

4 PROOF OF SECURITY

While our protocol variations differ in the number of shards and secret shares, their security is dependent on the base protocol. WLOG, we define our security as the following game between a challenger and an adversary for the base protocol: Let there be a challenger running the protocol over n servers in the presence of an adversary A such that up to t servers can be compromised by A. Given this setup:

1. A selects a database of $N = r \times c$ records and sends it to the challenger.
2. Challenger processes the database through the protocol and distributes the shares across n servers such that each share contains a vector x of k secrets.
3. A now selects two subsets of columns S_0 and S_1 from the same row of the database to be modified. The new updated values are also selected by A. A sends these subsets along with their indices and new values to be updated to the challenger. The choice of the subsets is restricted based on the hiding variation of the protocol we have chosen.
4. Challenger decides to randomly choose one of the two subsets S_b and only modifies that in the database according to the protocol.
5. A observes the modified database and outputs its guess for the value for $b = 0, 1$.

Let P_b be the probability with which A outputs 1 given $b = 0, 1$.

4.1 Definition

The database update $(k - t - N, S_b)$ is private, if for all semi-honest PPT adversaries A, we have $P_0 - P_1$ as being negligible i.e., A is not able to guess the column that the client modified.

Proof. To prove our protocol's security we will show how it can be reduce to the packed secret sharing *(PSS)* protocol mentioned in section 2.4.
For the given database, the client first distributes the database values using a *(PSS)* scheme with the packed secret represented with a vector x of size k for each shard. For example, in row-hiding protocol, the client will have $|c|$ number of vectors each of size $k = |r|$ to represent the values in the database as packed secrets. In the game with the adversary, the client then chooses one subset for updating and converts it into secret vectors x_j^{new} of size k. These secret vectors will depend on the protocol variation. For example in case of row hiding, our subset values will represent parts of some columns and all those columns will be used as secret vectors.

Now for each such vector x_j^{new}, the client produces a polynomial f_j and creates shares s_{ij} for each database hosting node db_i hosting corresponding database shards shards, using *PSS* such that all shares for that vector are updated across the distributed system. For any node db_i, give its current share value for the vector, $s_i^{current}$, its new share will be updated to $s_i^{new} = s_i^{current} + s_{ij}$. The updated shares when reconstructed using *(PSS)* will result in the updated values as had been demanded by the adversary A. But, since all the shares for the entire vector have been modified, from the adversary's view, it will not be able to guess with more that negligible probability, which of the given subsets was actually had its values changed. Additionally, since we are following the *(PSS)* protocol to distribute and update the shares across the servers, the security definition will hold, i.e., the adversary with control of only up to $t + k - 1$ servers will not be able to learn anything about the new values from its shares alone. □

For simplicity, the following restrictions will be applied on the adversary in the security game, for our three variations: For column and row-hiding, all the values from each subset should (ideally) only come form a single column/row. In case, this is not the case, the client will need to update the shares for all the rows/columns from which the two subsets draw their values. For the cell-hiding, the entire database has to be updated for change in a single value. However, we restrict our adversary to selecting the subsets in some

pattern, then we can achieve much more efficient and privacy-preserving outcomes.

5 ANALYSIS

To the best of our knowledge, all other protocols in this field have been designed specifically for a single client-server model. That is to say, the focus of their design has been on improving the efficiency of the computation and communication aspects of the protocol in a non-distributed setup. All previous protocols like Lipmaa's (Lipmaa and Zhang, 2010) results in a complete update of the entire database i.e., all n encrypted records will be modified. For a distributed database, this would require necessary updates in all the locations/shards which would cost around $O(n^2)$ updates assuming there are n instances of the database for high availability. In a more particular scenario where the data is hosted on a blockchain, this would also result in immense gas costs. Others, like (Ishai et al., 2004) have implemented a distributed setup, somewhat similar to ours, but it only works for private information retrieval and not for updating. Since our protocol focuses on minimizing the net difference in terms of records changed between the original and updated database, in terms of communication and computational efficiency, it only needs $O(n)$ updates assuming there are n instances of the database for high availability for updating a number of shares related only to the subset.

Batch Updates. The existing *PIR-Writing* protocols do not explicitly address the issue of batch updates, which is a critical consideration for reducing gas costs on public blockchains(Sguanci et al., 2021). The reason for this is that these protocols typically involve accessing and updating individual elements of a distributed database one at a time, which can quickly become prohibitively expensive in terms of the gas fees required for each transaction. However, our protocol takes a different approach by utilizing packed secret sharing, which allows us to compress an entire set of data into a single field element. By doing so, we are able to update multiple elements within the data set by only modifying this single field element at all instances of the distributed database. This approach drastically reduces the number of transactions required to update the entire data set, leading to significant cost savings in terms of gas fees.

Privacy Improvement. Unsupervised sharding could lead to information leakage. However, Sharding the dataset based on the apriori knowledge of the user access patterns can turn out to be an efficient method to improve privacy in practical scenarios. A trivial way to achieve this could be by distributing data with patterns of frequent access uniformly into different shards.

6 APPLICATIONS

There are several important applications of our protocol in various fields such as medical research, finance, and government, where sensitive data must be stored and manipulated securely and privately. Our protocol can be used as an add-on with existing protocols without modifying the system drastically. This provides an easy mechanism to protect existing protocols that are susceptible to side-channel attacks. This protocol is useful to protect clients data whenever it is updated on a remote unreliable server. For example, in case of medical information being store on a hospital server. With the optimizations in place, it is very difficult for an adversary to glean information about which patient record was updated.

7 CONCLUSION

In this paper, we have presented a novel information theoretic *PIR-Writing* protocol, *PIUDI*, that is highly suitable and efficient for distributed database settings. The protocol is designed to mitigate two main types of attacks, correlation attacks and frequency attacks, which have been identified as major vulnerabilities in existing *PIR* protocols.

The proposed protocol not only addresses these vulnerabilities but also improves efficiency using batch updates. This makes our protocol highly efficient and scalable, which is critical for distributed architectures. Another key advantage of this protocol is that it is highly suitable for public blockchains. With the growing popularity of blockchain technology, reducing the gas cost is a critical consideration for any blockchain-based protocol. This protocol achieves this by drastically reducing the gas cost, making it a highly desirable solution for blockchain-based applications.

Future work in this area could explore further improvements to the protocol, such as exploring more efficient methods of batch updates or investigating additional attack types that may be mitigated through the use of this protocol. Another avenue of future research is reducing the gas cost of updates when the data is completely on blockchain without intervention off-chain programs. We hope that our work provides an excellent foundation for such research in this area,

Table 1: Comparison of related schemes in distributed database setting with n instances each with m data objects each of l bits.

Scheme	Lipmaa	Boneh	Chandran	PIUDI
Communication	$(logn+l)k*m$(best case)	$O(l*m\sqrt{n})$	$O(m*\sqrt{l^{1+\alpha}n})polylog(n)$	$O(l*n)$
Computation	$O(m*(n*logn+n))$	$O(l*m*n)$	$O(n*m*polylog(n))$	$O(n)$
DB size change	None	None	None	None

Table 2: Gas fee comparison between trival encrypted db(Best case), homomorphically encrypted db(Best case) and our approach for updating n data objects with c being the gas cost for a trivial addition operation.

Scheme	Encrypted	Homomorphic encrypted	PIUDI
Single update gas cost	$O(n*logn+n)*c$	$O(n*logn+n)*c$	$O(1)*c$
Batch update gas cost	N.A.	N.A.	$O(1)*c$

and we anticipate it will have practical implications in the field of privacy-preserving distributed databases.

REFERENCES

Bagui, S. and Nguyen, L. T. (2015). Database sharding: to provide fault tolerance and scalability of big data on the cloud. *International Journal of Cloud Applications and Computing (IJCAC)*, 5(2):36–52.

Boneh, D., Kushilevitz, E., Ostrovsky, R., and Skeith, W. E. (2007). Public key encryption that allows pir queries. In *Advances in Cryptology-CRYPTO 2007: 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007. Proceedings 27*, pages 50–67. Springer.

Chor, B., Kushilevitz, E., Goldreich, O., and Sudan, M. (1998). Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981.

Dwork, C., Smith, A., Steinke, T., and Ullman, J. (2017). Exposed! a survey of attacks on private data. *Annual Review of Statistics and Its Application*, 4:61–84.

Franklin, M. and Yung, M. (1992). Communication complexity of secure computation. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 699–710.

Gasarch, W. (2004). A survey on private information retrieval. *Bulletin of the EATCS*, 82(72-107):113.

Goldreich, O. and Ostrovsky, R. (1996). Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473.

Grubbs, P., Ristenpart, T., and Shmatikov, V. (2017). Why your encrypted database is not secure. In *Proceedings of the 16th workshop on hot topics in operating systems*, pages 162–168.

Ishai, Y., Kushilevitz, E., Ostrovsky, R., and Sahai, A. (2004). Batch codes and their applications. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 262–271.

Ishai, Y. and Paskin, A. (2007). Evaluating branching programs on encrypted data. In *Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007. Proceedings 4*, pages 575–594. Springer.

Islam, M. S., Kuzu, M., and Kantarcioglu, M. (2012). Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In *Ndss*, volume 20, page 12. Citeseer.

Lacharité, M.-S., Minaud, B., and Paterson, K. G. (2018). Improved reconstruction attacks on encrypted data using range query leakage. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 297–314. IEEE.

Lipmaa, H. and Zhang, B. (2010). Two new efficient pir-writing protocols. In *Applied Cryptography and Network Security: 8th International Conference, ACNS 2010, Beijing, China, June 22-25, 2010. Proceedings 8*, pages 438–455. Springer.

Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., and Saxena, P. (2016). A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 17–30.

Papadimitriou, A., Bhagwan, R., Chandran, N., Ramjee, R., Haeberlen, A., Singh, H., Modi, A., and Badrinarayanan, S. (2016). Big data analytics over encrypted datasets with seabed. In *OSDI*, volume 16, pages 587–602.

Popa, R. A., Redfield, C. M., Zeldovich, N., and Balakrishnan, H. (2011). Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the twenty-third ACM symposium on operating systems principles*, pages 85–100.

Popa, R. A., Stark, E., Helfer, J., Valdez, S., Zeldovich, N., Kaashoek, M. F., and Balakrishnan, H. (2014). Building web applications on top of encrypted data using mylar. In *NSDI*, volume 14, pages 157–172.

Sguanci, C., Spatafora, R., and Vergani, A. M. (2021). Layer 2 blockchain scaling: A survey. *arXiv preprint arXiv:2107.10881*.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

Stefanov, E., Dijk, M. v., Shi, E., Chan, T.-H. H., Fletcher, C., Ren, L., Yu, X., and Devadas, S. (2018). Path oram: an extremely simple oblivious ram protocol. *Journal of the ACM (JACM)*, 65(4):1–26.

Zolotukhin, M., Hämäläinen, T., Kokkonen, T., and Siltanen, J. (2014). Analysis of http requests for anomaly detection of web attacks. In *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, pages 406–411. IEEE.