Spring 10-13-2020

DBKnot: A Transparent and Seamless, Pluggable Tamper Evident Database

Islam Khalil
ikhalil@aucegypt.edu

The American University in Cairo

DBKnot

A Transparent and Seamless, Pluggable Tamper Evident Database

A Thesis Submitted to

The Department of Computer Science and Engineering

In Partial Fulfilment of The Requirements For

The Degree of The Master of Science

by

Islam Khalil

Supervisors

Dr. Sherif El-Kassas

And

Dr. Karim Sobh

Fall 2020

Abstract

Database integrity is crucial to organizations that rely on databases of important data. They suffer from the vulnerability to internal fraud. Database tampering by internal malicious employees with high technical authorization to their infrastructure or even compromised by externals is one of the important attack vectors.

This thesis addresses such challenge in a class of problems where data is appended only and is immutable. Examples of operations where data does not change is a) financial institutions (banks, accounting systems, stock market, etc., b) registries and notary systems where important data is kept but is never subject to change, and c) system logs that must be kept intact for performance and forensic inspection if needed. The target of the approach is implementation seamlessness with little-or-no changes required in existing systems.

Transaction tracking for tamper detection is done by utilizing a common hashtable that serially and cumulatively hashes transactions together while using an external time-stamper and signer to sign such linkages together. This allows transactions to be tracked without any of the organizations' data leaving their premises and going to any third-party which also reduces the performance impact of tracking. This is done so by adding a tracking layer and embedding it inside the data workflow while keeping it as un-invasive as possible.

DBKnot implements such features a) natively into databases, or b) embedded inside Object Relational Mapping (ORM) frameworks, and finally c) outlines a direction of implementing it as a stand-alone microservice reverse-proxy. A prototype ORM and database layer has been developed and tested for seamlessness of integration and ease of use. Additionally, different models of optimization by implementing pipelining parallelism in the hashing/signing process have been tested in order to check their impact on performance.

Stock-market information was used for experimentation with DBKnot and the initial results gave a slightly less than 100% increase in transaction time by using the most basic, sequential, and synchronous version of DBKnot. Signing and hashing overhead does not show significant increase per record with the increased amount of data. A number of different alternate optimizations were done to the design that via testing have resulted in significant increase in performance.

Contents

List of Figures

1 Introduction

This thesis proposes a novel solution to protecting database integrity by providing a transparent and seamless middleware for securing database transactions against possible tampering by individuals who have full administrative access to the database and all its related infrastructure.

1.1 Fraud and Database Integrity

With the growing need for digitalization by various industries and the pervasiveness of software systems of varying levels of sophistication, there is also a growing need for securing such systems that manage vital information that impacts aspects of our everyday lives. Such information like bank transactions, medical information, government records, as well as many other critical information often fall as prey for perpetrators who often are insiders or collude with insiders to commit their fraud crimes. External malicious actors as well are in many cases the players responsible for committing fraud and tampering of sensitive databases. Many of such cases involve tampering with existing systems and making fraudulent transactions that go unnoticed due to the fact that they are committed by insiders who already have access and permission to the systems they tamper with. According to the Association of Certified Fraud Examiners (ACFE) 2018 report[1], $7 Billion of losses were incurred due to internal fraud alone with an average fraud scheme going for 16 months unnoticed. Small Business lose twice as much as big organizations due to their lack of proper access to **a) Internal control processes that mitigate against such fraud** and **b) Systems in place that protect against such tampering**.

According to Harvard Business Review[2], more than 80 million insider security breaches occur every year costing tens of billions of dollars every year in the US alone. According to Accenture[3] and The World Economic Forum (WEF)[4], the cost of insider malicious activity constitutes 15% of all cybercrime. The IETF's RFC 4810[5] [6] guidelines for "Long Term Archive Services Requirements" indicate that non-repudiation and integrity are important to any store of data to protect against potential tampering.

The number of internal fraud cases resulting in compromising the integrity of organizations' data is increasing year after year [2]. For example, in the year 2010 alone, internal fraud has increased at a rate of 20%. One of the causes of such an increase is the increasing complexity and use of IT solutions and its corresponding increase in the number of internal and external

stakeholders needed to operate such systems. For example, an incident when $350,000 were stolen from 4 Citibank customers by employees of a software and service company that Citibank had contracted[2].

The cost of such incidents is very high in addition of the difficulty in identifying such fraudulent activity that can go for long durations unnoticed.

1.2 Outsider vs. Insider Threats

As we will see below, different techniques for securing systems and their corresponding data exist. Most of the surveyed solutions focus on protecting database systems from tampering by outsiders and are less effective when it comes to insiders. For example, database administrators who have access rights to the whole database systems. Our goal is to detect possible tampering even by insider users who have legitimate and full access rights to the data and the highest privileges.

1.3 Regulatory Compliance and Governance

As a result, various governments have put in place different regulations to reduce/eliminate such risks. Among such regulations are the Gramm-Leach-Bliley Act by the US Federal Trade Commission (FTC) [7] which mandates that companies engaging in financial service put in place necessary measures to safe-guard their sensitive data against tampering. Another act that was decreed by the US congress is the Sarbanes-Oxley Act[8] (SOX) which mandates that companies protect their data and ensure that destruction of evidence does not occur for the purpose of later investigation of corruption and fraud cases. This act was made as a reaction to a number of major corruption scandals including Enron and WorldCom. Health Insurance Portability and Accountability Act[9] (HIPAA) by the US Department of Health and Human Services (HHS) is also an example which regulates access and changes to medical records.

1.4 Databases

Database systems started to be used in the 1960s and since then have quickly been adopted widely to become the dominant form of data storage. Database systems range from tiny databases that store configuration parameters of different systems, small databases that manage small aspect of our everyday life to large systems that manage the data of large institutions like banks and governments.

Over the past years, database management systems have evolved greatly. Different flavors include time-series databases, spatial or temporal databases, to non-relational

document/object databases, and graph databases. With the increasing complexity of data being managed, new models and paradigms are evolving to increase the flexibility or efficiency of dealing with such evolving needs.

Due to relational databases being the primary technology used to store and manage enterprise data, it is necessary to have the means to maintain integrity and detect tampering and detecting fraudulent actions resulting from tampering with such data rather than letting them go unnoticed or relying on trust.

1.5 Common Strategies

According to [2], a number of common practices are adopted by organizations to address their security needs. Such practices, even though are effective against different forms of external or internal threats, they might not be effective in the detection of fraudulent practices by insiders who already have legitimate access to the data being compromised and have not gained access to the data by means of compromising other security loopholes.

Among the common security practices are:

- **Access control:** Proper access control measures are the first steps for securing data. Implementing proper access control ensures that only authorized persons have access to their corresponding data. Access control techniques and managing permissions and privileges vary spanning a large spectrum of degrees of complexity of application and granularity of access permissions. Access control however does not protect against fraudulent actions by persons who – due to the nature of their work - already have full administrative access to the target data. Database system administrators for example are among those who have the highest level of privileges for data access.

  Password policies and employee awareness: Humans are most often the weakest link in securing any system or network. Breaches are often caused by an employee leaking their passwords (possibly unintentionally) to an unauthorized person. In the cases we are targeting, awareness will not solve the problem because the malicious person is already authorized to access the data and is not in need for stealing any access control passwords or tokens in order to gain access.

- **Separation/segregation of duties (SoD):** One of the common practices for securing data is to split the responsibility for affecting a transaction between two different people who are required to cooperate in order to make the transaction. According to[10], SoD is defined as follows: “The principle of Separation of Duty (SoD) is used to prevent threats posed by malicious insiders (e.g., corrupt employee) or detect errors unintentionally committed by an unknowing employee. It works by distributing trust in performing a task to multiple individuals and only through the cooperation of those individuals that the full authority to perform the task can be acquired.”. The solution we are proposing to the problems borrows from SoD the logging and transaction chaining part and moves it to an external service. SoD however targets applications that are different from what we are targeting. For example:
    - SoD is designed into the business process itself rather than transparently embedded into existing database applications
    - SoD secures data on an application level (which is important) not against insiders who have access to the database-level itself.

- **Updates and security patches:** Maintaining systems up-to-date and ensuring that all issues, bugs, and vulnerabilities are addressed in a timely manner is crucial to protecting against attackers taking advantage of such vulnerabilities to gain unauthorized access to data. Regular updates however, although extremely necessary, does not cover insider-initiated fraud.
- **Firewalls:** Similar to access control, firewalls play a more general role, among other roles, to prevent unauthorized malicious access to internal systems. Access control works on a systems level, firewalls on the other hand work on a network level. A firewall is also called a Border Protection Device (BPD) because it controls the inflow and outflow of traffic between different network zones of different trust levels by implementing certain security policies.[11] There is a wide range of firewalls ranging from simple packet-filters to stateful multi-layer firewalls, to application level firewalls. Firewalls can either be a host based firewall or sitting on the border and securing a whole network. [11]–[16]

1.6 Problem Characterization and Motivation

1.6.1 Motivation

With a $7 billion annually of losses due to internal fraud with each scheme takes on average 16 months totally unnoticed, there is a need for effective and efficient tamper detection systems.

With the increased reliance on database systems, vulnerability to fraud through tampering with the organizations' data is growing. Thus, there is an increased need for being able to detect such tampering in order to deter fraudsters and reduce the probability of malicious tampering with data. Existing work has been addressing this problem on a number of levels. Some of which assume that there is a central point of trust inside the organization. Other solutions address the same problem in different contexts like protecting documents or whole immutable database rather than a transactional database system. Other systems rely on an external notary that gets a copy of all data which constitutes a large overhead on the database operations. In addition, some of the solutions require significant changes to the underlying architecture of the target systems.

1.6.2 Problem Characterization

Systems that works using a traditional database are vulnerable to tampering. The current challenge is that there are no existing solutions that safeguard internal processes on a system-level by providing tamper detection against malicious insiders. Most solutions protect data on the application level most solutions do not do so without requiring that some data leave the premises (or servers) of the enterprise and be within the control of a third party. Such a vulnerability exists on all ranges of applications including ones that deal directly with a database, other systems that are built utilizing an Object Relational Mapping framework to also web based REST APIs and microservice architectures.

In addition, there needs to be a solution that allows introducing these features into already existing applications without requiring major revamps into existing infrastructures.

1.6.3 Our General Direction

The goal of this work is to design a solution that enables systems based on traditional databases to be tamper-evident. Different integration models are to be discussed (on the ORM level, database level, or web-service level). The primary goal is to eliminate the need for trust inside the organization while minimizing the overhead added by the solution. Ease

of integration is key while requiring zero-or-little changes to existing systems. The solution should be able to detect tampering either by external hackers or by internal malicious employees, staff, and system administrators who have full permissions on the target database. This is done by relying more on information accountability rather than information restriction[17], [1], [18].

In the process of coming up with such a solution, a number of different technologies will be examind, in addition to related work. Important assessment criteria for the proposed solution will be **a) Impact on functionality** of existing systems, and **b) Performance** impact

The proposed approach is designed be suitable to a specific class of problems (primarily append-only systems) but on the other hand provides other advantages that are not present in the existing proposed solutions. The primary target advantages are: a) Less impact on performance, b) Ease of applying without the need to significantly disrupt existing systems in place, c) data does not leave the premises of the existing systems.

There are numerous applications that would significantly benefit from an append-only protected ledger database. Especially in cases where traditional block-chain would be an overkill and where the decentralization of trust is not needed. They still can benefit from transaction security without having to do a full block-chain deployment.

Example applications are:

- Server security logs
- Banking transactions
- Accounting ledgers in enterprises
- Notary and real-estate records
- Birth and death records
- Time and attendance systems

As illustrated in Figure 1 - Insider Threat and Figure 2– Database Management System Interaction, in most of such applications, users, or systems, or IoT devices usually interact with applications that reside on top of a database. Inside the organization, there are users with different privilege levels the top of whom is the system administrator of the database. Possible tampering could be committed on different levels. On a system administrator level

however the risk is that a) The sysadmin can commit the fraud and b) The sysadmin can cover-up any traces or logs of the fraudulent activity he has performed since he/she is the one responsible for all system permissions, logs, monitoring, etc.

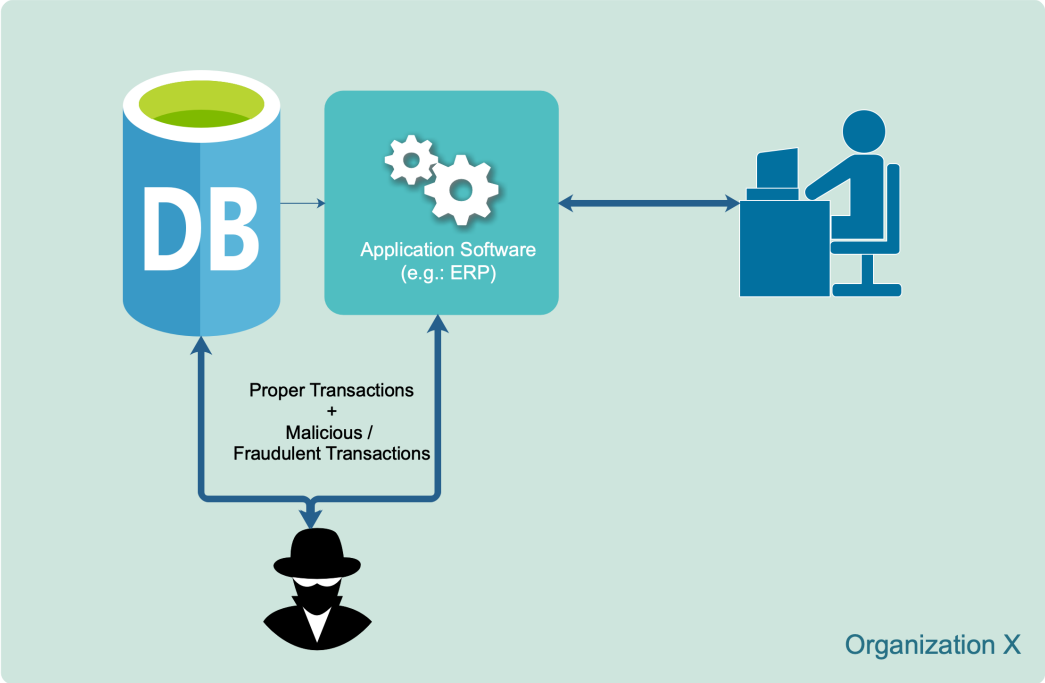


Figure 1 - Insider Threat

1.7 Thesis Layout

This thesis is structured as follows: The first chapter is a brief introduction on the need for tamper-evident database systems. The second chapter covers a brief background on technologies that are in contexts surrounding our proposed solution that we have looked at. Some solutions are also included in the background that even though are not directly impacting the solution but to present how similar classes of use-cases are solved in other areas (filesystems, operating systems, document repositories, etc.). Chapter 3 contains a brief survey of a set of different works in the same direction as our proposed solution. Some of them are listed for comparison and others were studied further due to their innovative ways of solving parts of the problem that we can benefit from or build on. The 4th chapter details out our solution to the problem (DBKnot). Followed by explanation of the experiments and results of the experimentation and testing in Chapter 5. Finally we close with the conclusion and areas where we see that future work could lead to further improvements.

2 Background

2.1 Database & Database Management System

2.1.1 Database Management Systems & Applications

The database management system (DBMS) is a software that facilitates the interaction between end users (application users, database administrators, and developers), applications, and the database itself for developing, analyzing, and interacting with data[19]. It also provides the necessary tools for the database design and development. The DBMS allows end users and applications to perform various database operations such as creating new records, saving, updating, deleting, reading, and manipulating data. The DBMS gives complete control to the database administrator to decide which data can be viewed by users and operations that can be performed by various users interacting with the database. As illustrated in Figure 2, the DBMS sits between the database and software-applications/users that need to interact with the database as the software necessary for managing all the database.

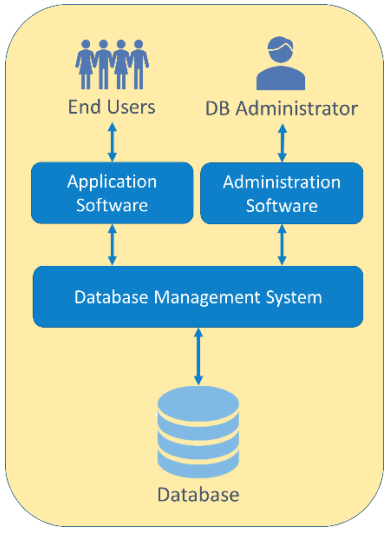


Figure 2– Database Management System Interaction

Applications where we use database management systems and interact with databases include but are not limited to telecom, banking systems, sales, airlines, education, and online shopping. For example, databases are used in the banking system for storing customer information, tracking day to day credit and debit transactions, generating monthly bank statements, etc.

2.1.2 Relational vs. Non-relational Databases

A database management system may be classified according to the database model that it supports. A database model simply represents the way data is structured, stored, and retrieved inside a database. Relational databases have been a common option for the storage of information since 1980s [20] or even further before. In relational databases, data is stored in tables containing rows and columns. Relationships are established through primary and foreign keys. An example of a relational database model is illustrated in Figure 3 where the data of products, customers, and orders are modeled using tables “Products”, “Customer_Orders”, and “Customers_Orders_Products”. Relationships are modeled using fields (foreign keys) inside each table.

Figure 3– Relational Database Model

Over the last few years, many new types of database management systems have emerged that are being labeled as non-relational databases or NoSQL systems [21]. These database management systems provide other ways for storage and retrieval of data that are modeled in means other than the tabular relations used in relational databases. The NoSQL wave was triggered by the Web 2.0 where companies like Amazon, Facebook, and Google introduced it. Some of these types are key-value stores, document databases, graph databases, and column-oriented stores [22].

2.1.3 Object Stores and Key-Value Pairs

One of the limitations of relational databases is that each item (entity) can only contain one coherent and fixed set of attributes. In a bank application example, each aspect of a customer's relationship (details, investments, accounts, and loans) with a bank must be stored as a separate row item in separate tables. All these tables are linked to each other using relationships represented as foreign keys inside each table.

A key-value pairs or key-value stores on the other hand stores all related data in one object. In the same bank example, an object in a key-value store can represent all data related to the customer instead of having a table for each type of data.

A key-value store is a database which uses an array of keys where each key is associated with only one value in a collection like a dictionary. A key-value store has one-way mapping from the key to the value as illustrated in Figure 4. The value can be stored as an integer, string, a JSON structure, or an array, or any other form of a complex object along with a key used to reference that value. The key-value stores are similar to hash tables where the keys are used as indexes, thus making it quick to reach particular entities by its key [22].

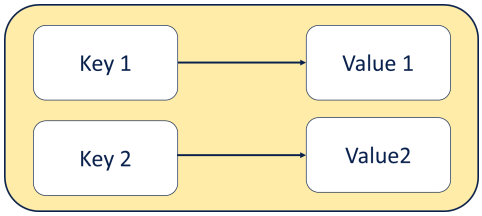


Figure 4- Key-Value Pairs

One of the advantages of key-value stores is its similarity to object-oriented semantics in cases where the value itself is often a complex object. In fact, the value in itself can also either be or refer to another key-value store. This comes however at the expense of performance when it comes to formulating complex data queries for information retrieval.

2.1.4 Database Transactions

A database transaction is a logical set of operations that are independently executed during one database access operation [19]. All types of database access operations which are held between the beginning and end of a transaction statements are considered as a single logical transaction. During the transaction, the database becomes inconsistent as illustrated in

Figure 5. Once the database transaction is committed, the database is changed from one consistent state to another consistent state.

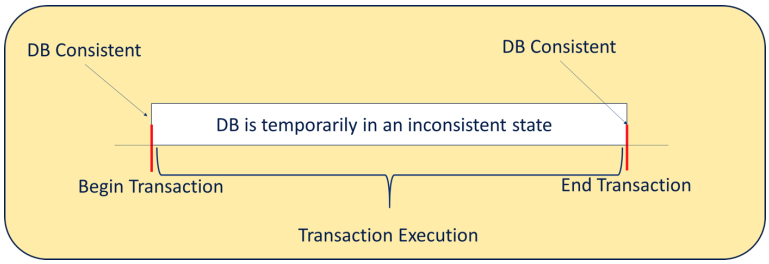


Figure 5 - Database Transaction Execution

Database transactions are implemented using different DB-level locking to guarantee the mutual exclusiveness of operations and therefore ensuring that they are atomic.

In the case of failure, the DBMS performs a rollback operation to leave the database in the original state before starting the execution of the failed transaction rather than leaving it in an unknown and inconsistent state.

2.1.5 Database Transactions ACID Properties

ACID (ACID stands for Atomicity, Consistency, Isolation, and Durability) properties are used during transaction processing for maintaining the integrity of a database. A relational database transaction must satisfy all ACID properties.

- **Atomicity**: No partial execution of a transaction. A transaction is a single unit of operation to be executed entirely or not executed at all.
- **Consistency**: A database should be moved from one consistent state to another once the transaction is executed.
- **Isolation**: During concurrent transaction execution, one transaction intermediate results should not affect the other executing transactions. Any transaction should be executed in isolation from other transactions.
- **Durability**: The changes of a database resulting from a transaction should be persistent even in the case of system failures.

2.1.6 Server-side Database Triggers

A database trigger is a special stored procedure that is executed automatically when specific actions occur within a database on a table or view [19]. Triggers can be defined to execute

either before or after any INSERT, UPDATE, or DELETE operation, either once per modified row, or once per SQL statement. If a trigger event occurs, the trigger's function is called at the appropriate time to handle the event.

2.1.7 Object-Relational Mapping (ORM)

Database management systems interact with databases using a query language. End users use query languages related to the database model they are using. SQL is the standard query language for relational databases. Cypher query language for example is used for a graph database management system named Neo4j [23]. Even though it is relatively standard, different database vendors have different "flavors" of their SQL implementations which impacts the interoperability of an application across different vendors' database management systems.

Object-relational mapping is the idea of being able to write queries and manipulate data inside a database using the object-oriented paradigm of the preferred programming language. The idea is to abstract the database system so that switching from a DBMS like MySQL to another one as PostgreSQL is possible without having to change the application layer. Examples of ORM layers are SQLAlchemy[24], Hibernate, and Django ORM[25].

2.1.8 Replication and Transaction Streaming

Database replication is the best strategy towards achieving high availability during disasters and providing fault tolerance against unexpected failures. Database replication can be done in at least three different ways:

1. Snapshot Replication: data on one server is simply copied to another server or to another database on the same server.
2. Merging Replication: data from two or more databases is combined into a single database.
3. Transactional replication: user systems receive full initial copies of the database and then receive periodic updates as data changes. Data is copied in near real-time from the master or primary server called publisher to the receiving server/database which called a subscriber.

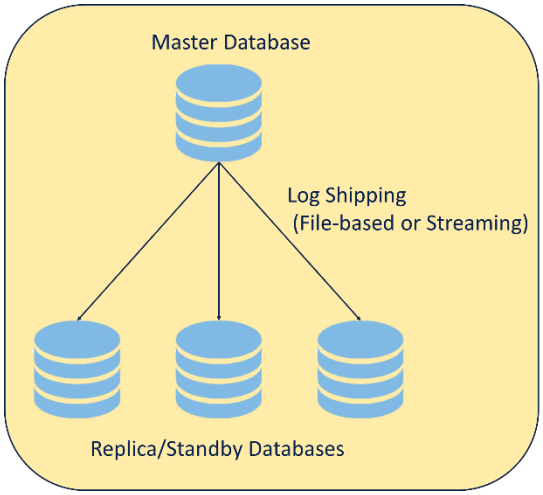


Figure 6 - Transactional Replication

In transactional replication, log shipping is the process of sending WAL (Write Ahead Log or REDO logs) files from the master database to the replicas to execute the logged operations at the replicas. Log shipping can be performed in multiple ways:

- File-based Log Shipping: sending the WAL file once it is filled with changes/transactions records.
- Streaming Log Shipping: streaming/sending WAL records to the backup database as they are generated, without waiting for the WAL file to be filled.

Streaming replication allows a backup server to stay more up to date than is possible with file-based log shipping. Logical decoding extracts changes that are persistent inside a database table into an easy format that can be interpreted without detailed knowledge of the database's internal state [26]. In PostgreSQL, logical decoding is implemented by decoding the contents of the log (WAL) files, which describes changes on a storage level, into an application-specific form such as a stream of tuples or SQL statements.

Some DBMS systems also have support for multi-master replication where users can write to the different master database servers. This is done through implementing a certification process by contacting the different masters for confirmation before performing the final commit [27]. There is usually however a time-gap between affecting the change on a server and on a secondary server. For this reason some call this a "virtually synchronous" replication rather than "real synchronous" replication. To implement this for example, MySQL uses "optimistic locking"[28] such an approach while being very effective in "low data contention" contexts, it however introduces another layer of error checking on the application level[29].

There are two primary topologies used in multi-master configuration[30]:

a) Active/Passive masters: in this case there is only one master active at a time and the second one is on “hot-standby” this is primarily used as a failover scenario.
b) Active/Active: in this case, both masters are active at the same time. Both can read and write to databases.

Generally, it is recommended to stay as much as possible away from active/active topologies due to both the complexity of handling errors and inconsistencies not only on the database but also on the application level, in addition to performance bottleneck caused [30].

2.2 Database Sharding

Sharding is a technique used by database management systems to slice database records into different slices. Sharding is primarily used for load-balancing reasons among different server instances.

The diagram below[31] describes horizontal and vertical database sharding:

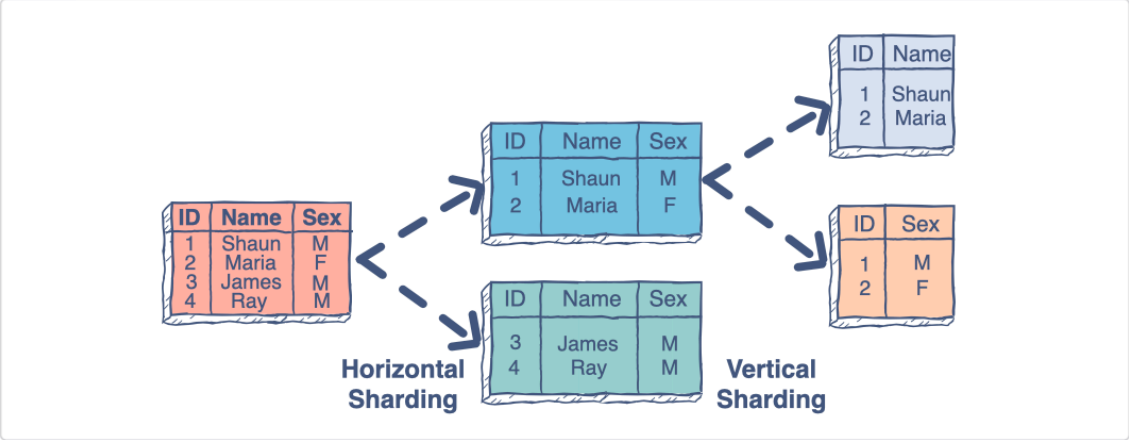


Figure 7 - Horizontal and Vertical Database Sharding[31]

An example of another use of sharding techniques is the Hadoop’s HDFS (Hadoop Filesystem). Hadoop is designed to help reliably distribute data as well as processing to a large cluster of computing and storage resources. Hadoop’s is mostly optimized for high-volume batch processing rather than interactive transactional applications [32]–[34].

2.3 Web Services and REST APIs

A fundamental requirement of software development is sharing data between two or more systems. APIs are application interfaces, meaning that it is a way for one application to share data and interact with another application in a standardized way. Web services APIs which are accessed through a network connection. Representational State Transfer (REST)[35] APIs are a standardized architecture for building web APIs using HTTP methods [36] to communicate data between applications or services. HTTP methods include POST, GET, PUT, PATCH, and DELETE methods. These correspond to create, read, update, and delete (or CRUD) operations, respectively as illustrated in Table 1.

HTTP Method	CRUD	Entire Collection (e.g. /products)	Specific Item (e.g. /products/{id}
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

Table 1 - HTTP Methods

A REST API request and response example is illustrated in Figure 8. The following are examples of different requests performed using a REST API that handles a set of users inside a database can:

- a “GET” request: /user/ returns a list of users inside the database
- a “POST” request: /user/123 creates a user with the ID 123
- a “PUT” request: /user/123 updates user 123
- a “GET” request: /user/123 returns the details of user 123
- a “DELETE” request: /user/123 deletes user 123

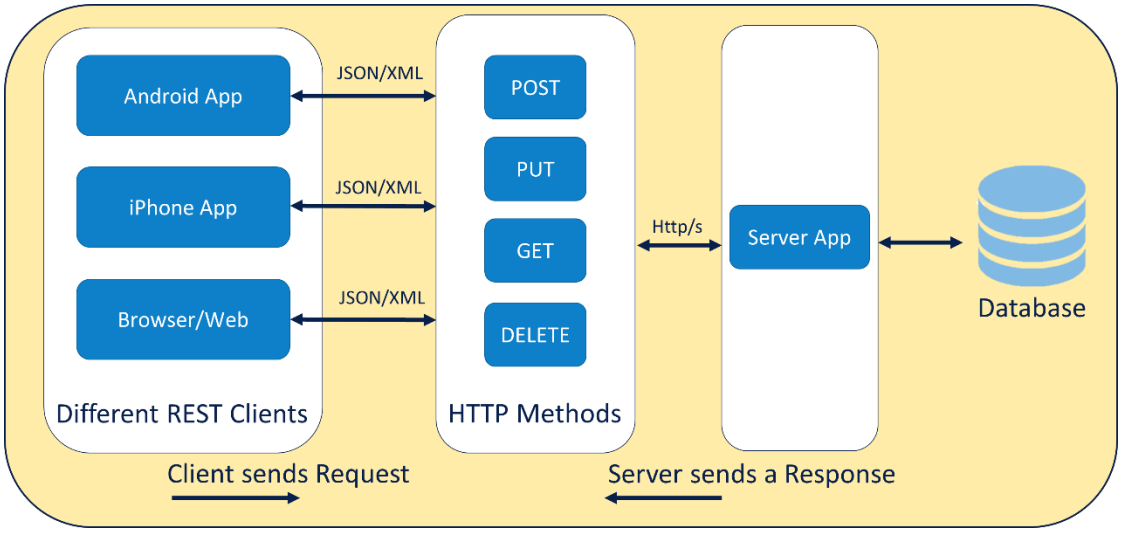


Figure 8- REST API Request and Response

A set of standard recommendations and constraints are available for RESTful web services:

- **Client-Server**: The application makes a request for a specific URL that is routed to a web server to return an HTTP response. The response might also include only data in a JSON format or other data exchange formats.
- **Stateless:** REST is stateless meaning the client request should contain all the information necessary to respond to a request. In other words, it should be possible to make two or more HTTP requests in any order and the same actions will be performed.
- **Cacheable**: A response should be defined as cacheable or not.
- **Layered**: The requesting client need not know whether it is communicating with the actual server, a proxy, or any other intermediary.

2.4 Reverse-Proxy Middleware

A proxy server is an intermediary server that forwards requests for content from multiple clients to different servers within a cluster or even across the web. A reverse proxy server is a proxy server that typically sits behind a firewall in a private network. The reverse proxy will take hold of requests and send them to the appropriate backend server. This allows the system administrator to use a server for multiple applications as well as to ensure smoother flow of network traffic between clients and servers. The reverse proxy server simply intercepts HTTP requests and redirects them to another server without the client knowing it. Since reverse proxies rely primarily on the HTTP protocol which is stateless by nature, reverse proxies are stateless too. State is managed by the HTTP server by means of exchanging a

session identifier (cookie, token, etc.) between the server and clients (web browsers or API consumers).

Common uses of reverse proxy server include load balancing, web acceleration, logging, integration, and security. One of the important uses or a reverse proxy is easy scaling as well as logging and auditing. Since all the incoming traffic is managed by the reverse proxy, it is easier to log and monitor the flow of the traffic.

Examples of a reverse proxy server include Nginx [37], Apache [38], HAProxy[39], and Squid [40]. An illustration of how a reverse proxy server works is presented in Figure 9.

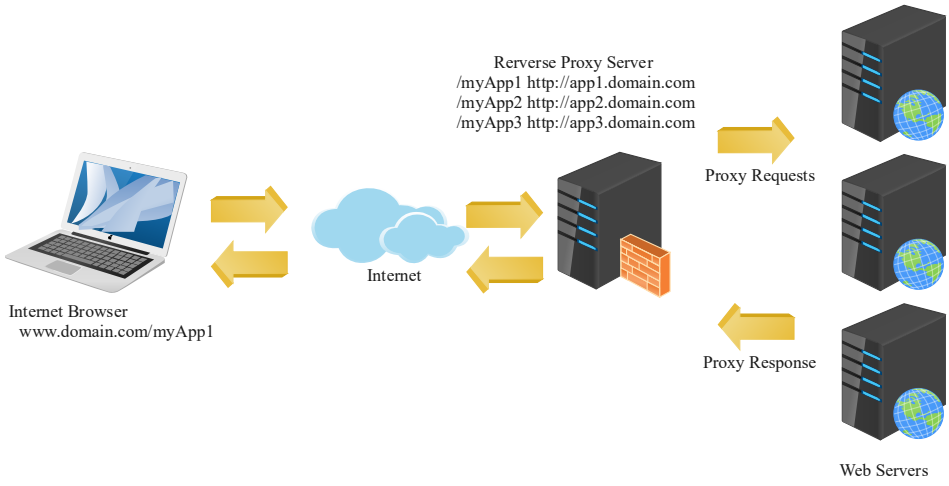


Figure 9 - Reverse Proxy Server Interactions

A reverse proxy middleware intercepts the requests, performs some actions on the request, and then forwards it to a server to get the response. An example of a middleware includes logging that logs specific data from the request, adding location and security data to the request, and can even involve some processing to the request before sending it back to the server to receive the request.

2.5 Message Queues & Publish-Subscribe Models [41]–[44]

Communication between multiple applications can be performed using messaging queuing or simply sending messages to each other. A message queue is considered a temporary message storage medium between the sender and the receiver when the receiving program is busy or not connected. A message contains the data that will be transported between a sender and a receiver application. A message queue, as illustrated in Figure 10, provides an asynchronous communications protocol that temporarily stores a message into a message queue and does not

require an instantaneous action to continue processing. A benefit from using message queues is that it provides asynchronity to the calling application so that execution does not hold until the call is performed or the message is delivered. If a reply is expected, the calling application can later choose to wait until a reply has been received from the called service.

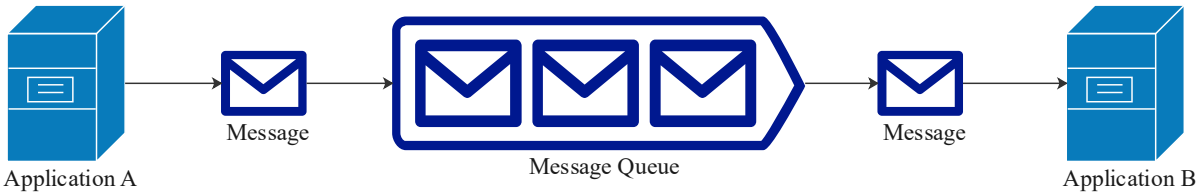


Figure 10 - Message Queue

A publish-subscribe model is a form of a message queue that is used in serverless and microservices architectures. In such model, subscribers subscribe to a topic and whenever a message is published to that topic from a publisher, all subscribers to that topic will receive this message. This model is usually used to decouple applications to increase performance, scalability, and reliability. The publish-subscribe model broadcast messages to different parts of a system asynchronously.

Some database management systems such as Oracle [45] and PostgreSQL [26] support a queue-based publish-subscribe model where database queues act as a temporary store for messages with the capability to allow publish and subscribe based on these queues.

IoT (Internet of Things) applications are a class of systems that often rely on message queues.

2.6 Blockchain

2.6.1 Brief

Blockchain is primarily a distributed ledger of transactions that aims to decentralize trust and secure transactions by making the transactions immutable[46].

The Wikipedia definition of blockchain includes: "*a growing list of records, called blocks, which are linked using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data (generally represented as a Merkle tree).*"[47]

The primary target of blockchain is to achieve the following capabilities:

- Append only
- Immutable
- Decentralized (Peer-to-peer)
- Unmodifiable
- Verifiable
- Permanent

Being structured as a cryptographic hash chain[48] makes a node immune to changes unless all following nodes are changed which is practically not feasible in large peer-to-peer networks. Other work on sequential verification of integrity was looked at.[49]

2.6.2 Trust

Blockchains are designed to operate in a network of decentralized trust. A number of nodes participate in a blockchain in which no node is required to trust another node. Transactions are to be done in a manner that guarantees their security and reliability despite that lack of a trust model. Based on Kevin Werbach in his book "Blockchain and The New Architecture of Trust"[50] , there are four different trust architectures:

- Peer-to-peer trust
- Institutional trust (contracts)
- Intermediary trust (PayPal or credit cards)
- Distributed trust which is what blockchain enables without any individual entities in the system trusting each other.

2.6.3 Blockchain Classes

There are two classes of a blockchain. A private (permissioned) blockchain and a public (permissionless blockchain). In a permissioned blockchain, nodes are authenticated and authorized before joining a network. Permissioned blockchains are usually owned/operated by an organization. A permissioned blockchain is defined as "operating in environments where participants have authenticated and verified identities"[46]. In a permissionless or public blockchains, nodes can join and leave the network without authentication. Bitcoin and most of the cryptocurrencies are considered permissionless blockchains.

2.6.4 Chaining of Blocks

Blockchain got its name from the fact that blocks are chained to each other. Each block contains a set of transactions. Blocks are verified and synchronized with other computers on

the network. Once verified, the blocks are chained to the last block in the blockchain as illustrated in Figure 11 – Chaining of hashes into blocks. To ensure the correct order of the blocks inside the blockchain, each block contains the hash of the previous block. Using the hash of the previous block ensures integrity between transactions. The first block in a blockchain is called the “Genesis block”. Each blockchain has its own genesis block.

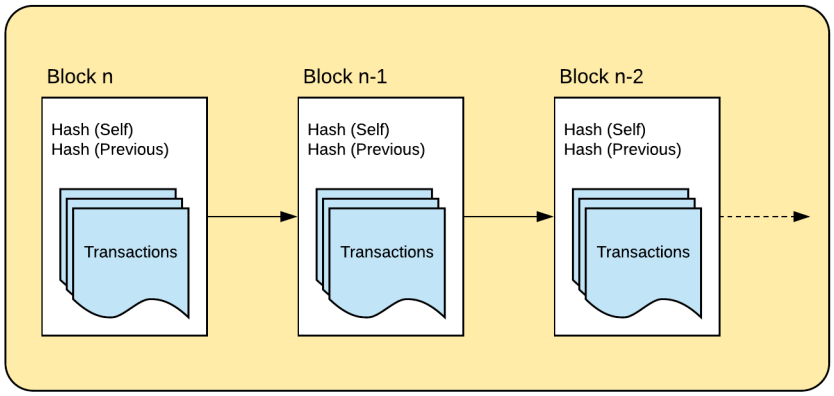


Figure 11 – Chaining of hashes into blocks

Full nodes are computers that stores the blockchain, they ensure fault tolerance in the network as there are no single points of failure due to having multiple full nodes in the same network. Some of the full nodes crypto-currencies are known as miners, these are the nodes that add blocks to the blockchain. Adding blocks to the Blockchain is considered completely secured and verified and this is only made possible because of consensus protocols. A consensus algorithm is a way to reach common agreement between all peers of the Blockchain network. The consensus algorithm makes sure that every new block added to the Blockchain is the only version of the truth that is agreed upon by all nodes in the Blockchain.

There are multiple consensus algorithms that exist including the following:

1. Proof of Work (PoW) is based on solving a complex mathematical puzzle. The first node to solve that puzzle gets to mine the next block.
2. Practical Byzantine Fault Tolerance (PBFT) is designed to work efficiently in asynchronous systems. Fault tolerance can be achieved if the correctly working nodes in the network achieved an agreement while there is a default vote value given to missing messages.

3. Proof of Stake (PoS) is based on investing the coins of the system by keeping them as stake. Validators after that validate blocks by placing bets if they discover a block to be added to the chain. Based on the actual block added, all validators get rewarded depending on the bets they did.
4. Other consensus algorithms include Proof of Burn (PoB), Proof of Elapsed Time, Proof of Capacity, Proof of Activity, Proof of Weight, Leased Proof of Stake, Proof of Importance, and others.

2.6.5 Drawbacks

2.6.5.1 51% Attack[51]

Although not practical in most real-life uses of block-chains specifically, public/permissionless block-chains. If more than 50% of the nodes conspire, they can take control of the whole network. They can monopolize the network, reverse transactions, and take total controls. The 51% Attacks are rare because it needs a computing power that competes with the rest of the network.

The cost of operating a blockchain is huge in terms of computing resources as well as the power consumption of such resources. For example, in blockchain based currencies like bitcoin, the power needed to mine a single coin is comparable (or even higher) than the power used by a household. This is due to the reliance on satisfying constraints like the Proof-Of-Work (PoW) in the process of generating a coin.

2.6.5.2 Performance

While the performance of most blockchain based systems is suitable for many applications, database transactions performance requirements are much higher than what could be satisfied by traditional and generic blockchain implementations[52]. One of the reasons for that is that blockchains are designed to cater to problems that are not always applicable to secure traditional transactional databases where using full blockchain features would be an overkill and an unnecessary tradeoff of performance in order to satisfy unneeded constraints that we can do without. According to[52]: “current block- chains’ performance is limited, far below what a state-of- the-art database system can offer”. According to [52]: “Bitcoin transaction throughput remains very low (7 transactions per second)” and “However, being non-deterministic and computationally expensive, [PoW] is unsuitable for applications such

as stock market, banking and finance which must handle large volumes of transactions in a deterministic manner."[53]

2.6.6 Relevance

Although the general concepts of blockchain provide features or solve problems different from the subject of the research like distribution of trust, proof of work, distribution of data, mining, peer-to-peer, etc. some of the underlying concepts will constitute some of the building blocks of this work. The chaining of hashes is one example. Handling the granularity in the form of blocks and optimization of the signed blocks is another example. Data integrity and tamper resistance is another key common goal.

2.7 Merkle Trees and Merkle DAGs

2.7.1 Brief

A fundament part of a blockchain technology are Merkle trees. Merkle trees are data structures that allow efficient and secure verification of data where non-leaf parent node is a hash of its respective child nodes [54], [55]. A Merkle tree is a binary tree that requires an even number of leaf nodes. A Merkle tree is a derivative of the concept of a Hash chain or a Hash List. A Merkle tree summarizes all block transactions by producing a hash to the entire set of transactions by repeatedly hashing pairs of nodes until the root of the tree. This ensures the integrity of the data transactions. If any change/update happens in a transaction, it will affect the root of the Merkle tree directly. The root of the Merkle tree, Merkle Root, is considered a crucial piece of data as it allows the verification of information efficiently. Each block in a blockchain has exactly one Merkle Root. Merkle trees work by structuring the hashes of a large number of data blocks into a hierarchical successive hash trees.

As illustrated in Figure 12, a Merkle tree of four transactions A, B, C, and D in a block is hashed. Pairs of leaf nodes are summarized in the parent nodes by hashing Hash A and Hash B resulting in Hash AB and the same for Hash CD. Both nodes are hashed again to produce the Merkle Root.

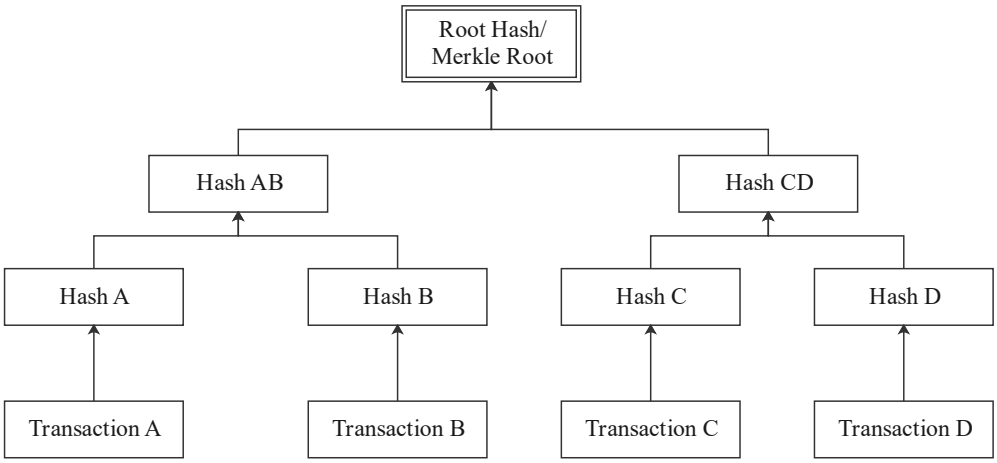


Figure 12 - Merkle Tree

The achievement of Merkle trees is that they reduce the verification complexity to a logarithmic order relative to the size of the data involved. For example, if a binary tree is used verification complexity will grow at the order of $\log_2(N)$ for a balanced tree.

A Merkle Distributed Acyclic Graph (DAG) is a graph where each node has an identifier that is a result of hashing the node's contents. Merkle DAGs can only be constructed from the leaves. Every node in a Merkle DAG is the root of a Merkle DAG itself (sub graph) and the subgraph is contained in the parent DAG. Merkle DAG nodes are immutable meaning any change in a node would alter its identifier and affect all ascendants. It is similar to the Merkle trees but there is no balance requirements and a node in this case can have several parents.

2.7.2 Relevance

In our research, we have looked into Merkle trees and similar variants to speed up the verification process of transaction hashes.

2.8 Third Party Data Integrity Audit Techniques

2.8.1 Brief

The use of a third-party auditor (TPA) provides efficiency, transparency, and the fairness in performing all the needed auditing tasks.

Using a third party to do or facilitate the audit process is not new. In their work "Auditing to Keep Online Storage Services Honest"[56], the authors go over different audit techniques. They define the role of internal and external audits.

Although the use of TPA provides several advantages, the fact that the TPA is an untrusted entity and can turn into a malicious user cannot be ignored. Techniques to ensure integrity of a TPA must also be taken into consideration.

2.9 Inter-Planetary Filesystem (IPFS)[57]–[59]

2.9.1 Brief

IPFS is a distributed file system. According to the IPFS GitHub account [57], [60], IPFS is a hypermedia distribution protocol which enables the creation of completely distributed applications. It is designed to connect all computing devices with the same system of files. IPFS is similar to a single BitTorrent swarm exchanging Git objects [61]. IPFS uses Merkle DAGs to verify the integrity of remotely stored data. IPFS allows data to be scattered in many locations and be accessed while guaranteeing its integrity.

IPFS is considered a peer-to-peer storage network. IPFS locates what is needed by its content address instead of its location. The fundamental principles behind IPFS are the identification through content addressing, DAG content linking, and content discovery through distributed hash tables.

2.9.2 Relevance

Although not targeting database applications, IPFS does the reverse of the problem at hand but for a normal filesystem. It provides content addressable remote storage of data. Performance is of great importance for IPFS. Different techniques that IPFS uses to handle data will be looked into in order to see if any of them could be of help in the efficient verification of database transactions.

2.10 Security by Design

According to the OWASP “Security By Design[18]” guidelines, the following are some of the important core pillars that contribute towards having a secure information system and how our work caters to them:

- Confidentiality: In the presented architecture, much of the integrity measures are based on exchanging hashes and signatures rather than exchanging the actual data. In fact, even the signatures and hashes themselves are never stored outside the data owner’s facility.

- Integrity: Integrity is the core goal of our work. Being able to detect any tampering that occurs to the data.
- Availability: One of the primary advantages of our proposed work is that all verification and checking is done at the same place with the data. No need to contact any external parties/servers in order to verify data integrity. This means that the architecture does not impact the availability of data access. In addition, all data is stored locally meaning that the system will not be vulnerable to any external outages that could impact data availability.

2.11 Data Governance[62]

Data governance on an organizational level focuses on guaranteeing data quality and integrity throughout the data lifecycle.

The key focus areas for data governance include:

- Availability
- Consistency
- Usability
- Integrity
- Security
- Accountability

The goal of the proposed solution is to cover the accountability, integrity, consistency and security requirements. More details on data governance strategies and lean data is provided by[63].

The pillars of data governance[62] are:

- Data Stewardship: accountability for different portions of the data
- Data Quality: correctness, completeness, accuracy, and consistency.
- Master Data Management: Organization-wide data consistency

2.12 Continuous Auditing and Realtime Assurance

The concept of continuous audit as opposed to manual/periodic audit has a number of benefits. As per D.Y. Chain[64], the following are among the primary differences/merits of continuous auditing:

- **Frequency:** Continuous audits are more frequent as opposed to periodic audits. This means that any issues will be uncovered sooner and closer to the event itself rather than later after time has passed.
- **Proactive:** Continuous audits are performed all the time (proactively) rather than waiting until there is an issue that requires an audit and do it reactively.
- **Automation:** Automatic systems take care of the audits. This results in greater efficiency and accuracy in comparison with manual audits.
- **Extent:** In continuous auditing, the whole population is checked which is more reliable in comparison to sampling some of the data to audit.
- **Testing:** Does not require humans to do the testing because they are automated.
- **Reporting:** Continuous audits are constantly checking the data which means that the reports are more frequent than traditional audits which provide periodic reports.

The solution we are presenting provides semantics that could be used as a base for performing continuous auditing. One of the design decisions that need to be evaluated is how to embed a continuous auditing process based on the proposed design without incurring a large performance overhead.

2.13 Levels of Assurance and Audit Objectives: Where do we fit?

According to[65], auditing is divided into 4 different levels summarized as follows:

1. Level 1: Transaction level
2. Level 2: Compliance level
3. Level 3: Estimates level
4. Level 4: Judgement level

Comparison: Source: [65]

Figure 13 – Levels of assurance and audit objectives

In the work presented, our target is to secure the first level (transactions level). In fact, we could think of our work as securing level zero which is the database level prior to becoming seen as a business transaction. Verifying transactions from the business standpoint is beyond the scope of our work.

3 Related Work

A number of different solutions have been proposed to target the problem we are addressing. Solutions vary in the way the problem is tackled. We have chosen to compare with work that addresses the challenge matching two criteria: a) Solutions based on our approach of cumulative serial hashing of transactions but that provide different functionality, have different performance implications, or require different levels of effort/invasiveness to embed into existing architectures. Such solutions are to be compared with our proposed solution to illustrate different tradeoffs between different proposed architectures. And b) solutions that tackle the same problem but using completely different approaches.

3.1 DRAGOON: An information accountability system for high-performance databases[17][66][67]

DRAGOON is an information accountability system that relies on continuous cryptographic hashing of transactions. DRAGOON primarily relies on an external "Digital Notarization Service" rather than just a simple external transaction signer.

The digital notarization service is responsible for storing hashes of all the transactions made.

The diagram below[17] explains the architecture of the solution proposed where the use of an external notarization service is included.

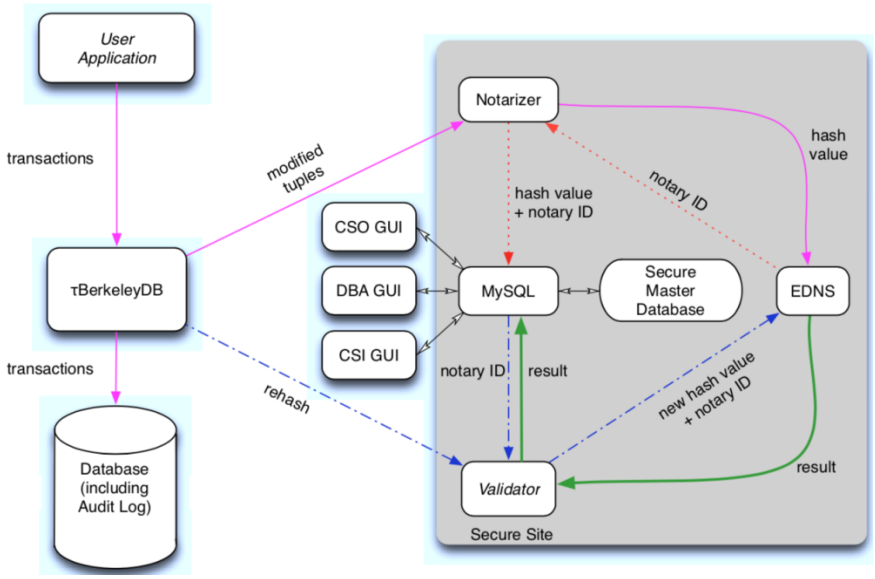


Figure 14 - DRAGOON Architecture[17]

As Figure 14 shows, when a user application performs a transaction on the database, the affected tuples are cumulatively hashed, and the hashes are sent to an external notarization service (EDNS). The notarization service signs the hashes and sends them back to the notarizer which in turn saves it into a MySQL database. The validator on the other hand, traverses all the database, hashes all the records and then contacts the notarizer for validation of the signed hashes.

3.2 Amazon QLDB (Quantum Ledger Database)

In March 2019, Amazon has announced that they are working on a blockchain database system named QLDB.[68]

QLDB solves part of the problem addressed in our work. Due to the nature of the problem we are targeting, the use-case is slightly different from that solved by Amazon.

Even though it targets a different use-case, we see the concept of QLDB as both a) An indication of the industry's need for integrity-verifiable ledger databases and b) Work that could be looked at to see how similar issues are tackled and for future enhancements when including updates and deletes.

QLDB provides the ledger database service based on the premise that there is a "central" and "trusted" authority which in this case is Amazon. Amazon in this case provides the signing and trust service as well as the hosting of the actual data. Which is exactly the model we are trying to avoid and solve. Having both the storage of the data as well as the verifiability of its integrity in the hands of the same party.

3.3 BigchainDB[69]

3.3.1 Brief

BigchainDB is a distributed database designed to leverage a blockchain network. BigchainDB provides the following features:

- Decentralized control
- Immutable transactions
- Database-like semantics while relying on a blockchain backend
- Ability to query structured data
- High transaction throughput
- Low latency

BigchainDB is byzantine fault tolerant with up to one third of the nodes failing gracefully. It does so by leveraging Tendermint[69][70].

Tindermint is a language/technology agnostic framework for distributing databases. It relies of bezantine-fault-tolerance (BFT) in addition to a blockchain implementation.

The objective of BigchainDB is to provide the following features in comparison to traditional database systems and to blockchain networks.

	Typical Blockchain	Typical Distributed Database	BigchainDB
Decentralization	✓		✓
Byzantine Fault Tolerance	✓		✓
Immutability	✓		✓
Owner-Controlled Assets	✓		✓
High Transaction Rate		✓	✓
Low Latency		✓	✓
Indexing & Querying of Structured Data		✓	✓

Figure 15- BigchainDB Value Proposition (SRC: BigchainDB Whitepaper)[69]

3.3.2 Network Layout

BigchainDB utilizes different components to achieve the features required.

- A fully replicated mongoDB database exists at each node
- Tendermint is utilized for inter-node communication

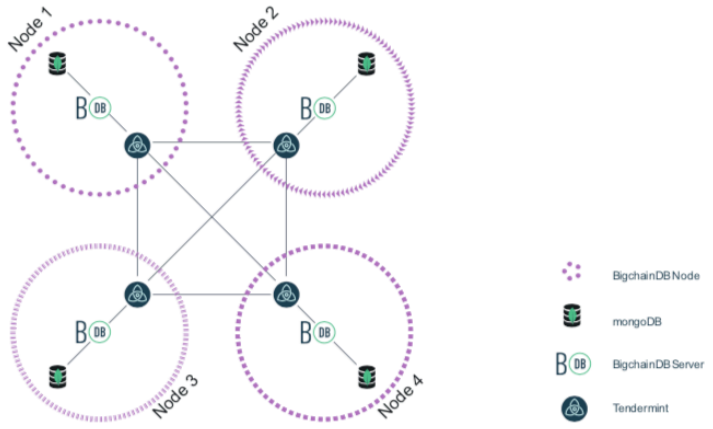


Figure 16 - BigchainDB Network Layout (SRC: BigchainDB Whitepaper)[69]

This research has proposed creating a tamper-evident model for databases. It has covered CRUD operations as well as complex databases including aggregate operations. The authors of the paper have designed a model for complex hierarchical structures as well as database operations that follow a DAG structure.

3.4 The Case of The Fake Picasso[71]

The authors of the paper provide a good model for provenance tracking. Provenance is the ownership of an item and where it is located. Provenance tracking is keeping record of the object as it exchanges hands. The focus of the research however is on documents rather than database transactions. The aim of the research is to provide a mechanism to track the "creation, ownership, and workflow of documents and to ensure a certain level of trust".

Typical applications for the proposed solution are financial and regulatory compliance records, DNA and lab results, and tests of food and food additives by organizations like the US FDA. Medical diagnosis and tracking of medical mal-practices across the work of different doctors and medical institutes is also an application.

One special merit of this work is being able to track documents as they cross organizational boundaries and move across different organizations while maintaining all their provenance data.

A number of different techniques of the research are interesting for us. For example their approach towards managing updates might open the door to having better support for updatable databases that utilize our presented architecture.

3.5 Provenance-Aware Storage Systems

The research work targets quite a similar problem but on the filesystem level instead of database transactions. The goal of the research is to track the lifecycle and "complete history" of a piece of data (i.e.: file or document) throughout its lifetime.[71]

One of the goals of the referenced research is to store provenance data inside the filesystem itself rather than using a separate external database to store it.

Provenance data is stored inside the filesystem in the form of metadata coherently that is more convenience during data backups, archiving, and transfers.

Another goal of the system is to embed the provenance features into the filesystem and provide them transparently to the filesystem user. An advantage of this is that all applications will work seamlessly and will not need to have application level support for provenance.

3.5.1 Similar Ideas:

Although it targets a different problem, we could benefit from the work to see approaches to provide seamlessness and application independent capturing and verification of provenance information.

3.6 A Simple Model of Separation of Duty for Access Control Models[10]

The paper targets Separation of Duty as a method for protecting against insider malicious behavior. The paper provides a formal model for a simple form of SoD. The proposed mode is integrated into RBAC (Role Based Access Control) of systems.

Separation of duty is done during user permission and authorization.

The paper also tackles the problem of detection of malicious / erroneous actions by insiders of the organization. Seamless integration is a common target of this research and us.

The paper however tackles the problem by integrating the SoD into the business workflow which is complete different from our approach which is much more seamless and business-workflow-agnostic.

3.7 Oracle Label Security[72]

Oracle label security targets a different class of problem being tackled which is data access in general rather than just tamper awareness. The challenge it tries to solve is maintaining data security/confidentiality of data being aggregated from multiple sources.

This is done by attaching labels to data. Such labels are used to classify data and then are later used for access control permission.

Where this solution fails short is that it assumes the existence of a trusted administrator who is responsible for the whole system. And this is exactly the problem we are solving.

3.8 Tapestry: Continuous Queries of Append-only Databases[73]

Although this work is completely different from the problem we are targeting, the concept of a continuous query is of much use to our proposed system.

Continuous queries are queries that are running continuously. They automatically take into consideration any data added to the database. And when the query result changes, a notification is given to a system.

We see this as possibly of value for the continuous and efficient verification of data and the automatic detection of any tampering live and at the time of the malicious action itself rather than waiting for a verification cycle to run.

As a future work, continuous query systems could be adapted to include tamper detection utilizing the DBKnot framework.

3.9 Paper: "Designing Better Filesystems Around Tags, not Hierarchies[74]"

Although it is not an academic research paper and is only indirectly addresses the problem at hand, we have decided we must include it because it has an excellent comparison between the feature usability of different file organization systems as well as a comparison with revision control systems (GIT and SVN) and content addressable systems (like IPFS) It also tackles the concept of versioning, file uniqueness, and tampering which is the problem at hand.

3.10 A Robust and Tamperproof Watermarking for Data Integrity in Relational Databases

This work makes databases tamper evident by adding checksums to individual database records and then a checksum for the whole database. Database is partitioned in order to make it difficult to guess how the aggregated checksum is generated[76]. The goal of the research is to watermark the whole database which is shipped and allow for re-checking the database to identify tampering. The target of this work is the buying and selling of whole databases rather than transactional databases.

The goal of watermarking the database is to use it as a proof of ownership, prove the authenticity of the data and detect any tampering that might occur. The author refers to steganography when it comes to hiding the watermark inside the data. Such "hiding" is done by choosing an unknown partitioning field for calculating the total checksum of the database.

3.11 Implementing a Tamper-Evident Database System

The goal of this research is also to make database systems tamper-evident. The goal is to protect databases from tampering either by malicious insiders or external players or from taking advantages of security loopholes in the database engine or the operating system.

Such measures are implemented without touching the database management system. They are implemented through a middleware that is placed above the database engine. Performance of the database query operation is 4 to 7 times slower[77] and the inserts are 8 to 11 times slower.

They consider such a performance difference as a dramatic improvement over using a tuple-level digital signatures. Such an improvement is achieved by using hierarchical structure to keep authenticity information.

The research uses hash trees that are similar to Merkle trees in order to make checking the validity more efficient. They use an adaptation of a structure called “interval trees” to store intervals of values rather than single values.[77]

3.12 Immutable Operating Systems

Recently there has been a general direction towards the implementation of operating systems in an immutable form[78]. Once installed, an operating system is never changed. There are two different components to immutability in operating systems. The first component is to design an operating system with inherent immutability (e.g.: Fedora Silverblue)[78], the second component is to change the installation process, and to containerize all applications to sandbox them from the actual operating system. Any additional packages, configurations, and tweaks are overlaid above the operating system in the form of independent container-like pieces each of them with its own required libraries and dependencies. This approach contributes to the stability of the operating system against breaking due to unintentional changes in system configuration and libraries as well as security against the addition of any malicious content. Any update should be atomic and should be easy to rollback without impacting the underlying operating system. As a matter of fact, in some cases, even operating systems upgrades to different versions are atomic operations that could be rolled back if needed.

Fedora “Flatpacks”[79], Ubuntu “Snaps”[80], [81] and “OSTree”[82] are examples of immutability in filesystems.

While application containerization like “Snaps” promises application portability, ease of use, and relative stability of the operating system, there are multiple controversies about them. Criticism includes: a) drifting away from standard opensource distribution model into proprietary distribution mechanisms, significantly much bigger storage requirements (in some cases can reach 100X for very small applications, as well as a security model that does not make the sandboxing as clean as perceived by users[80], [81], [83].

Snaps and flatpacks introduce the concept of transparent immutability and tamper protection to operating systems. We one other hand aim to provide tamper detection to database systems and achieve similar transparency.

3.13 Forensix: A Robust, High-Performance Reconstruction System[84]

Forensix performs a similar function with two primary differences:

1) Forensix is system-level and is embedded into the kernel rather than into the database. Forensix monitors all system calls performed by users.
2) A log of the actual transactions rather than the hashes is kept into a database. This log helps in retrieving compromised data.
3) Activities are logged on a per-session basis where all activities per user could be examined.
4) All data is stored locally meaning that an insider could tamper with the data and erase all trails leading to the malicious actions.

Similar systems that cater to the same problem in different approaches and that work on a system level and utilizes logging and reconstruction are [85]

3.13.1 ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay [85]

ReVirt is designed to log and track all changes inside a virtual machine. The purpose of the logging is to allow replay of all instructions and putting the whole VM into any previous state desired. As mentioned, ReVirt applies to a whole operating system instead of just a database. Additionally, the amount of data generated from ReVirt to be able to reconstruct a full operating system is relative large compared to a live transactional database system. Therefore, ReVirt is not applicable in our targeted use-case.

3.14 Remote Verification Techniques for Data Integrity

3.14.1 Brief

A number of different research work has been done on the remote verification of data integrity[6], [86], [87], [88]. Such work however focuses on the verification of data stored at remote third parties rather than local data. These techniques are not applicable to our case because of the following reasons:

- Data is to be stored locally rather than remotely.
- Potential malicious tampering could be done by internal people who have access to the data as well as all stored hashes.
- Performance will not be suitable in a database use-case.

3.15 Leveraging AI to Detect Fraudulent Transactions

AI is one of the techniques that are starting to be used to identify fraudulent transactions. For example, according to HBR[89], the Royal Bank of Scotland has prevented to loss of $9M in fraudulent transactions using AI. And PayPal has reduced their false alerts on transactions to a half by employing AI too.

3.16 Comparison With Other Related Work

Work	
DBKnot	+ Seamless integration with zero or little rework + Externalized signer for greater security + Multi-level support (DB, ORM, Web-service) + Minimal performance overhead (target) - Currently Append-only (updates, deletes need to be researched) - Tamper detection only (no data retrieval)

Dragoon[17]	+ Tampered data completely recoverable - Cost of duplicate storage - Cost of data transfer (bandwidth + speed) - Data stored outside main system
Amazon QLDB[68]	+ Supported by amazon + Serverless (SaaS) - Data hosted externally - A new and different database (For new applications) - No seamless integration
BigchainDB[69]	- No seamless integration into existing systems - Data is transferred to third party - Transfer overhead - Confidentiality concerns - Reliance of external third-party for data storage which is outside our control
The Case of the Fake Picasso	+ Provenance tracking of creation, ownership, and workflow + Track all stakeholders who touched the asset + Inter-organization tracking + Handles and tracks updates - Designed especially for tracking documents rather than database transactions which is of a completely different nature in terms of frequency and timeliness requirement.
Model for Provenance and Tamper Detection [71]	+ Manages updates - No seamless integration into existing systems - Focus on documents and aggregate values
Tamper Evident Provenance [75]	+ Full CRUD support - Different database - No seamless integration into existing systems

Separation of Duty	+ Tracks changes in user authorization and permissions - Assumes local administrator is not malicious
Oracle Label Security	+ Manages data aggregated from multiple sources + attaches tags to use for finer grain access control - Assumes local administrator is not malicious
Tapestry: Continuous Queries	+ Targets keeping a constantly updated summary state + summary is kept up-to-date - Not for tracking malicious users
Better File Organization Around Tags[74]	+ Excellent support for versioning + Deduplication support + Tag based filesystem + Based on git - Targeting files rather than database
A Robust and Tamperproof Watermarking for Data Integrity in Relational Databases	- Target is tamper detection in whole databases that are bought and sold rather than in a transactional manner. Goal is preserving integrity during buying and selling of databases.
Implementing a Tamper-Evident Database System	+ Detects malicious tampering by insiders + Uses hierarchical structures for performance. Similar to Merkle trees. - Protects blocks of data - Does not rely on an external signer
Immutable Operating Systems	+Protects OS from malicious changes +OS image is signed +Limited access to OS by applications or administrator -Limited to operating systems and not for transactional databases.

Forensix	+ Session based logging allowing examining on a per-user level - Verification data is stored locally which allows an insider to tamper with data and erase all trails if they have access.
ReVirt	- Focused on system level intrusion detection - Works primarily on a virtual machine level rather than on a database level

3.17 Summary of Related Work Comparison

By looking at the related work, the primary gaps that our solution fills are:

- **Trust of an Insider:** Many of the solutions provide measures to protect or detect data tampering on an application level or on a database level with all requirements present inhouse and within the control of the internal DBA team. This comes with the implicit assumption that the internal top-most system administrators with highest level of access to systems and databases are fully trusted and cannot be malicious or even collude to tamper with data. Our goal is, while maintaining the highest level of privilege to internal database admins, we still provide a tamper-evident mechanism.
- **Trust of Third Party:** Some of the commercial solutions provided (Amazon QLDB) assume the organization trusts the third party with protecting its data. Our solution eliminates the need for this trust.
- **No Data Transfer:** Some of the solutions resort to providing an external verifiable copy of the data. This adds some complications like a) confidentiality of data at third party and in transit, b) performance penalty of transferring all data. We totally eliminate the need for transferring an organization's data and keep it completely inhouse.
- **Database:** Some of the solutions protect other objects than databases. For example, documents, filesystem, or even entire operating systems. Our goal is transactional databases.
- **Transactional:** Some of the solutions do protect data but cater more to a batch processing model rather than live transactional systems, we cover the transactional component.
- **Database Specific:** Some of the work provides solutions that have to be implemented in a database specific setup. Even though we have this approach among one of our

solutions but we also provide two other alternatives that are completely database agnostic.

- **Transparent:** Some of the solutions are not transparent and require modifications at the application level in order to function. We provide a solution that is as seamless as possible and that requires zero or very little modifications on the application level. Modifications required at the database level or at the middle-ware level are minor ones that are add-on configurations rather than being invasive. Our goal has been to design a solution that could be transparently retrofitted into existing systems with a) non-invasive approach, and b) empowers old and currently existing systems as well

Our goal has been to address the abovementioned gaps as much as possible. The reason we have chosen the gaps identified above is that they are vital in order for any solution to be applicable in existing real industrial use-cases rather than just propose a solution that stops only at the theoretical level and falls short of being suitable for solving real-life scenarios. Our goal is to provide a solution that does not impractically require total change in an underlying infrastructure.

4 Proposed Solution

4.1 Proposed Solution overview

In this section we cover the basics of DBKnot and the different ways of implementations that will be described in more details throughout the document.

4.1.1 Solution Brief

In our presented solution we build a transparent and seamless middleware for securing database transactions against possible tampering by individuals who have full administrative access to the database and all its related infrastructure. The way this is to be achieved is by leveraging some features of the technology similar to blockchain to interweave sequences of transactions in an unbreakable chain. This is to be done by generating a unique hash for each transaction and using it in a chain of transactions. Any attempts to modify previously entered data will break the hash and therefore the sequence of transactions following such transaction will be invalidated.

In order to guarantee that such a chain could not be re-generated following any tampering attempt, an external source is used for time-stamped signing of hashes. The external timestamp signer is external to the entity so it is beyond the reach of any internal system administrator. Another alternative could be a physical Hardware Security Module (HSM).

The following integration techniques in Figure 17 - Integration Options are assessed:

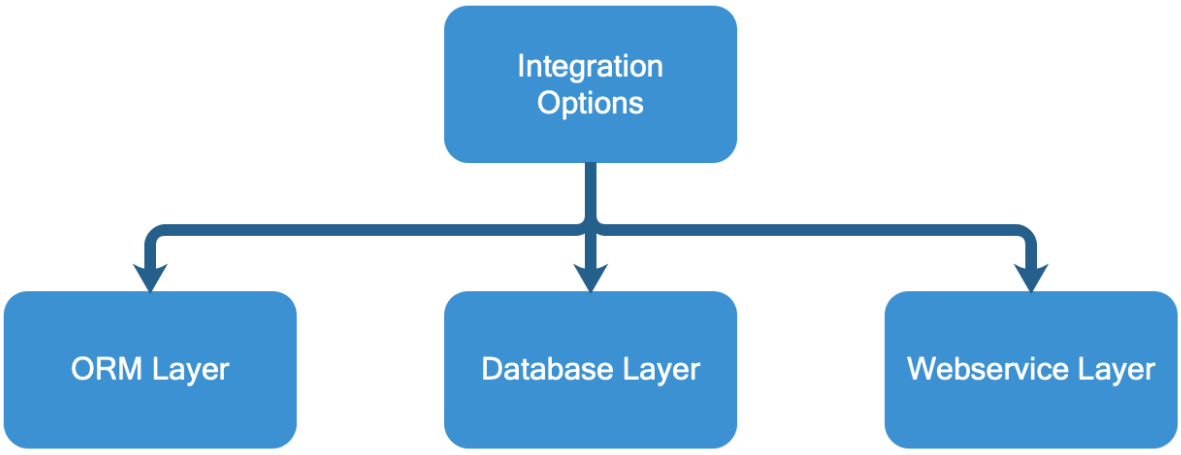


Figure 17 - Integration Options

4.1.2 ORM Layer

Many modern complex application development frameworks provide an ORM (Object Relational Mapping) framework that assists developers by simplifying development semantics and making them conform to their object oriented development paradigm. It also helps them leverage all the capabilities of the programming language they are using. One implementation technique is to embed the DBKnot support into existing ORM layers. Such support empowers developers to include DBKnot features into their database models using a simple declarative notation without having to go through any implementation details. The idea here is to embed the DBKnot functionality inside the ORM layer itself and provide a totally transparent and seamless experience to application developers that requires near-zero changes to their application code.

In addition to the declarative semantics and ease of use by developers, embedding tamper-detection layer inside the ORM layer also makes it completely database agnostic. Meaning that the same implementation will work on any database as long as it is supported by the used ORM layer without any change at all.

4.1.3 Database Layer

In this design, the DBKnot features are implemented as a set of scripts/macros/triggers/etc. on the level of the database directly. The goal of this approach is that it will also be totally seamless. In addition, being close to the database layer makes it perform with less overhead.

4.1.4 Web Reverse Proxy - Microservices Architecture

Similar to Alternative 1, the DBKnot functionality could be implemented inside a middleware. The benefit of injecting the functionality in the form of a middleware is that it could allow the functionality to be retrofitted into existing applications with doing zero changes to the existing application. This way existing applications can benefit from DBKnot and secure their data seamlessly.

This approach is better suited to cater to applications with microservice architectures.

Challenge: This will require an easy-to-use mini language/syntax for application developers to define their application web-service's semantics

Advantage: Totally non-invasive, could be totally external to server inside a reverse proxy.

4.1.5 Preliminary Comparison

The following is an our perspective of the complexity of implementation and usage.

Criteria/Integration	ORM Layer	Database Engine Layer	Web Service Layer
Implementation	Simple	Complex	More Complex
Ease of Use	Should be easy	Should be easy	Should be easy
Seamlessness	Easy	Complex	More Complex
Usage	Simple Declarative	Simple DBMS Configuration	Reverse-Proxy Rollout

4.1.6 Assumptions

The presented architecture is designed while making some assumptions on the nature of data/system being managed.

- **Immutability:** Data tables tracked by DBKnot is immutable. As mentioned above, there is a multitude of applications where data is only added and never changed. Accounting and banking transactions, logs, etc.
- **Different Records:** The system assumes that there are no 2 exact duplicate records within any of the tracked tables. At least a key should be there to differentiate. This is due to the fact that DBKnot doesn't touch existing table structures and does not add any keys. Therefore, database records need to be uniquely "content addressable". This should not be much of a concern because most of such data, even in the extreme unlikely case of exact duplicate records existing, will at least have a date or an autogenerated primary key.
- **External Signer:** DBKnot depends on the fact that an external signer or signing-authority is external to the organization and is not within the reach of any insiders. This is to reduce the probability of collusion on data tampering to commit internal fraud.

4.2 Introduction of an External Third Party Signer

The solution relies on the introduction of a third-party signing authority. The third-party is an external entity that is outside the reach of organization insiders and thus reduces and ideally eliminates the possibility of collusion among internal and external stakeholders.

We introduced in Figure 18, an independent signer + time-stamper service (in red). The signer/time-stamper is a totally external entity that could even be outside the organization. The signer service could cater to different organizations as illustrated in the diagram.

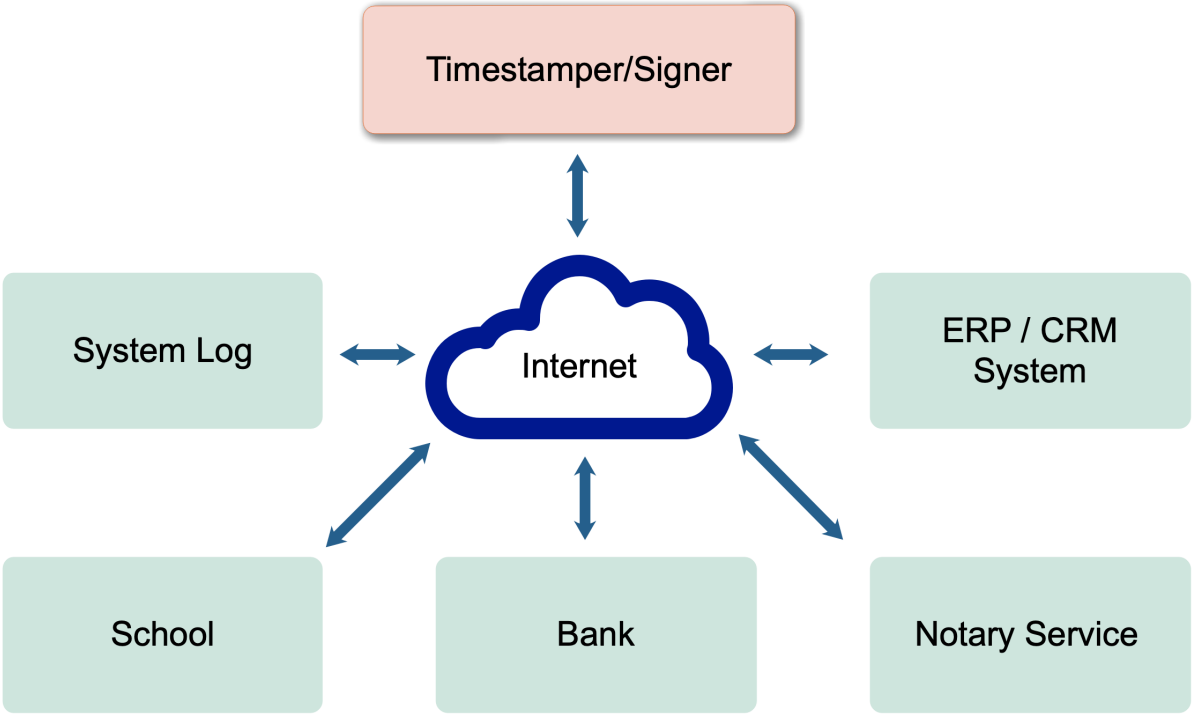


Figure 18 - Introduction of a Third-Party Signing Service

Figure 19 shows how users, databases, and applications inside an organizations work normally without any major architectural change. The only difference is that they are connected to the external signing service. In Figure 18 - Introduction of a Third-Party Signing Service, in organization “N” at the bottom left, a malicious insider can be present and despite

full access to all organization's data, they will not be able to tamper with the data without breaking the signature chain signed by the primary authority and thus, be detectable.

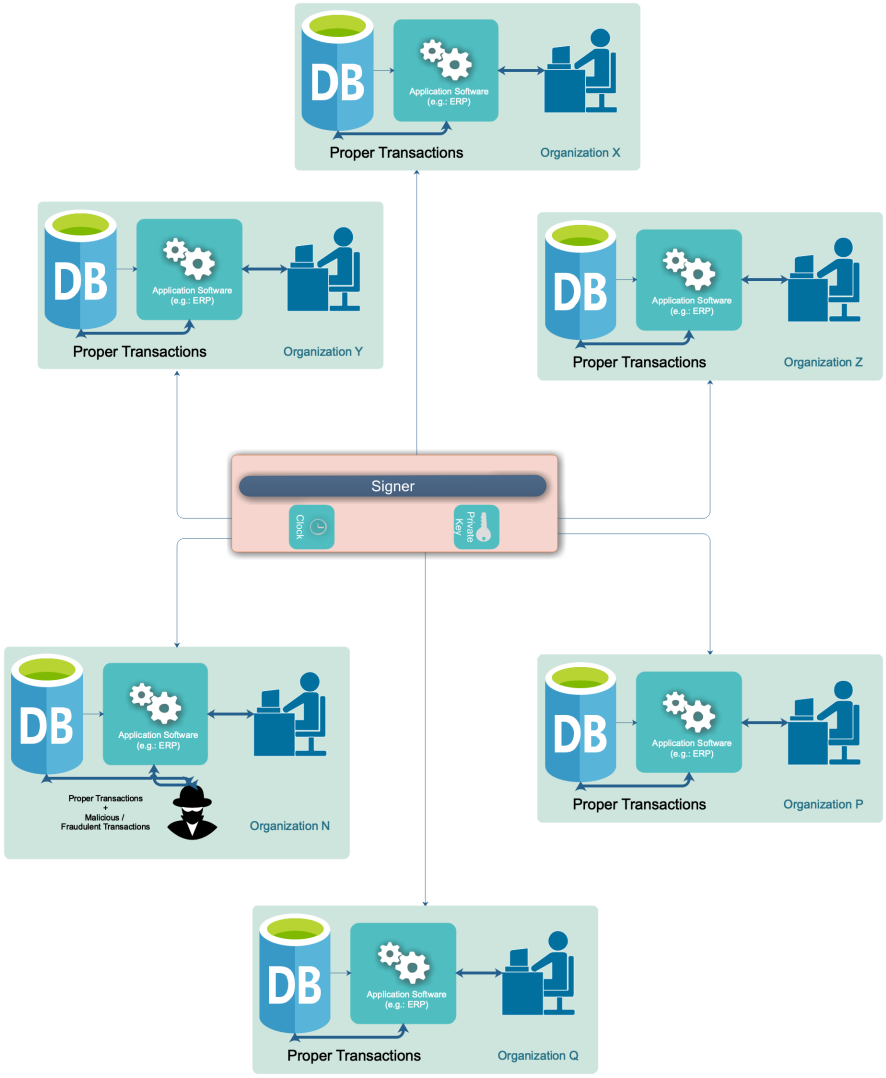


Figure 19 – Detailed Introduction of a third-party external signing authority

4.3 Stateless Signer

In addition to being an external entity, the signer is designed to operate in a completely stateless manner. DBKnot does not rely on the signer keeping any information regarding the data being signed or its corresponding hashes. Such statelessness makes the following possible:

1- **Less Vulnerable:** The signer implementation is very simple. It is made up of only a private-key, signing algorithm, in addition to a clock. The simplicity of implementation reduces

any possible attack-vectors which makes the signer easier to secure against any possible vulnerabilities.

2- **No Storage – Confidentiality:** No storage is needed on the signer end which adds to security and privacy. This provides zero-knowledge securing of the data since it only acts as a signer and not as a repository or secondary storage service.

3- **No Data Transferred:** Actual data never leaves the premises of the user. Alternatively, only a hash is exchanged for the signing process. This reduces a) The network traffic and overhead due to data transfer, b) Vulnerability of data in-transit to both exposure as well as tampering, and c) Having to trust the external signing party on all organization's data.

4- **Workload Balancing:** Statelessness makes it possible to balance load across as many signer nodes as needed as long as their clocks are well synced. This makes it easy to scale the signing service by adding more servers and distributing the workload among those servers.

5- **Multi-Site Failover:** Statelessness also allows signers to be rolled-out at multiple different sites. This provides added reliability in the case of a failure of a whole site due to a total internet outage or a blackout in the hosted area/country

6- **Proximity:** Statelessness allows servers to be distributed in a way that increase proximity to the users of the servers. This reduces signing latency and duration cost of network delays. This approach is commonly used by Content Distribution Networks (CDNs)

4.4 Methods Used – ORM Level Integration

4.4.1 Simplistic Usage

The integration layer is designed to provide a completely seamless user experience to developers. In the current implementation, as illustrated in Figure 20, all a user (developer) needs to do is to have his/her model classes extend a class (a mixin) that provides all needed functionality.

```
class Test(DBKnotMixin):
    name=models.CharField("Name",max_length=50)
    def __str__(self):
        return self.name
```

Figure 20 - ORM - Simple Mixin Implementation

Another approach is to modify the ORM itself to include the functionality. The advantage of such an approach is that existing code will run as-is without any change at all. Without even declaratively attaching to a DBKnot tracker.

4.4.2 Standard ORM Operations

Object Relational Mapping (ORM) frameworks[90]–[93] sit between developer applications and databases. They provide developers with full object oriented semantics to the database.

ORM frameworks allow system developers to use object oriented design to model their data without having to worry about how this maps to the database. ORM frameworks in turn take care of the mapping between data objects on one hand, and tables and relations on the other hand.

At design phase, ORM layer is responsible for generating the Data Definition Language (DDL) necessary to create the required tables. In SQL these are SQL INSERT statements. The ORM takes care of choosing the necessary dialect of the underlying database by utilizing different "drivers" for different databases.

ORM layers are also responsible for maintaining the consistency of the mapping throughout the development cycle by propagating any changes done to the model to reflect immediately into the database structure while preserving all data. This is a process that some call it "migration".

After the mapping is done, and during runtime, the ORM layer implements all OOP Create, Retrieve, Update, and Delete (CRUD) operations by mapping them to their corresponding Data Manipulation Language (DML) statements. In SQL, this is done by using INSERT, SELECT, UPDATE, and DELETE SQL statements respectively. As done in the DDL, all DML statements are generated by the ORM driver that corresponds to the database being used which in turn ensures that the necessary SQL flavor is used.

Figure 21 - Standard ORM Operations shows how the ORM layer sits between the developer code and the database itself and abstracts away all of the DBMS specific relational database operations.

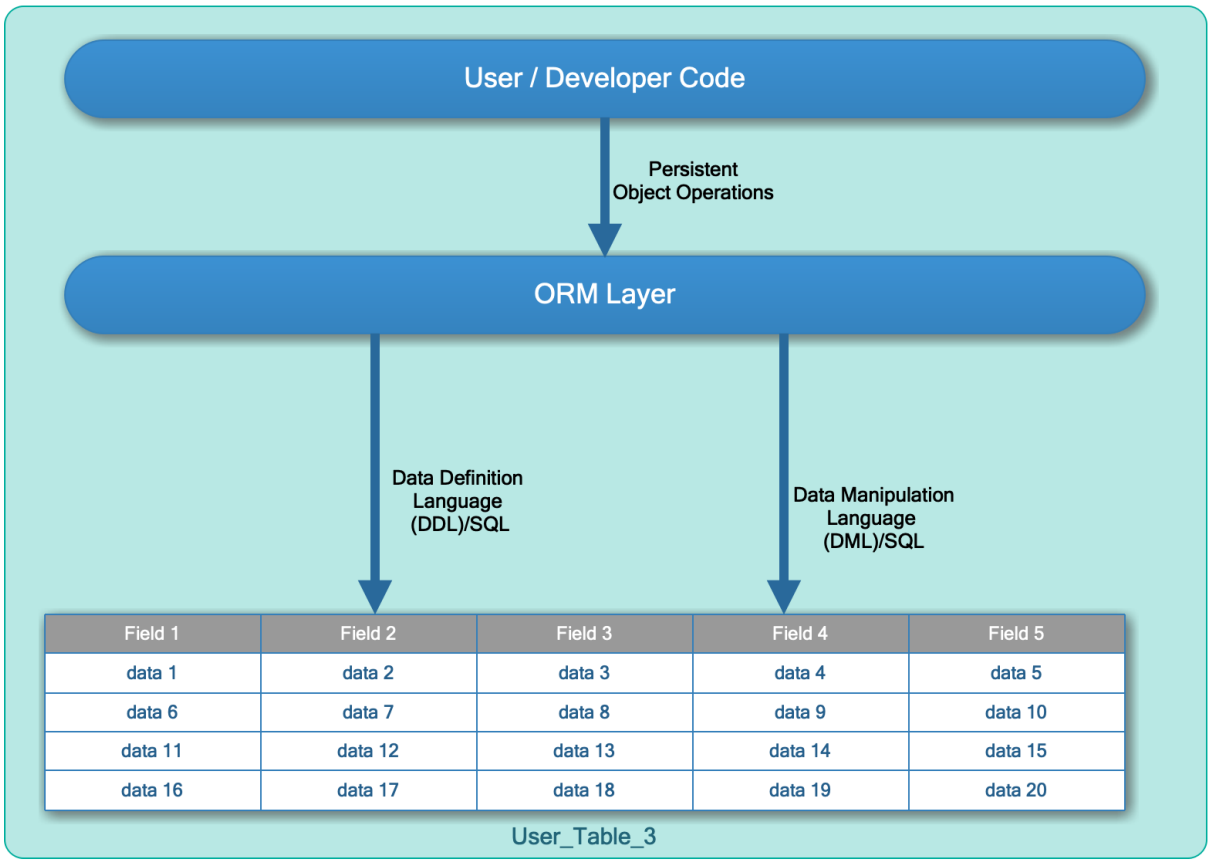


Figure 21 - Standard ORM Operations

The activity diagram in Figure 22- ORM Default illustrates the default functionality for an object-relational-mapping framework (ORM). The user application uses the ORM to perform operations of objects and properties. The ORM in turn, takes these operations and converts them to their corresponding SQL statements of the database-engine being used. The database engine afterwards performs the required operation directly on the database.

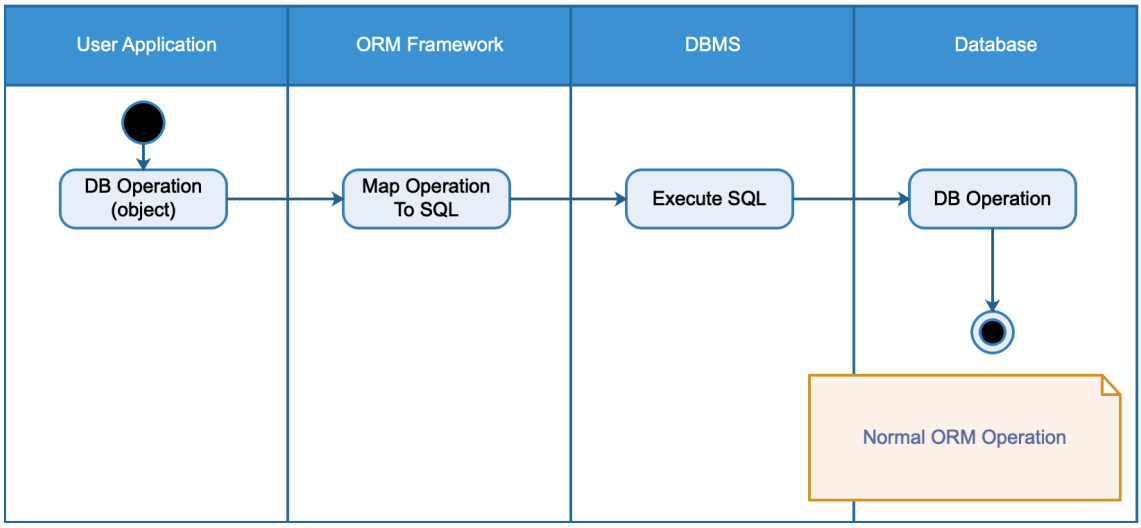


Figure 22- ORM Default Behavior – Activity Diagram

4.4.3 The Signer

The direction adopted is to introduce an externalized time-stamper/signer and/or a tamper-resistant HSM (Hardware Security Module). The role of the signer is to sign a hash of each record/transaction that gets added to the database. In addition to the record, a hash of the previous record is added. A timestamp is also added to the signed data in order to protect against future signing-replay attacks.

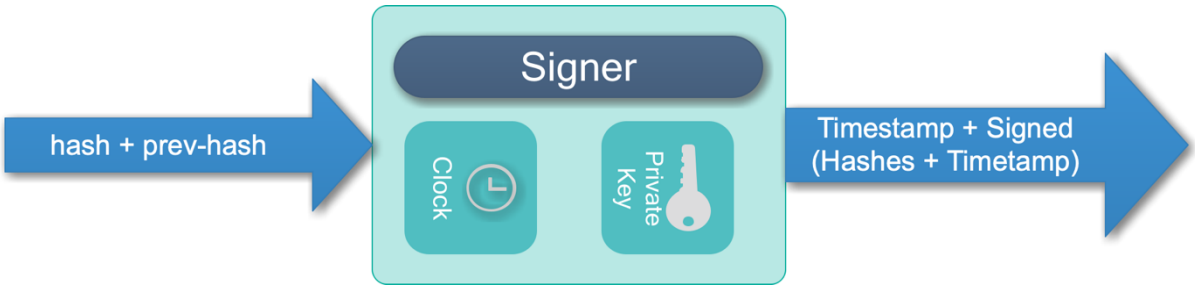


Figure 23 - Signer Service

4.4.4 Inserting the ORM Overlay / Interceptor

In order to implement DBKnot functionality into an ORM layer, an interceptor service is implemented (illustrated in Figure 21 - Standard ORM Operations). The role of such an interceptor layer is to capture all INSERT operations and ensure that all housekeeping (hashing + signing) actions are done properly. This layer is embedded in the “save” operation of the ORM models.

Figure 24 - ORM Interceptor Shows how the ORM layer fits in the complete picture with the signer in place.

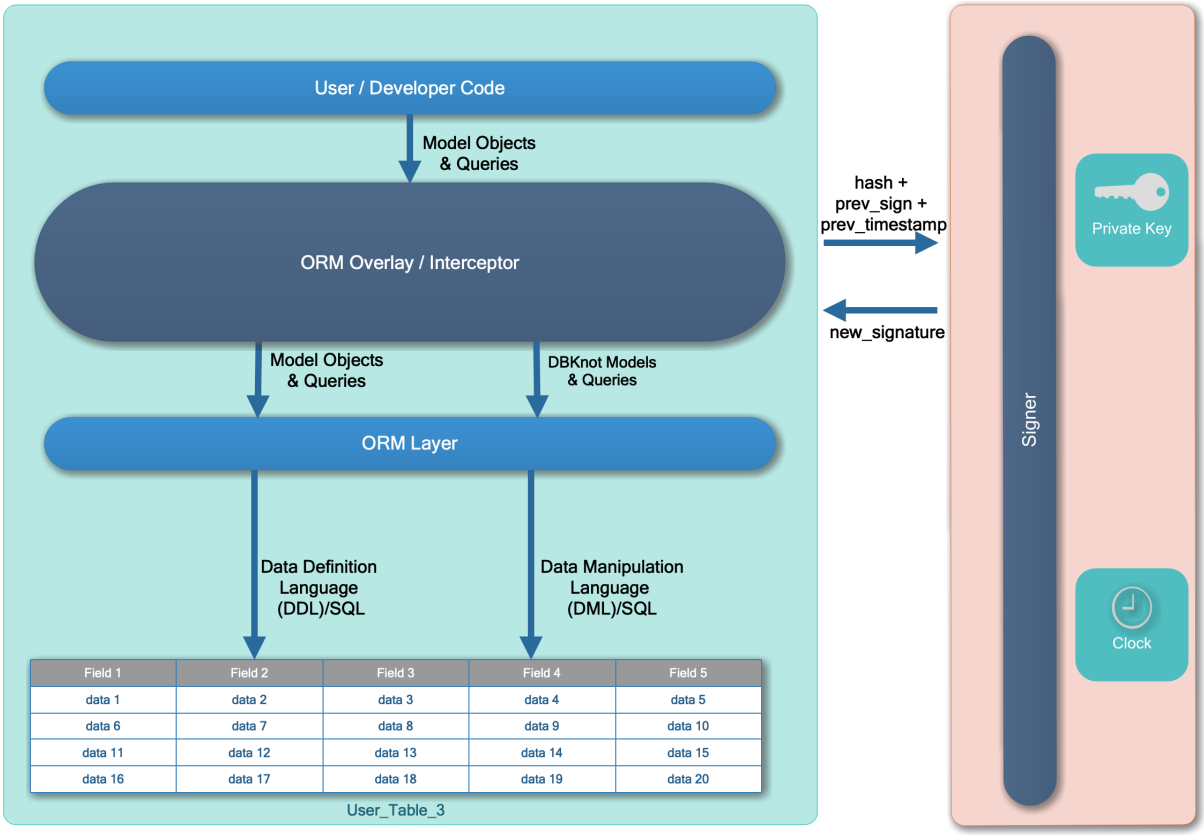


Figure 24 - ORM Interceptor

As Figure 24 - ORM Interceptor shows, the ORM interceptor sits right below the user code masquerading as the original ORM layer and therefore requiring not changes in the software itself.

As the user code initiates any persistent database operations (insert operations) that are tagged as trackable, the ORM interceptor takes the transaction, passes it to the original ORM layer which takes care of the transaction as normally expected. Afterwards, the ORM interceptor starts doing its own hashing and signing actions by hashing the record and adding it into a local hash table and then communicating with an external signer to sign the transaction and save the signed hash linked with the previous hash.

Figure 25 shows how the DBKnot hook is inserted in the middle of the operation. DBKnot intercepts all calls to the ORM, performs the needed hashing and signing functionality, and passes execution to the original ORM framework.

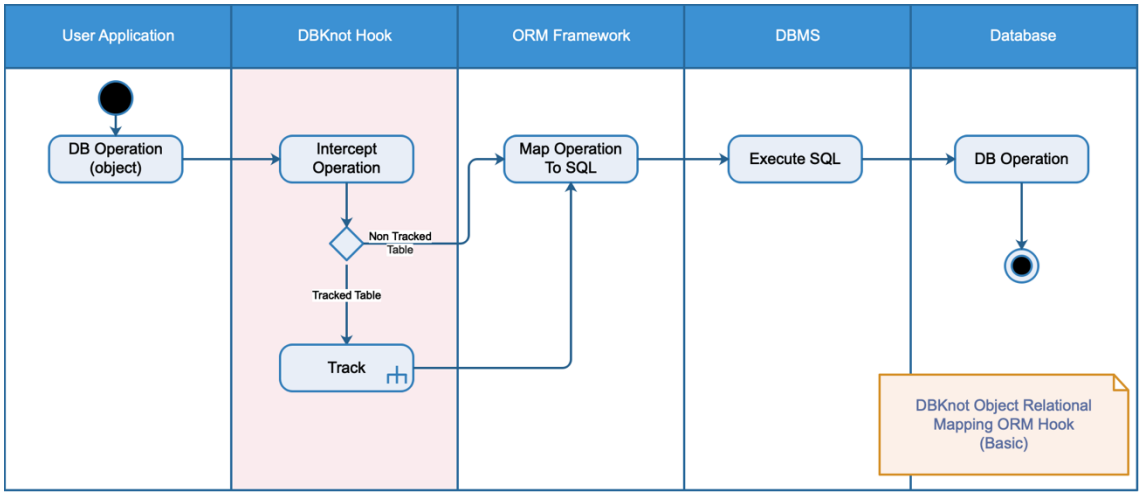


Figure 25 - Adding DBKnot ORM Hook – Basic - Activity Diagram

Because the tracking process (hashing and signing) is not a dependency for the database operation to complete, the tracking process could be called asynchronously as per Figure 26.

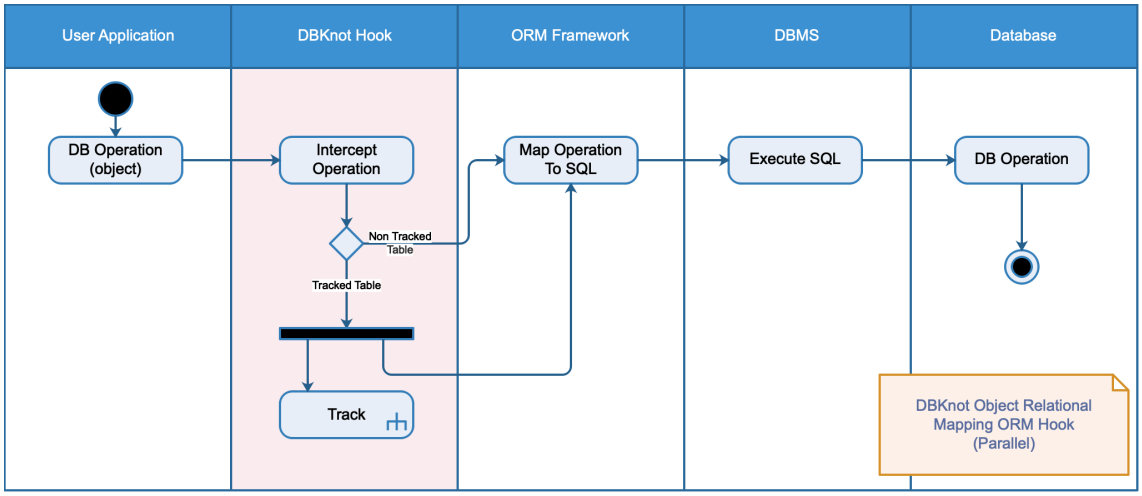


Figure 26 – Adding DBKnot ORM hook - Parallel - Activity Diagram

When a transaction is detected, a hash is calculated for the transaction and then a signer is called to calculate the corresponding signature value as per Figure 27.

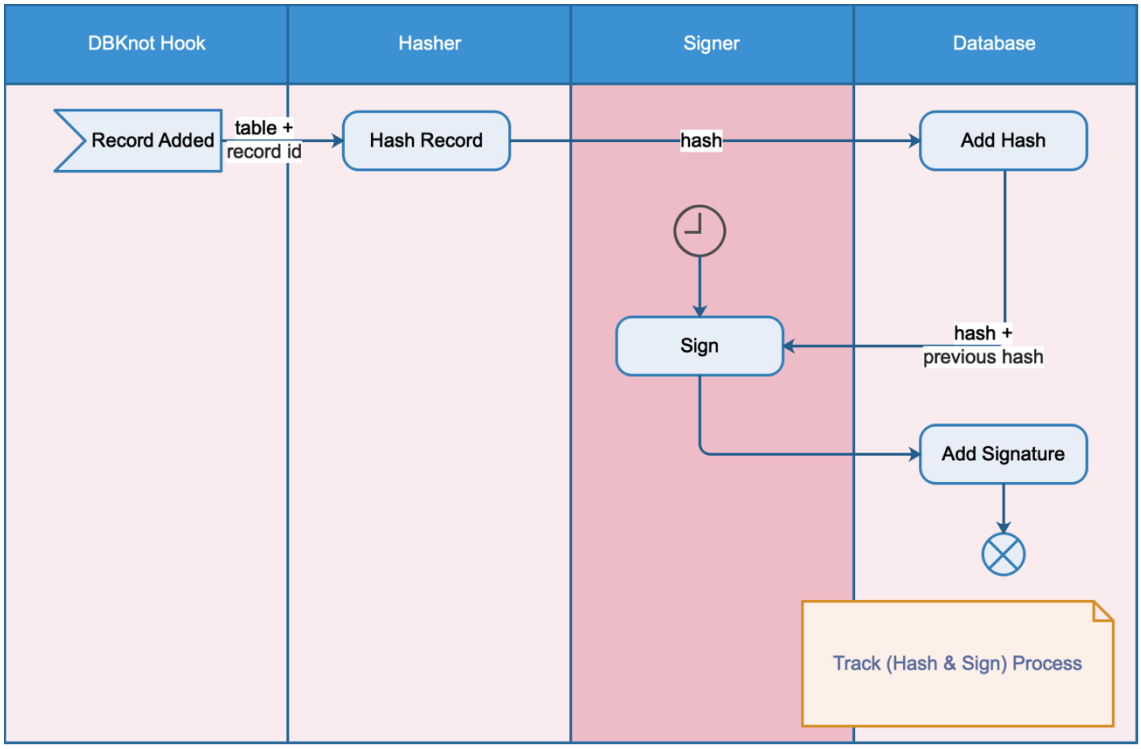


Figure 27 - Tracking (Hashing & Signing Process)

4.5 Methods Used – Database Level Integration

DBKnot also supports database-level integration. This is done by embedding triggers on tracked tables. When a record is inserted in a tracked table, a the trigger will be fired and will perform all the needed tracking functionality.

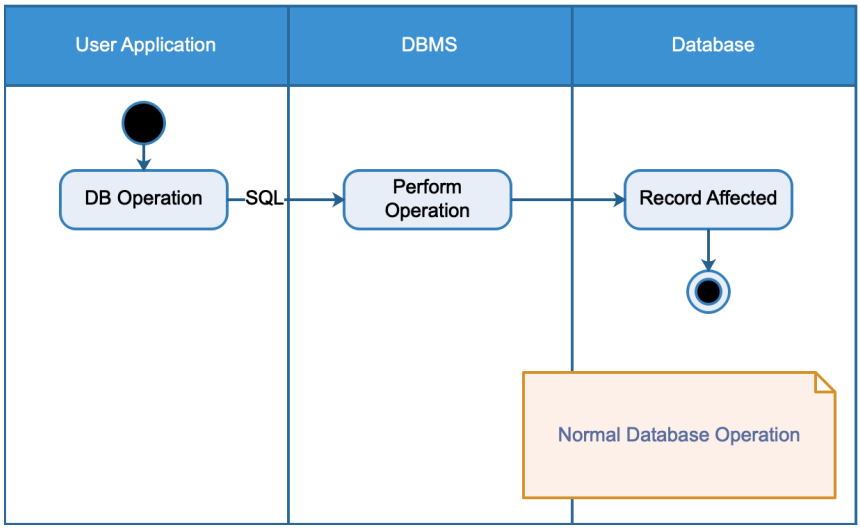


Figure 28 - Normal Database Operation

The default behavior in Figure 28 is changed by adding the DBKnot layer. The DBKnot layer is called via a database trigger that tracks desired tables.

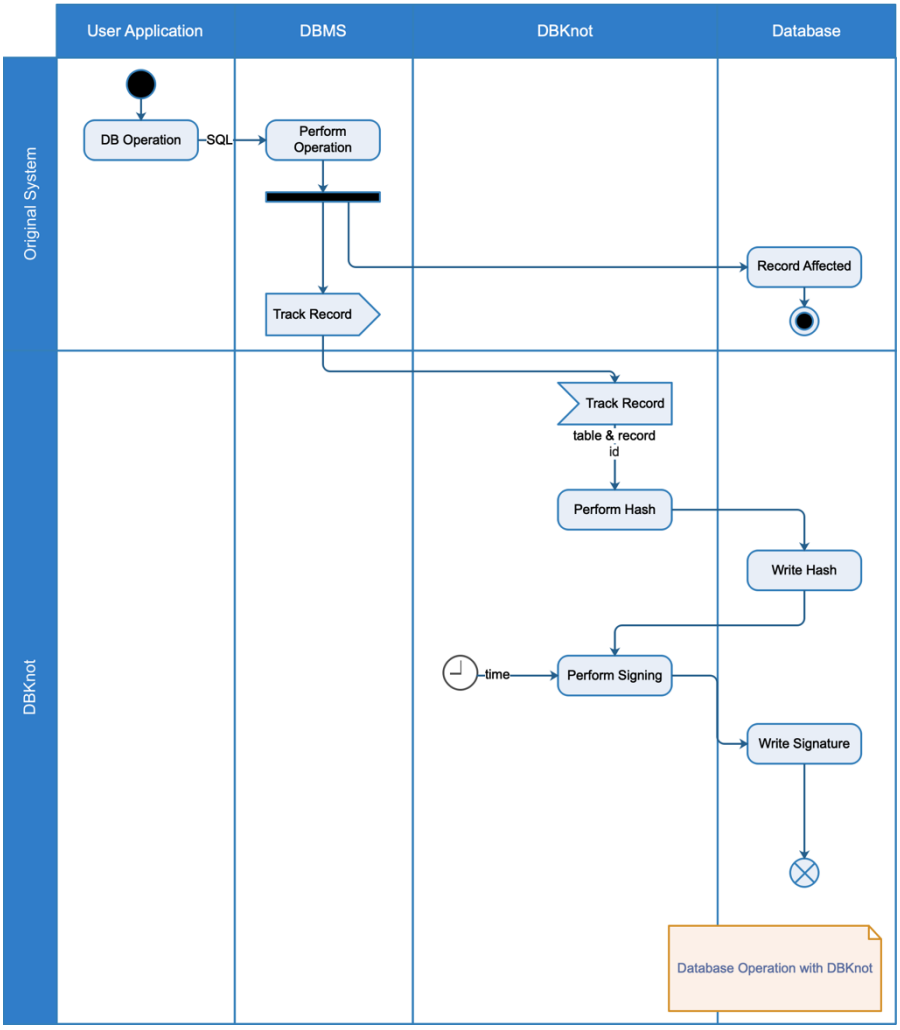


Figure 29 - Database Level DBKnot Integration

Figure 29 shows the asynchronous version of the DBKnot database level integration where the hashing and signing functionality is signaled by a trigger in the database level.

4.5.1 The Signer

The direction adopted is to introduce an externalized time-stamper/signer and/or a tamper-resistant HSM (Hardware Security Module). The role of the signer is to sign a hash of each record/transaction that gets added to the database. In addition to the record, a hash of the previous record will be added. A timestamp is also added to the signed data in order to protect against future signing-replay attacks.

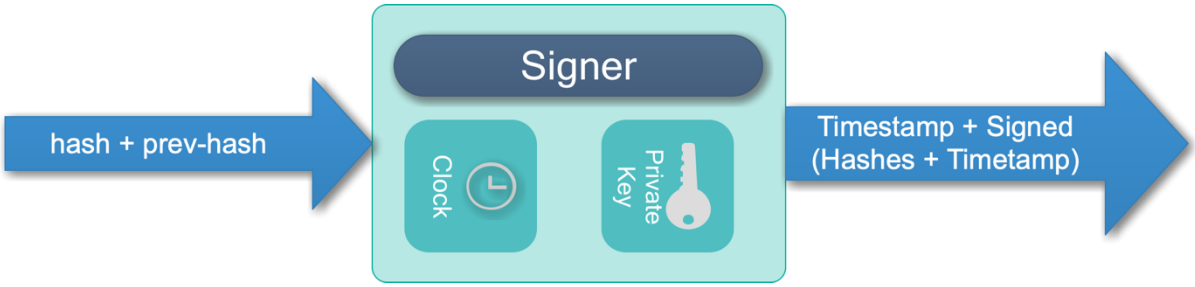


Figure 30 - Signer Service

4.5.2 A Chain of Hashes

A chain of the hashed transactions is being maintained. The chain includes the signed hashes of the data as well as the timestamps. Each record will include a hash of the previous record.

The chain of hashes is the only item that is added to the existing database. All other tables, field definitions, and records are untouched and remain intact.

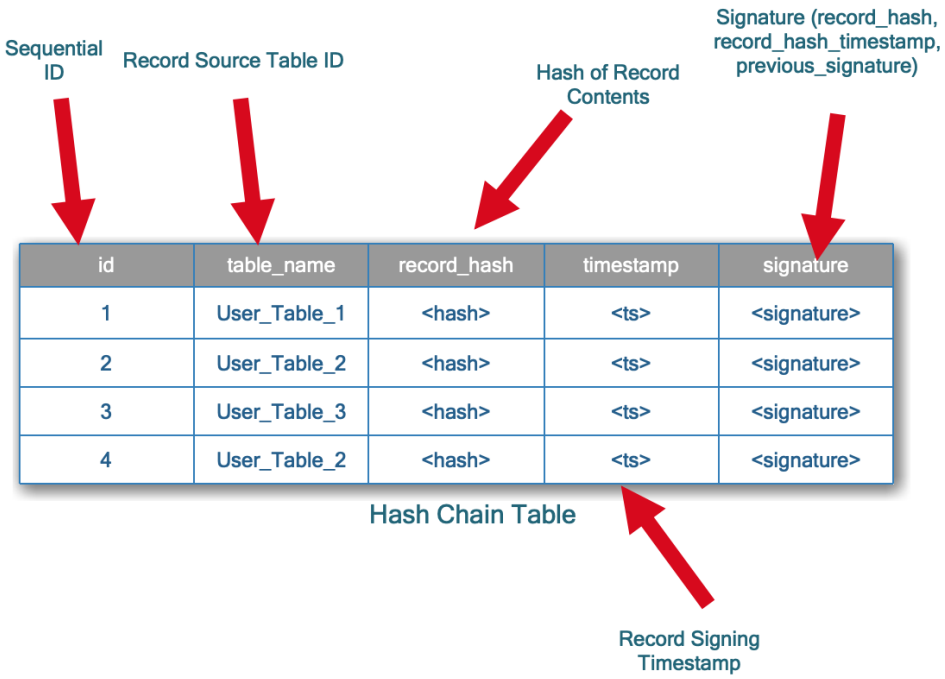


Figure 31 - Chain of hashes

As illustrated in Figure 31, The hash-chain-table is made up of the following fields:

1- **id:** A sequential ID. This is very important for identifying the sequence of transactions hashed. This is used during the signing and signature verification process.

2- **table_name:** The name of the table where the record came from. The hashing table is a database-wide table. Meaning that it contains hashes of all records regardless of which

table they come from. This keeps the hashing table as the only item added to the database and avoids making any changes of any other tables of the database to be secured.

3- **record_hash:** A hash of the record chained is placed in this field. In this research, MD5 hashing has been used. It is necessary that a fast hashing algorithm is used. Hashing is applied to a structure that contains a concatenated form of all record fields. SHA-256 or 512 could also replace MD5 for added security but with their corresponding performance tradeoff[94, p. 2]. We believe however that such a change may or may not be necessary depending on the application. It is not practical (in fact almost not possible) to generate a reverse hash for a specific number or piece of information that needs to be tampered. The only possibility here will be to generate a reverse hash to corrupt the data rather than put any meaningful data. Again, it could be configurable and left to be decided on a case-by-case basis.
4- **timestamp:** This field is filled by the data returned from the signer. It is the signature timestamp.
5- **signature:** In this field, the signature itself is stored as returned by the signer.

4.5.3 The Hasher

The hasher is the first step of the process. As soon as record is appended to any of the tracked tables, a hashing process is triggered. The hasher takes the inserted record, creates a structure that represents the concatenation of all fields, hashes that structure, and inserts all information describing that record in the hash table as described in section 4.5.2.

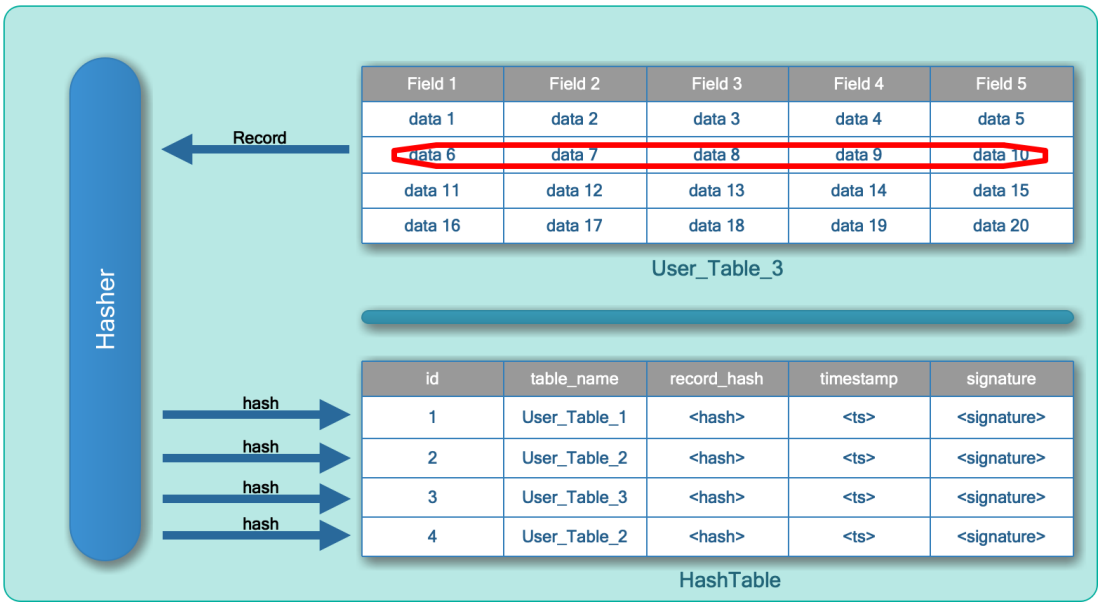


Figure 32 - Hasher

Parallelizable Hashing: By nature, the hashing process is parallelizable. This will utilize any available parallelism infrastructure present at the database server to optimize signing performance. In addition, it could be done by any external server that has access to the same database or a live replica of the database to relieve the primary server from extra computation work.

4.5.4 Inserting the Signer and Time-stamper

Once a record has been added, and after it has gotten automatically hashed, the corresponding hash record will be passed to the signer. The signer will take the hash record, add to it the preceding record together with a timestamp and sign them all with the signer public key. The signature of the preceding record could be appended to the hashed string instead of the hash, but we see that the hash will be sufficient because it will not be possible to tamper with the hash without breaking the signature. The resulting signature and timestamp will be returned to the database server and stored inside the hash table.

The signature saved in the hash table will be used for verification.

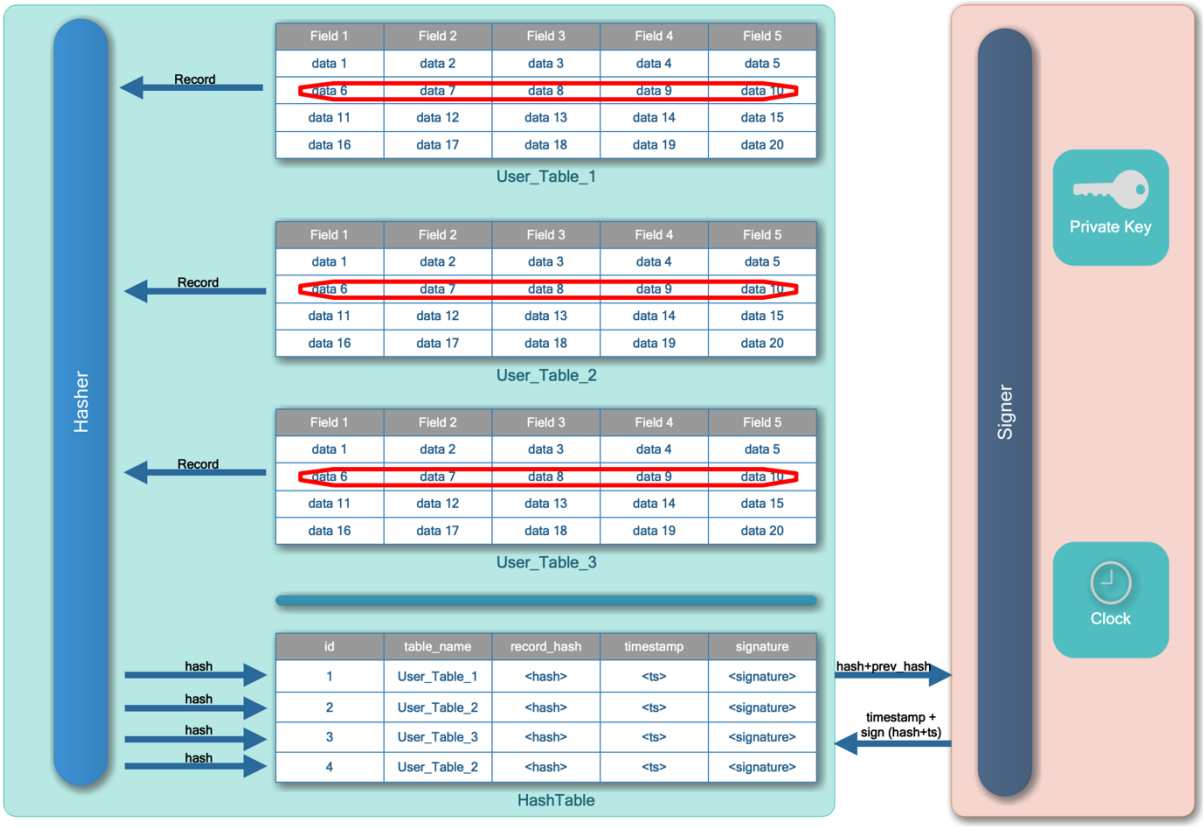


Figure 33 - Signer and time-stamper

4.6 Methods Used – Web Service/API

In this approach, the DBKnot functionality is to be implemented in the form of a reverse proxy/middleware that sits between all incoming API requests and the system being tracked.

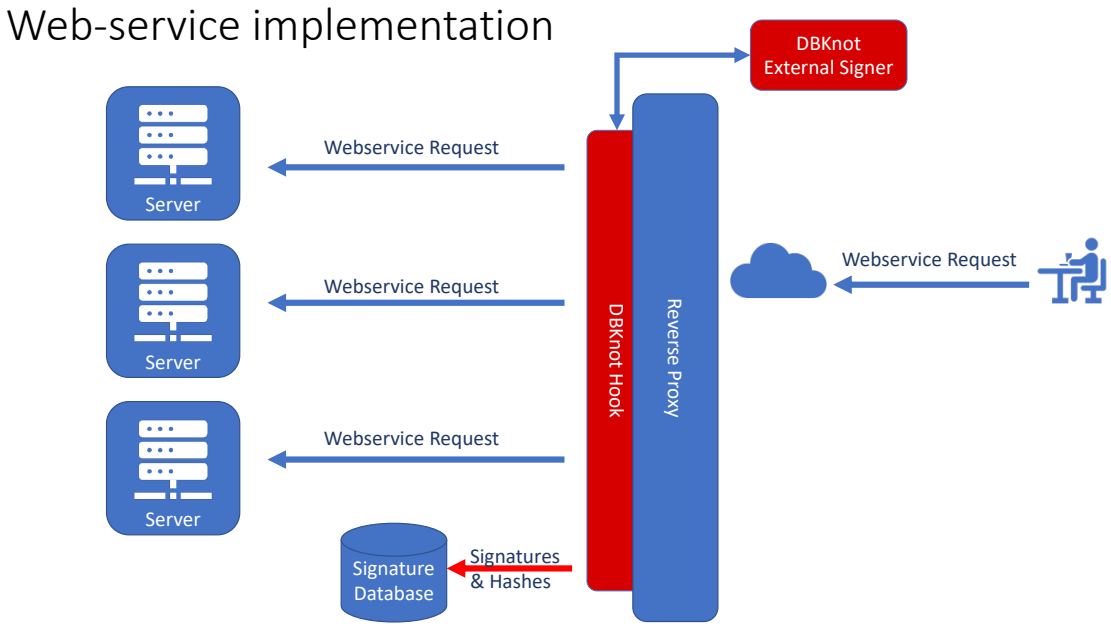


Figure 34 - Web Service Implementation

The following are the advantages of implementing DBKnot in the form of a web-service intermediary:

- **Technology agnostic:** Totally decoupled from any underlying technology used by the software implementation.
- **Supports hybrid microservices:** In an enterprise application or a set of applications that is dependent on numerous microservices, this design will be able to support all of the services even if they are implemented by different softwares/applications (e.g.: billing software + accounting software + CRM software, etc.)
- **Multi-server support:** This approach will function regardless of the number of back-end servers providing the service. It will also work in load-balancing use-cases.

The drawback / challenge however to implementing DBKnot as a web-service is the lack of adherence to a concrete and clear CRUD standard in the usage of REST web services. Accordingly, such implementation will need to be configurable to match each service that it intercepts. So, even though the original software is untouched, work will need to be done at the reverse-proxy level in order to configure DBKnot. And this will make it implementation specific.

4.7 Verification Steps

Verification of records and thus, the detection of possible tampering falls into the following 3 categories:

1- Malicious addition of a record
2- Malicious deletion of existing records
3- Malicious tampering with hashes or signatures
4- Malicious change of existing records

Each one of the categories is illustrated in more detail in their corresponding sections below.

4.7.1 Maliciously Added Record

When a record is maliciously added to one of the table by bypassing the hashing/signing functionality provided by DBKnot, when checked, such a record will not have a corresponding hash and signature as illustrated in Figure 35. Such check can be done on single or multiple records when requested. Alternatively, a housekeeping patrolling thread can regularly patrol the database to spot inconsistencies.

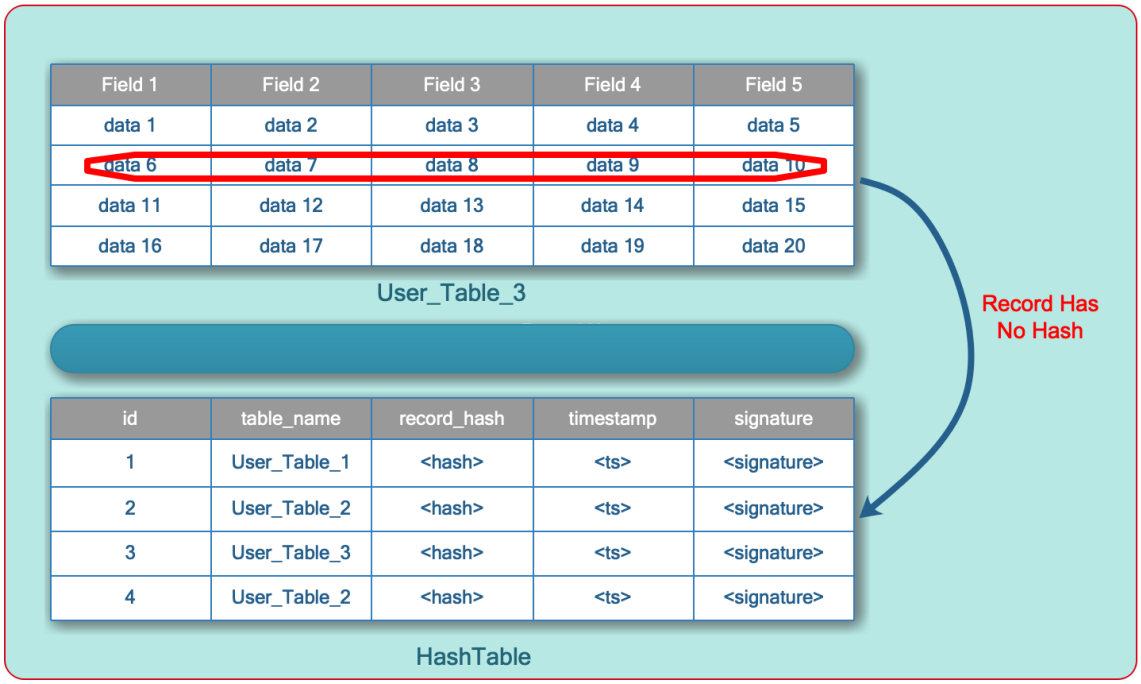


Figure 35 - Detection of a Maliciously Added Record

4.7.2 Maliciously Deleted Record

If a malicious user deletes a record from one of the tables, the deleted record's corresponding hash will still be present in the hashing table. During a verification pass, the dangling hash will be detected and will indicate that a record has been deleted. The table name of the deleted record will be identified and the creation time of the deleted record will also be present.

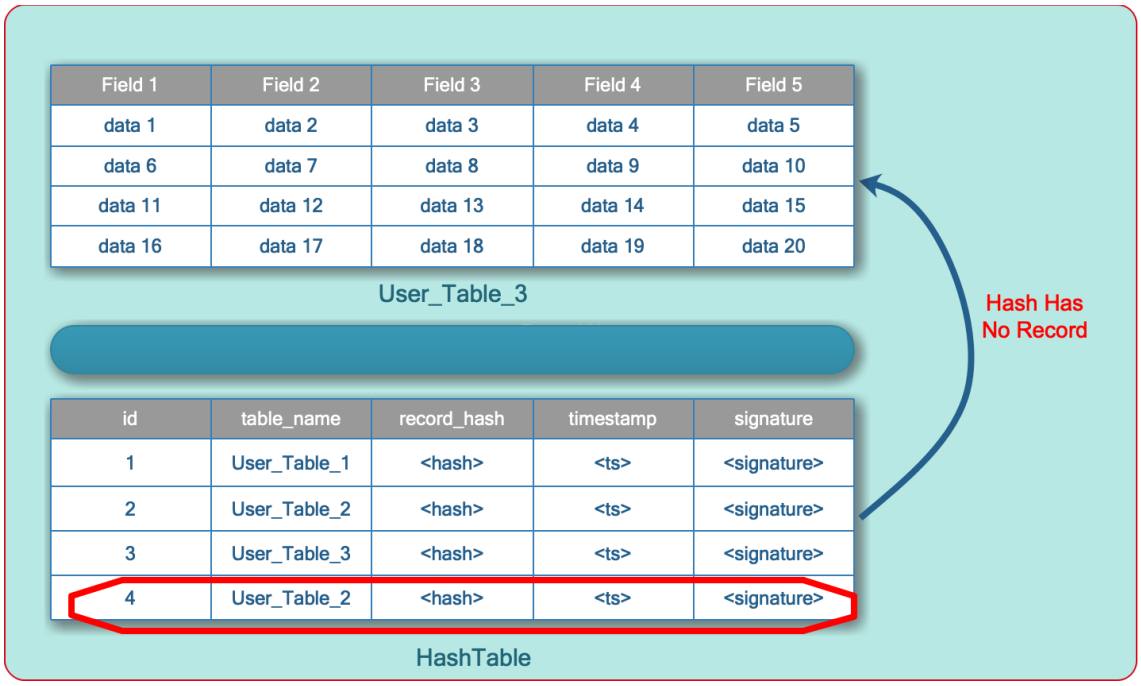


Figure 36 - Detection of a Maliciously Deleted Record

4.7.3 Signature Tampering

In addition to all the database tables, DBKnot relies on a local hashes/signatures database to detect tampering events. Being local, the table is within the reach of internal system administrators who can tamper with it and change it in an attempt to hide any trails of their fraudulent incident.

Any tampering with the hash / signature table will be detected by finding invalid signatures of chained hashes within the table.

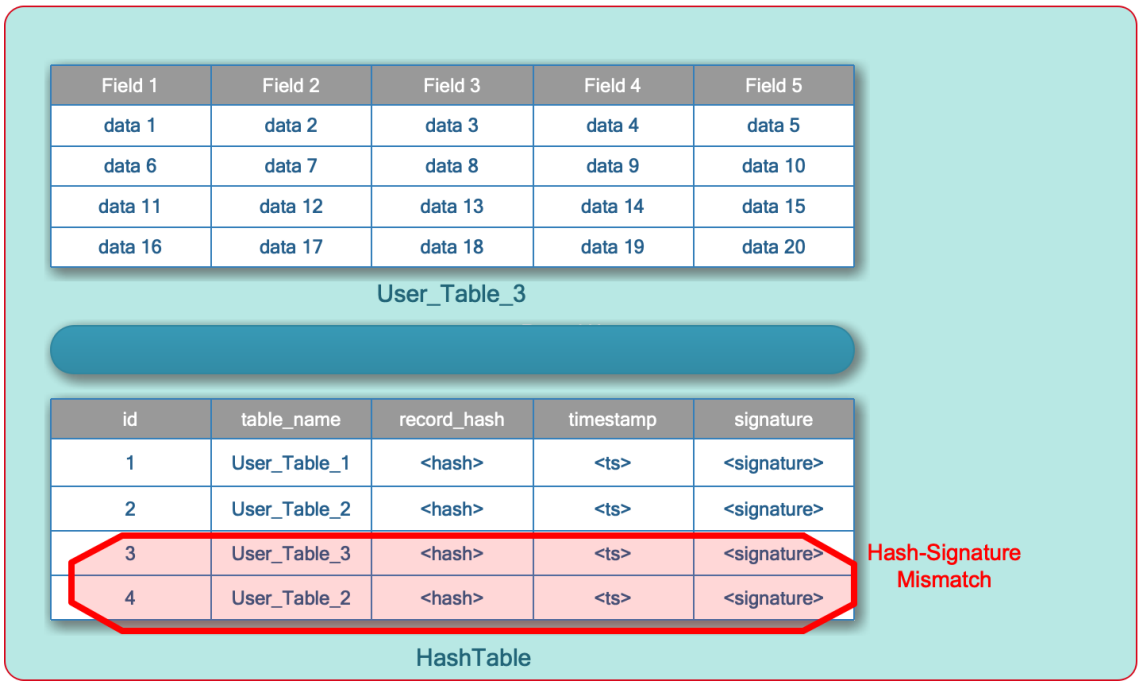


Figure 37 - Detection of Signature/Hash Tampering

4.7.4 Malicious Updating of Records

An updated record will be caught in both the addition and deletion verification processes. This is because its hash will no longer be present in the hashing table and therefore it will be detected as new maliciously added record. At the same time, its corresponding hash will be left dangling in the hash table while it will indicate that the record has been deleted.

A combination of a dangling as well as a hashless record indicate a changed record.

4.7.5 Types of Verification

There are two cases when a verification is triggered. The first one is at data-read or insertion time where 1 record needs to be verified. The verification step will trace the record back throughout the chain through an “n” predefined depth before generating the assumption that it was not tampered with within a particular time-window (1 week, 1 month, 1 year, etc.)

The second case is the case of patrolling threads/processes. These are housekeeping threads that regularly patrol the database to check and confirm the correctness of all records, hashes, signatures, and linkages.

We believe more work could be done on both verification cases to optimize such a process and increase the coverage of tests within the same short duration of time.

4.8 Performance Optimization

The additional tracking/hashing/signing layer does not come without an expense. There is of course a performance impact on insert transactions into the database. In this section we illustrate a number of different optimizations that could be used to mitigate and reduce such an impact. Most of them will be for the purpose of introducing different form of parallelism into the design.

4.8.1 Signing Parallelization

4.8.1.1 Hash Sharding – Consistent Hashing

In this design illustrated in Figure 38 - Parallel Signers - Consistent Hashing, a technique similar to database record sharding is used to distribute workload on a number of different shards. Instead of chaining signed blocks in a purely sequential manner, they are chained in a round-robin form. In this case, if the system is configured to use **"n"** shards, then each record **"i"** will be chained with distributed to shard **"$s = i \% n$"**. The record will be linked to the previous record in the same shard too. Please note that the "I" is the sequence ID of the hash record rather than the ID of any of the tables. So, there is no possibility of collisions with other IDs in the system.

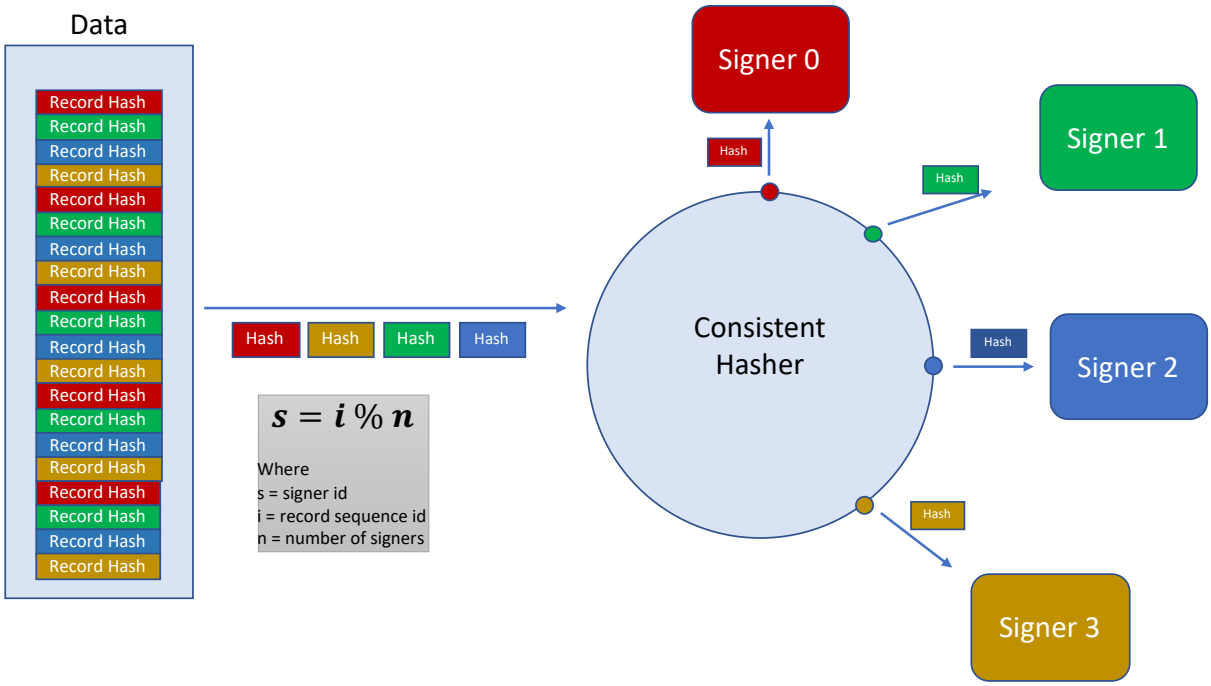


Figure 38 - Parallel Signers - Consistent Hashing

The advantage of this technique is that it breaks down the added latency and sequentiality of the process and introduces a degree of parallelism. Utilizing this method, a number of insert statements together with their corresponding hashes could be done in parallel without having to wait for each other to finish.

The tradeoff in this approach is that database verification is divided into “n” independent chunks which makes the chaining process less complex. One mitigation for that is to introduce occasional inter-shard linkages to tightly intertwine them together and eliminate that independence.

Physically, the implementation could be done using two different approaches, the first one is as outlined here, the hash entry slices/shards will be located in the same table and will be easily identified by id/address/location. On the other hand, the shards could be physically split in to physically different tables. This could be of help if a DBA believes that multiple tables will add to performance in a distributed clustering scenario.

4.8.1.2 Parallel Signers – Multi-queue Implementation

Figure 39 illustrates a multi-queue based implementation of the abovementioned hash sharding methodology.

Multiple queues are created, each for a corresponding shard. Transactions are fed into the queues in a round-robin form. Each queue in turn feeds the transactions into a signer worker thread/process/server.

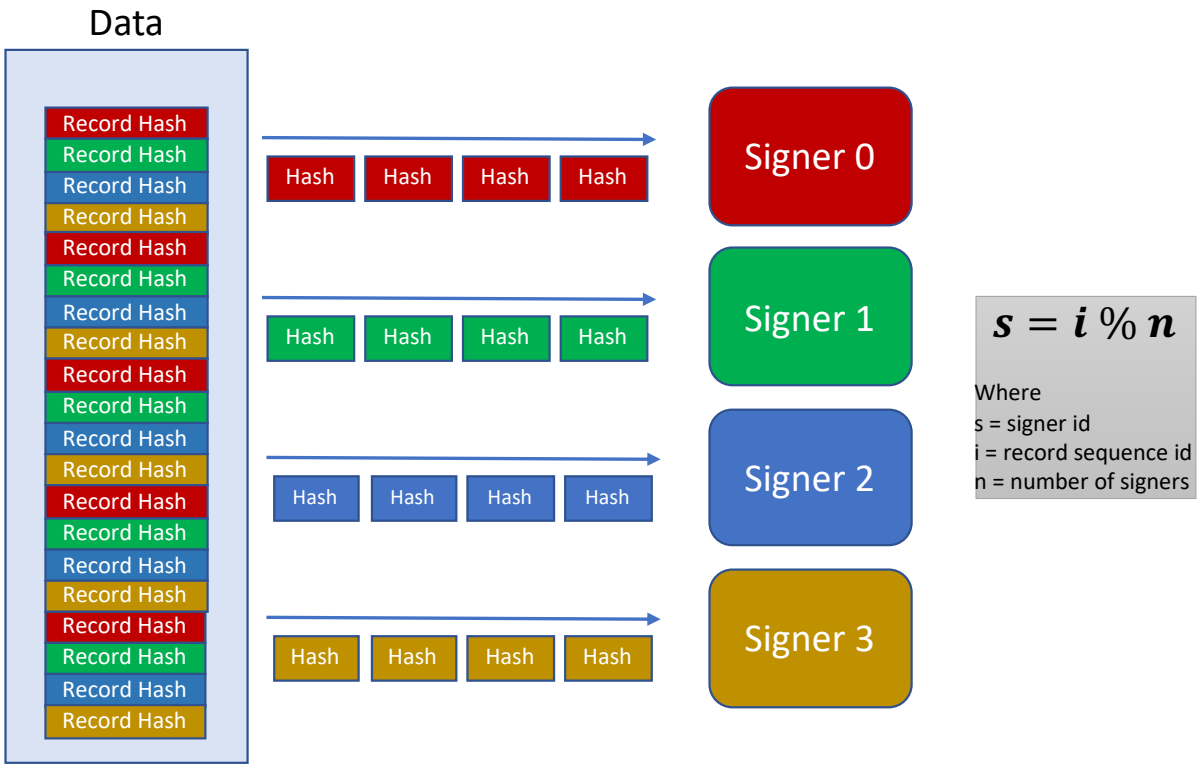


Figure 39 - Parallel Signers - Multi-Queue Implementation

Figure 39 Shows the queue based implementation of the consistent hashing approach outlined in Figure 38. Hashes are distributed to different signers in a round-robin manner through different queues. The round-robin selection criteria is the id of the hash in the hash table which guarantees fairness due to its reliable sequential nature. Please note that even though the hashing process or chains outlines in this section or in other alternative sections are divided into separate blocks, the assumption of record uniqueness still holds.

4.8.1.3 Linking of Hashes

Figure 40 illustrates how consecutive transactions are linked, hashed, chained, and signed together and how they are split into groups.

Chain 1	17 -> 13 -> 9 -> 5 -> 1
Chain 2	18 -> 14 -> 10 -> 6 -> 2
Chain 3	19 -> 15 -> 11 -> 7 -> 3
Chain 4	20 -> 16 -> 12 -> 8 -> 4

$$s = i \,\%\, n$$

Where
s = signer id
i = record sequence id
n = number of signers

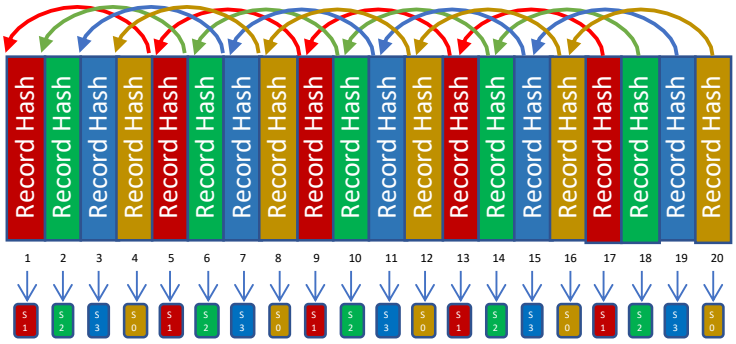


Figure 40 - Parallel Signers - Linking of Hashes

4.8.2 Coarse Grained Block Signing

In this approach, instead of performing hashing and signing on a record-by-record level, records are grouped into blocks. Each block is hashed together and then the group hash is signed by the signer.

4.8.2.1 Fixed Size Blocks

The figure below (Figure 41) shows how transactions batches are broken down into blocks and each block is hashed and signed separately. This approach reduces the signing overhead and enhances performance. Instead of a hashtable with an entry for every record, a smaller hashtable is utilized with a record per batch. There is a tradeoff however between the batch (block) size and the time required to verify a record.

Another drawback is that records of a whole batch will remain untracked until the batch is completed and signed. This will be problematic in cases where the database undergoes few transactions.

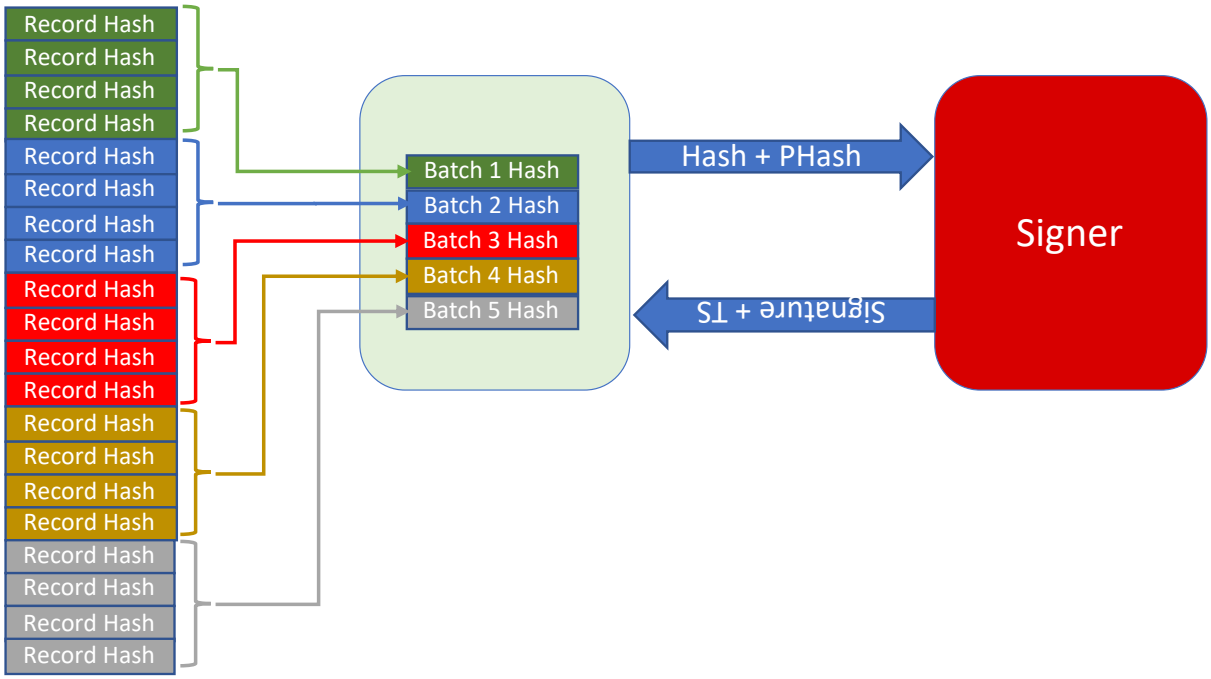


Figure 41 - Coarse Grained Block Signing

4.8.2.2 Variable Size Blocks

The solution to the previous shortcoming is to introduce variable-sized blocks (illustrated in Figure 42 - Coarse Grained Signing - Variable Block Size) where if a block remains open for a certain (configurable) duration of time, the system generates a clock-event. This clock event with its corresponding timestamp will force the closing and signing of the open block regardless of the number of records in the block. This approach will also have the added benefit of being able to work in an environment with intermittent or unreliable connectivity.

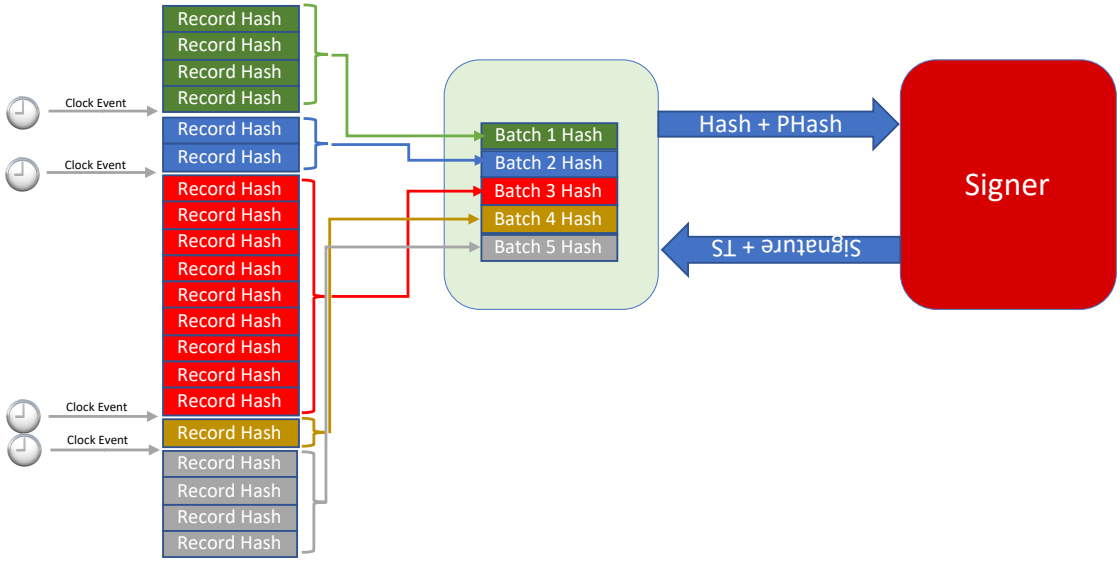


Figure 42 - Coarse Grained Signing - Variable Block Size

4.8.3 Serial/Successive Multi-Stage Signers

Another variant of the proposed design (Figure 43 - Multi-Stage Signing) is to employ a "multi-stage" successive signing where more than one signer are used in sequence. This approach will provide added security but will also increase the overhead in the transaction signing process.

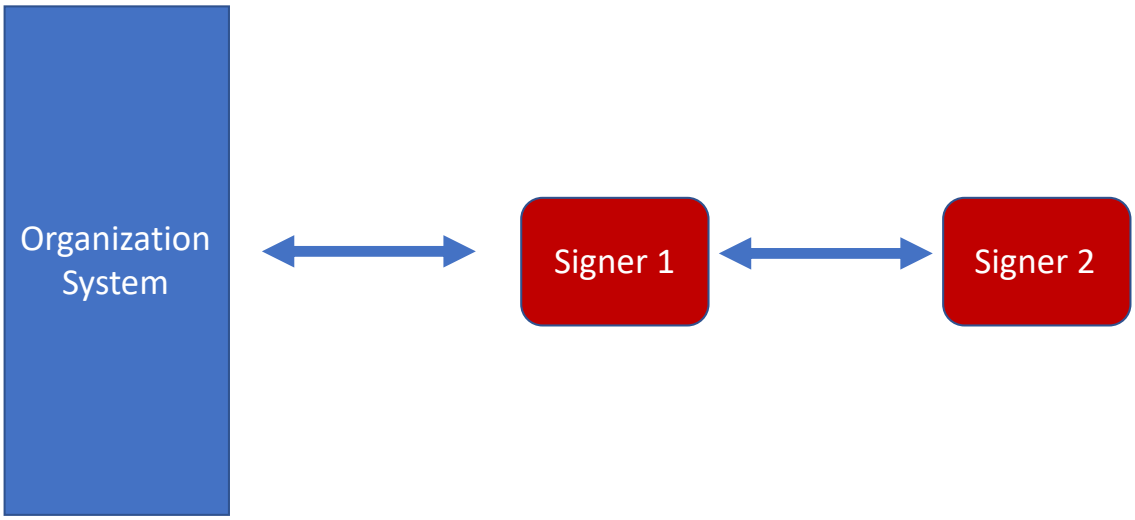


Figure 43 - Multi-Stage Signing

4.9 Performance Optimization – Pipelining

Four different techniques are being used for handling sequentiality / parallelism in implementing the DBKnot chaining process.

The first technique is purely sequential, the second technique pipelines the signing process, the third technique pipelines both the hashing and signing processes combined, and the fourth technique designs everything to be pipelined.

Each one of the techniques will be further explained in its own corresponding section.

4.9.1 Parameters

For each of the techniques used, there are 3 assumed scenarios that will be tested. All the scenarios are variants of the following set of variables:

- **Transaction time:** The time taken to perform a transaction on the database
- **Hashing time:** The time taken to hash a transaction
- **Signature time:** The time taken to sign the hashes and produce a signature

All Variables

$$n = number\ of\ transactions$$
$$t = transaction\ time\ (t1 \rightarrow short\ transaction, t2 \rightarrow long\ [4X]\ transaction)$$
$$h = hashing\ time$$
$$s = signing\ time$$
$$v = total\ batch\ duration$$

Figure 44 - Testing Variables

The following categories of transactions were derived from the preceding variables:

- **Transaction Bound:** In these scenarios, the transaction time is the longest of the 3 numbers.
- **Hashing Bound:** In these scenarios, the hashing time is the longest of the 3 numbers.
- **Signing Bound:** In these scenarios, the signing time is the longest of the 3 numbers.

All tests are done on 2 batches of transactions, one of them is made up of transactions that require a small "t1" to run, another one is a long batch with transactions taking longer time "t2" where ($t2 = 4 \times t1$). There are two other intermediate batches but we have decided to not include their results in this document due to the sufficient clarity of the other samples.

4.9.2 Technique 1: Inline Hashing & Signing

The first technique is used is to perform the transaction, followed by the hashing process, followed by the signing process. They are all done in series as illustrated in **Error! Reference source not found.** Figure 45.

Figure 45 - Inline Hashing & Signing

Error! Reference source not found. Figure 47 shows 3 scenarios of implementing the “all-inline” sequential method. Such scenarios are used in for comparison of different techniques under different conditions.

The formula in Figure 46**Error! Reference source not found.** shows that due to the linear dependency nature of this approach, the total time taken is a simple sum of the total time taken for each transaction (transaction time “t” + hashing time “h” plus signing time “s”) and that the process is a very basic sequential one without any performance gains from any potential parallelism.

All Inline Formula:

$$v = \sum_{i=0}^{n} t + h + s$$

Figure 46 - Formula for "all inline"

Figure 47**Error! Reference source not found.** shows a visual representation of the all-inline technique. The illustration shows the 3 cases: a) transaction-bound, b) hashing bound, c) signature bound. We will notice here that it is consistent with the equation in Figure 46.

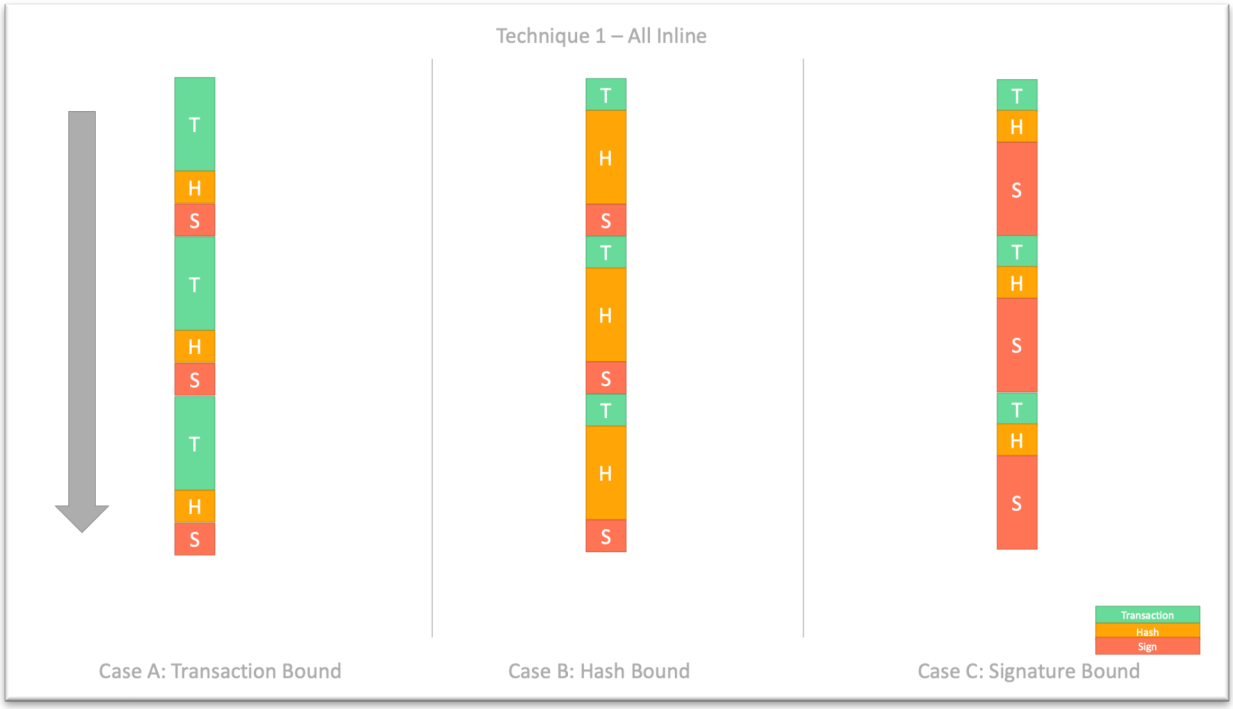


Figure 47 - Inline Hashing And Signing Illustration

The activity diagram in Figure 48 illustrate this technique and its sequential nature of inlining the transaction, hashing, and signing processes.

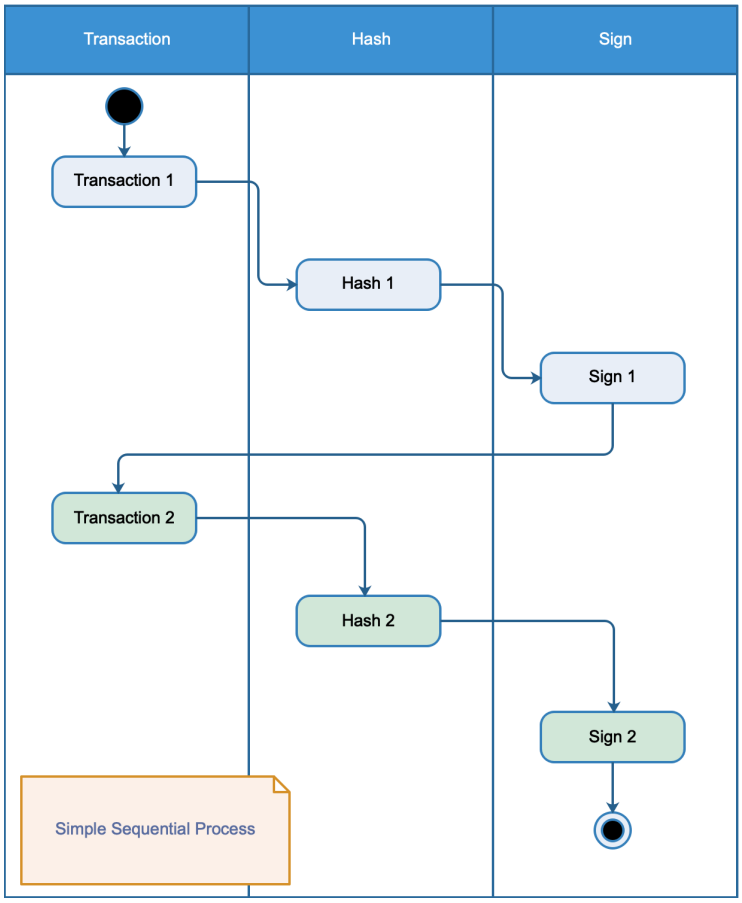


Figure 48 - Technique 1 – Inline Hashing and Signing - Activity Diagram

4.9.3 Technique 2: Partial Concurrency Through Signature Pipelining

This technique removes the signing process out of the main execution pipeline to allow running it in parallel when needed to gain some performance.

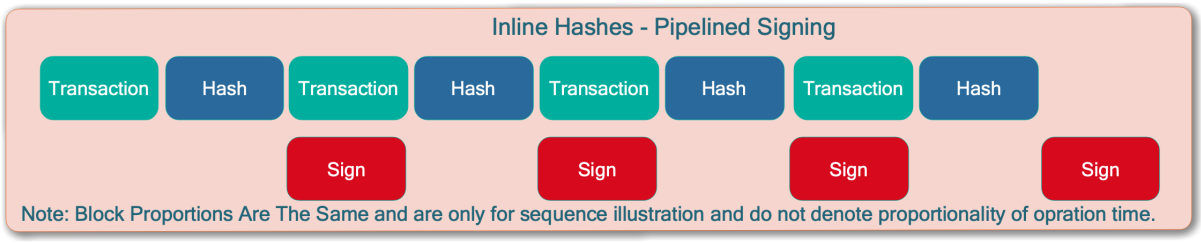


Figure 49 - Partial Concurrency Through Signature Pipelining

Formula for Signature Pipelining:

$v1 = s + \sum_{i=0}^{n} t + h$	$v2 = t + \sum_{i=0}^{n} s + h$
$v = \max(v1, v2)$	

Figure 50 - Formula for signature Pipelining

Figure 51**Error! Reference source not found.** shows a visual representation of the signature pipelining technique. The illustration shows the 3 cases: a) transaction-bound, b) hashing bound, c) signature bound

Figure 51 - Signature Pipelining Illustration

Figure 52 shows the parallelization of the signing process. Please note that the transaction and hashing remain sequential.

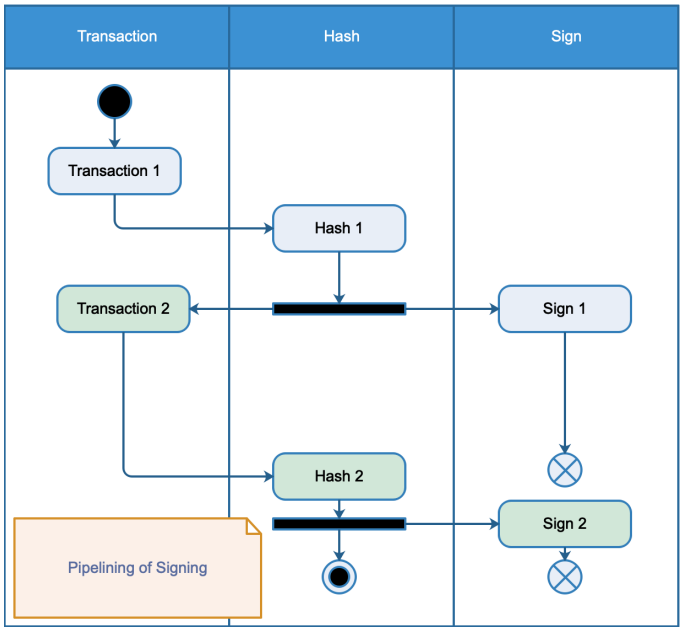


Figure 52 - Pipelined Signing - Activity Diagram

4.9.4 Technique 3: Concurrency Through Hash and Signature Pipelining

This technique separates the hashing and signing from the main thread and executes them separately in a single thread of sequential execution.

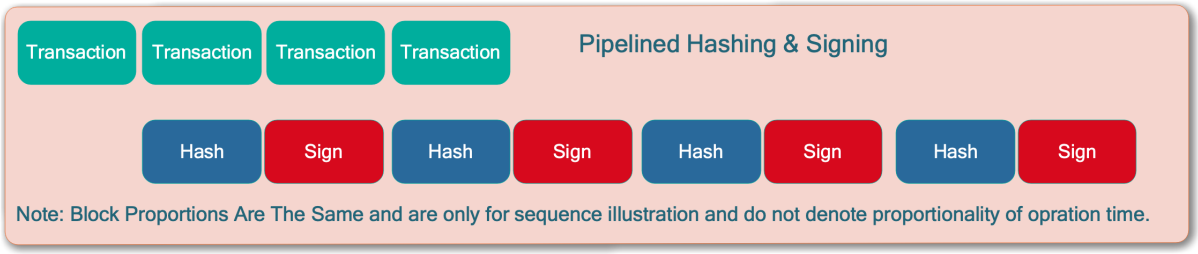


Figure 53 - Concurrency through Hash and Signature Pipelining

Formula for Hash And Signature Pipelining:

$v1 = s + \sum_{i=0}^{n} t + h$	$v2 = t + \sum_{i=0}^{n} s + h$
$v = \max(v1, v2)$	

Figure 54 - Formula for signature and hash pipelining

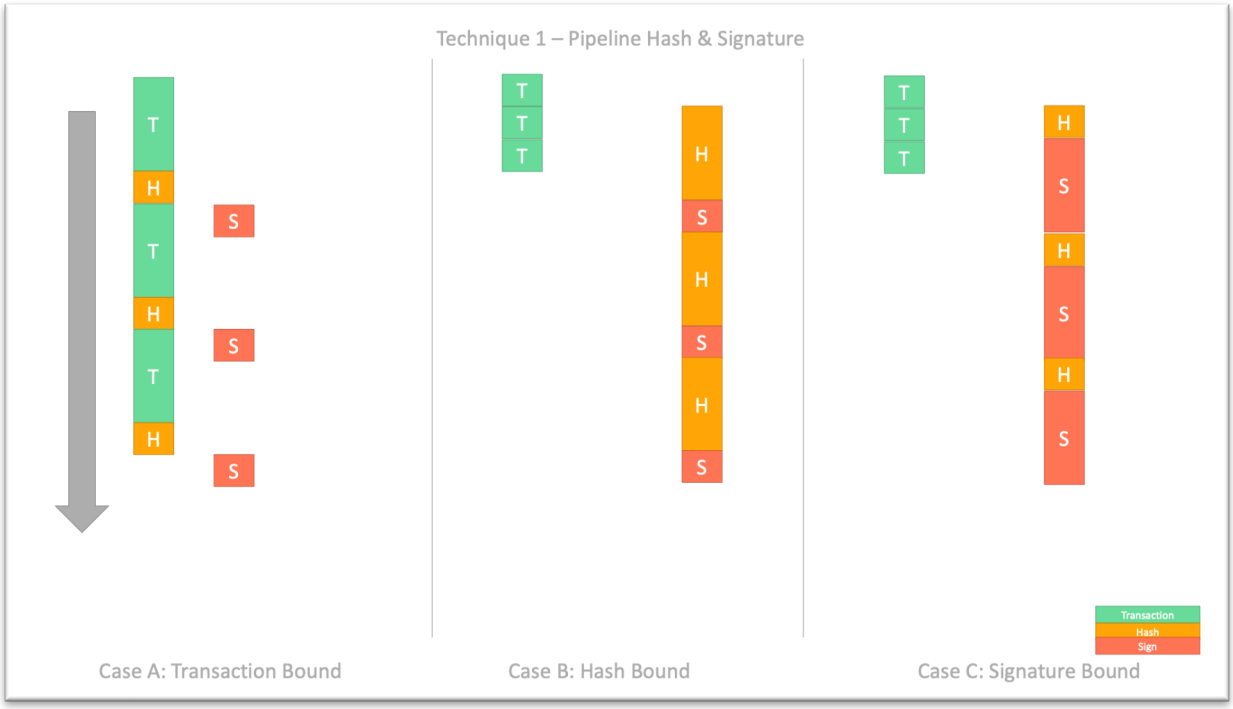


Figure 55 - Hash and Signature Pipelining - Illustration

Figure 56 shows the 3rd technique of pipelining both the siging and hashing. Please note that they are both sequential as well. The signing process has been increased in duration to illustrate the sequential nature of the process and its impact.

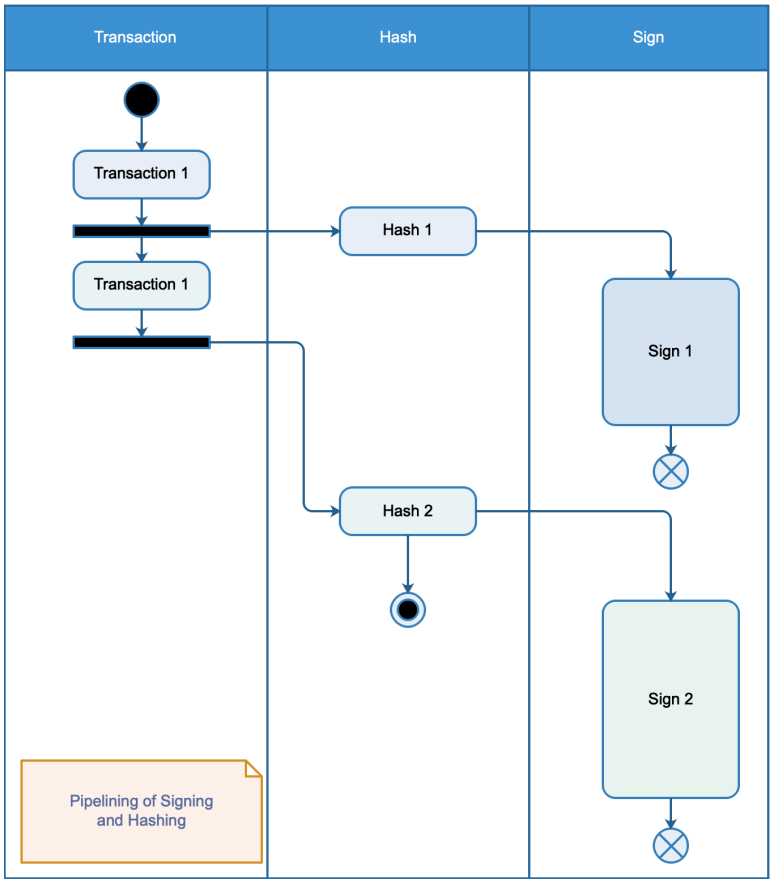


Figure 56 - Pipelined Hashing and Signing - Activity Diagram

4.9.5 Technique 4: Concurrency Through Pipelining All Operations

This technique as illustrated in **Error! Reference source not found.** is different from all the others above. In this technique we separate each of the 3 steps (transaction, hashing, and pipelining) into its own pipeline and let them run asynchronously while preserving sequence dependencies.

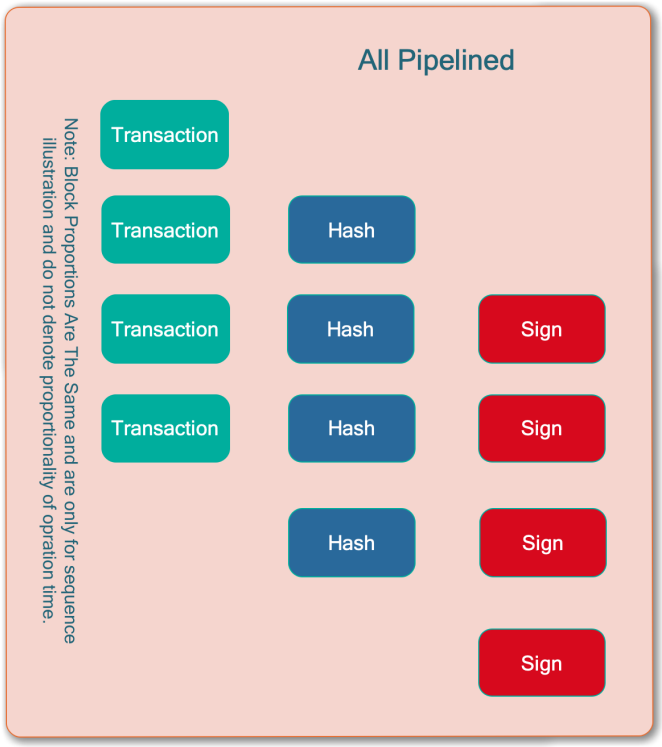


Figure 57 - Pipelining All Operations

Formula All Pipelining:

$v1 = h + s + \sum_{i=0}^{n} t$	$v2 = t + s + \sum_{i=0}^{n} h$	$v3 = t + h + \sum_{i=0}^{n} s$
$v = \max(v1, v2, v3)$		

Figure 58 - Formula for pipelining all operations

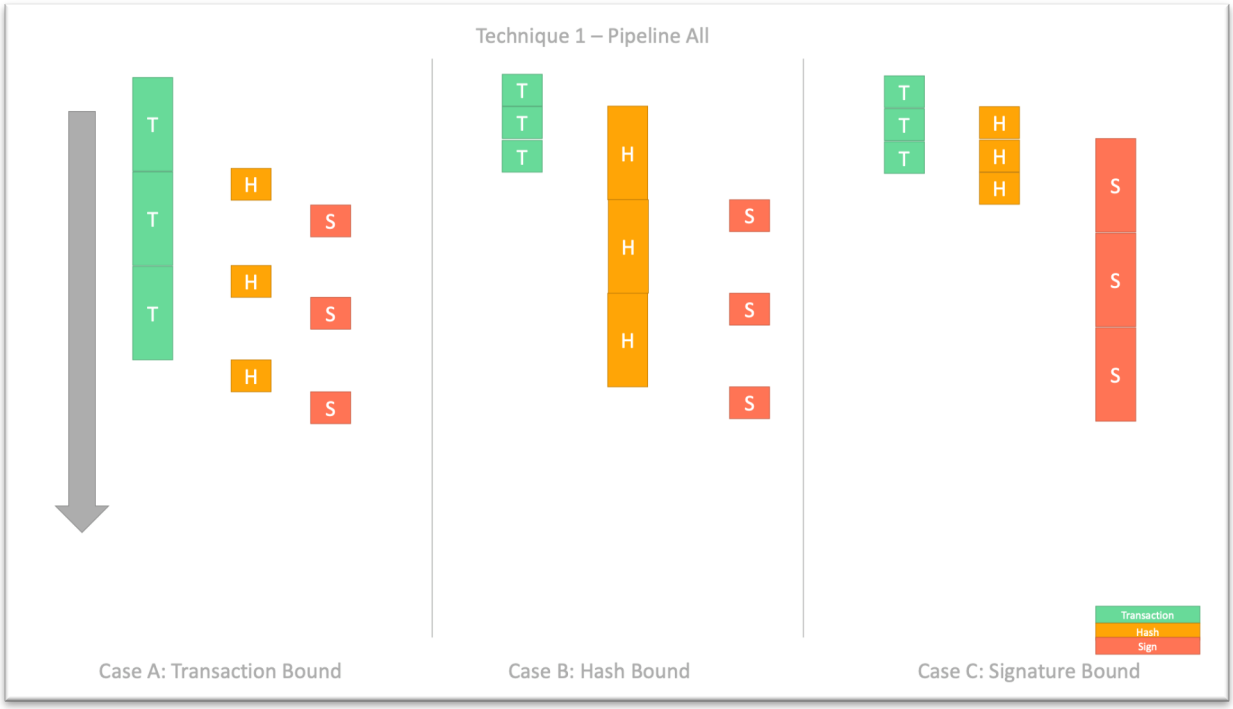


Figure 59 - Pipelining All - Illustration

Figure 60 illustrates technique 4 where everything runs in parallel. Where a hasher is separate from a signer and separate from the main transaction thread of execution.

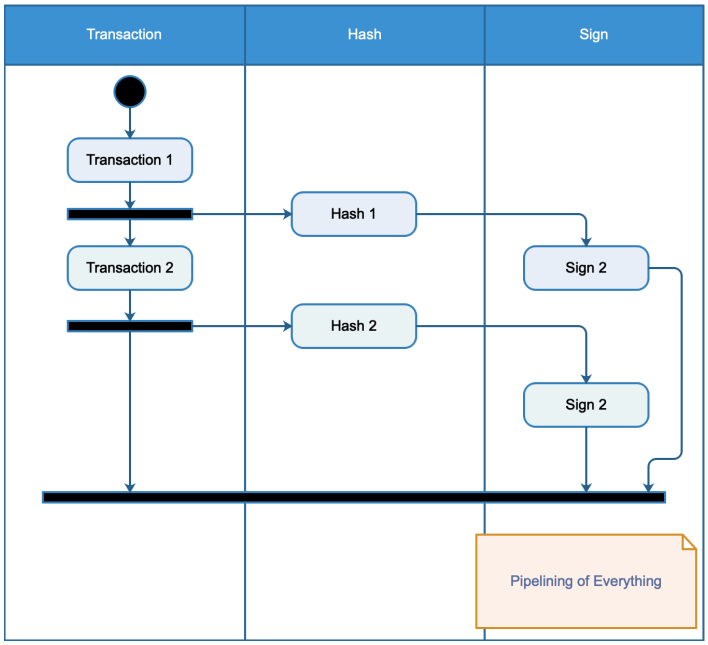


Figure 60 - Pipeline All - Activity Diagram

5 Experimentation and Results

The following sections are designed as an assessment of implementing DBKnot in different contexts and under different parameters. Some of the assessments will be done by testing on assumed/generated workloads while others will be done by real experimentation using real data.

Workloads were automatically generated by taking into consideration covering all different combinations of different inputs. For example, signing time was generated to include a whole spectrum of signing time to take into consideration the existence of local vs. remote signer and different delays in the signing process. The same was done for the hashing time as well as transaction time.

5.1 Setup

The following hardware and software configuration was used to run the experiments of the proposed architecture and that yielded the results below.

Item	Specs
CPU(s)	Intel Core i7-4790 CPU – 3.6Ghz
System Harddrive	Samsung SSD 850 EVO **250GB** (EMT03B6Q) Read Throughput: **493.24 MB/sec**
Data Harddrive	**2TB** WDC WD20EZRZ-00Z5HB0 (80.00A80) Read Throughput: **157.19 MB/sec**
PostgreSQL	12.2
Docker	19.03.11
Python	3.8.2
Psycopg	2.8.5

Figure 61 - Experiment Setup Environment

5.2 Dataset – Stock market Data [95]

The sample chosen for testing the DBKnot model is the stock market data from 1980 until 2013. [95]

Data sample size: 16,993,158 records

Data is formatted as a single table with the following data:

- Stock name

- Open price
- Close price
- Low
- High
- Volume
- Date

Stockmarket data was used for the initial experiments then the experiments were extrapolated with generated/assumed workloads.

As illustrated in the solution design section, all experiments are done on two types of transactions. A short transaction taking time “x” and a long transaction taking time 4x

5.2.1 Technique 1: Inline Hashing and Signing

By examining Figure 62 we will notice that the results are as expected in such a simple scenario. Total duration starts small and increases along the x-axis (hashing time) and along the y-axis as well (signing time). By combining both together, the total duration increases as we move diagonally along the x and y axis at the same time.

5.2.1.1 Short (X) Transaction Duration

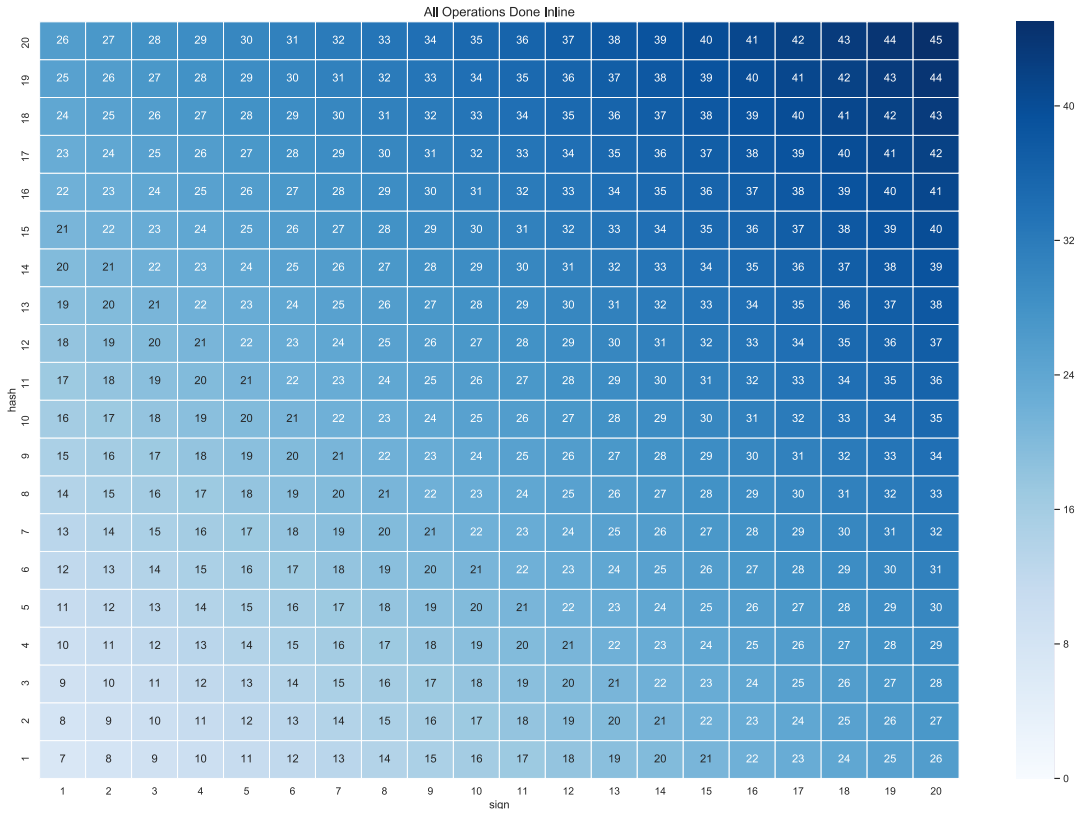


Figure 62 - Inline Hashing and Signing - Short Transactions Heatmap

5.2.1.2 Long (4X) Transaction Duration

Figure 63 shows for the long-duration transactions the same exact pattern as in the short-duration one above. The only difference is the scale which is normal because of the difference in transaction time. Instead of a maximum of 45 for the short transactions, the maximum for the long transactions is 60. The distribution of the pattern is the same though.

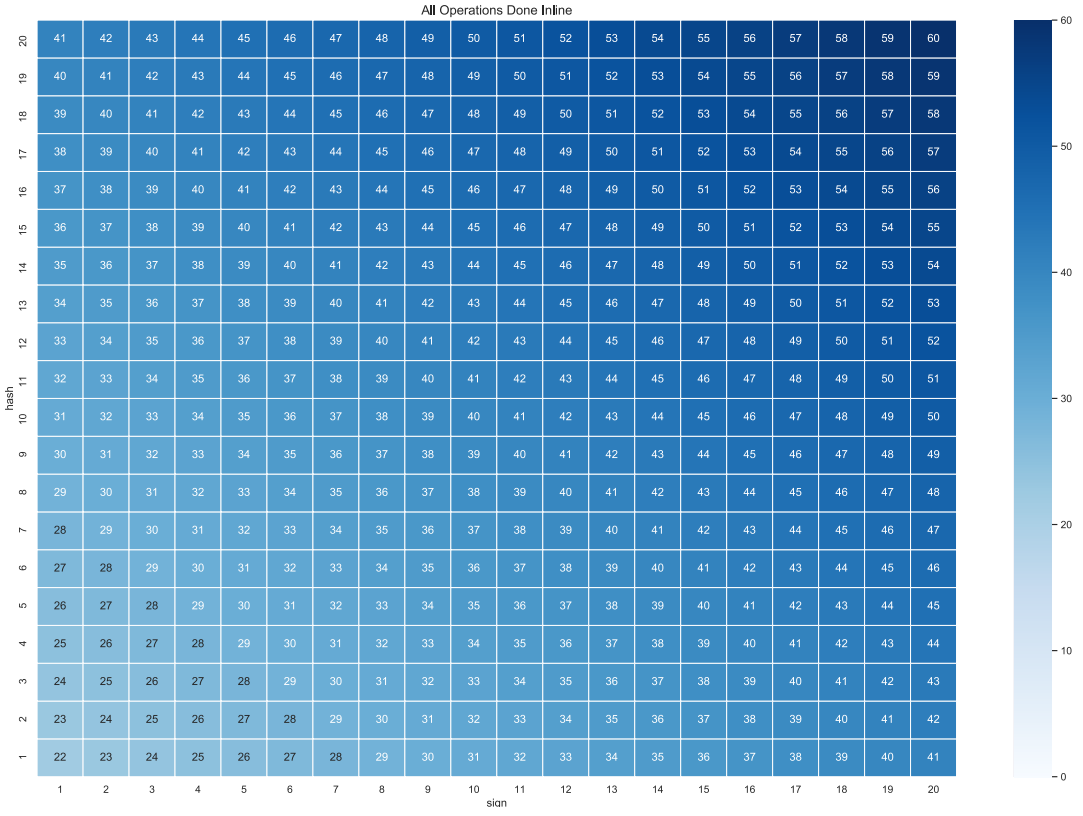


Figure 63 - Inline Hashing and Signing - Long Transactions Heatmap

5.2.2 Technique 2: Partial Concurrency Through Signature Pipelining

5.2.2.1 Short (X) Transaction Duration

By examining the heatmap below, we find out that the performance of the signature is almost the same as the all-inline technique above from the distribution standpoint. There is however a difference in the overall performance where the pipelining of the signing process has achieved a slight improvement.

It is also worth noting that the impact of the increase in the time taken to sign transactions is significantly less than the impact of a similar change in the hashing time. It is also much less than the similar impact in the all-inline technique above.

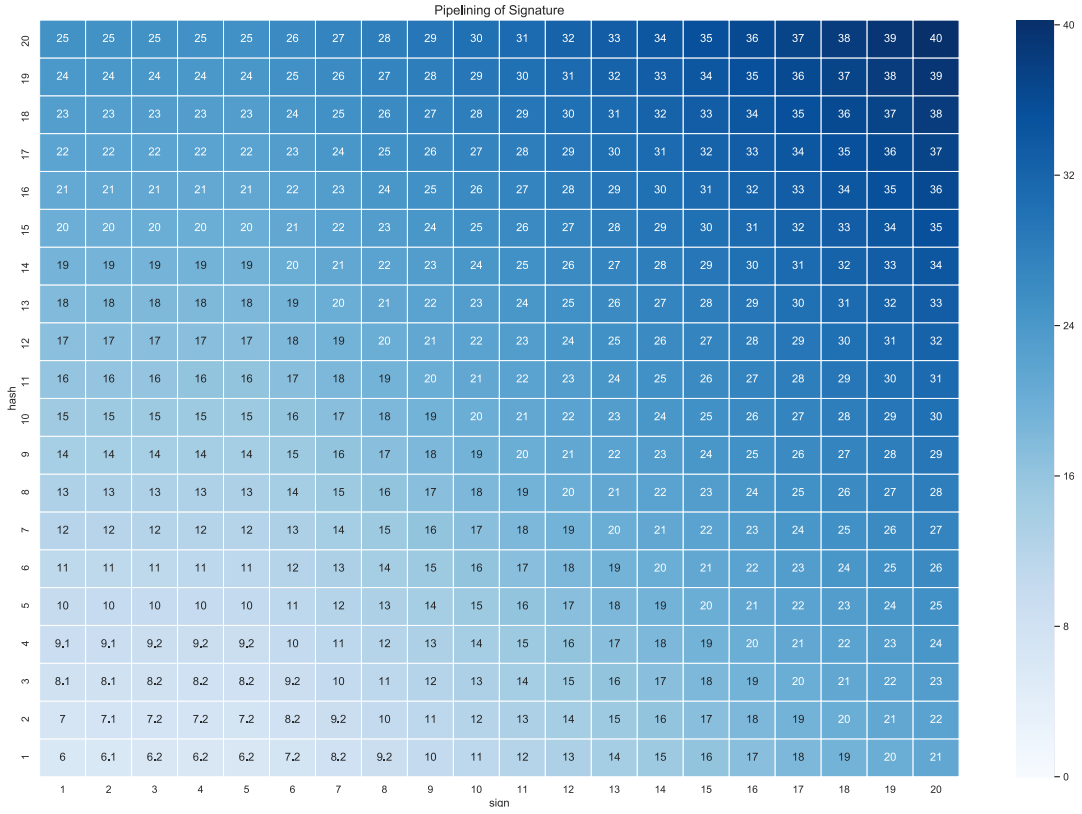


Figure 64 - Signature Pipelining - Short Transactions Heatmap

5.2.2.2 Long (4X) Transaction Duration

The heatmap below highlights the properties of this technique much more. Such a behavior is still consistent as the duration of transactions increases.

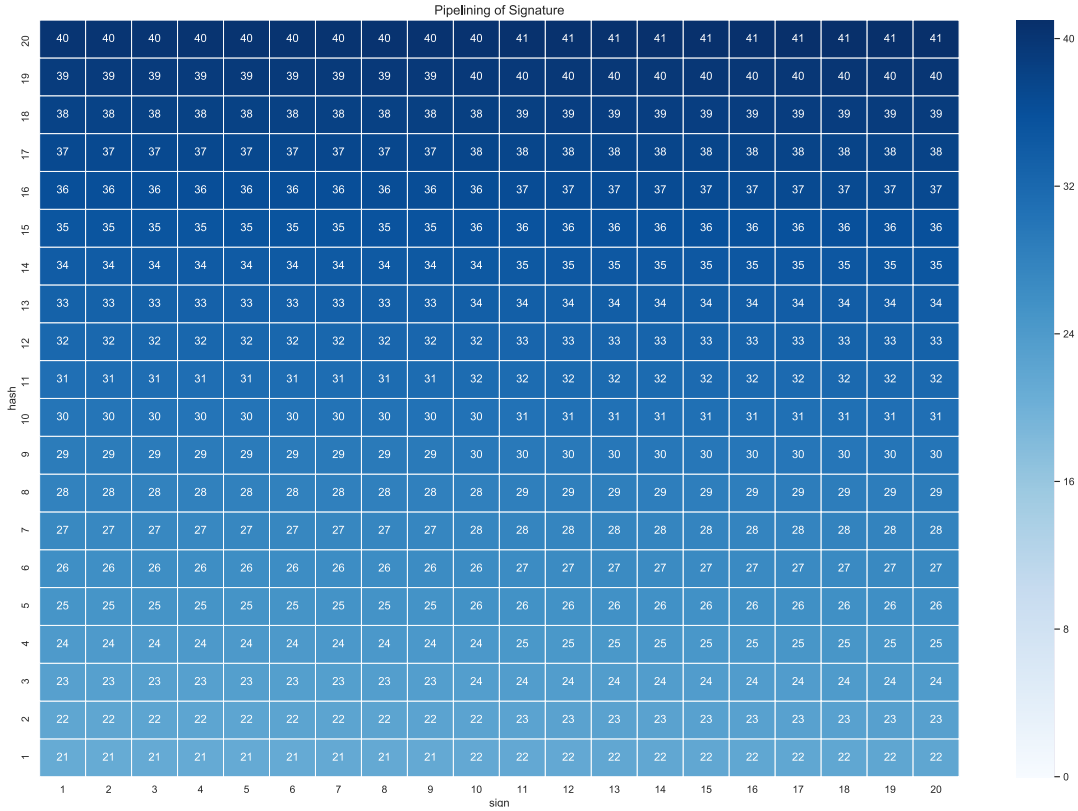


Figure 65 - Signature Pipelining - Long Transactions Heatmap

5.2.3 Technique 3: Concurrency Through Hash and Signature Pipelining

By looking at the resulting charts, we will find out that the pattern is similar to that of technique 1 in terms of distribution. i.e.: Both hashing and signing impact the performance equally in terms of distribution pattern. There is of course a difference in overall performance that will be highlighted later in the document.

5.2.3.1 Short (X) Transaction Duration

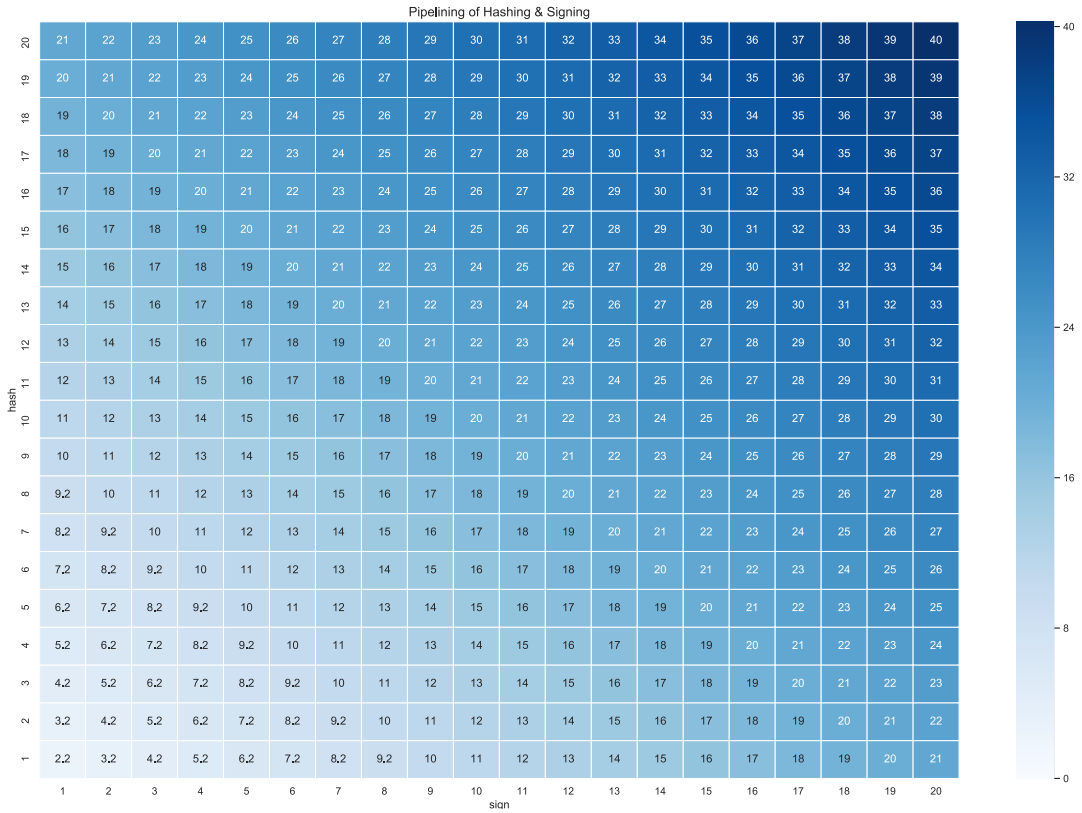


Figure 66 - Hash And Signature Pipelining - Short Transactions Heatmap

5.2.3.2 Long (4X) Transaction Duration

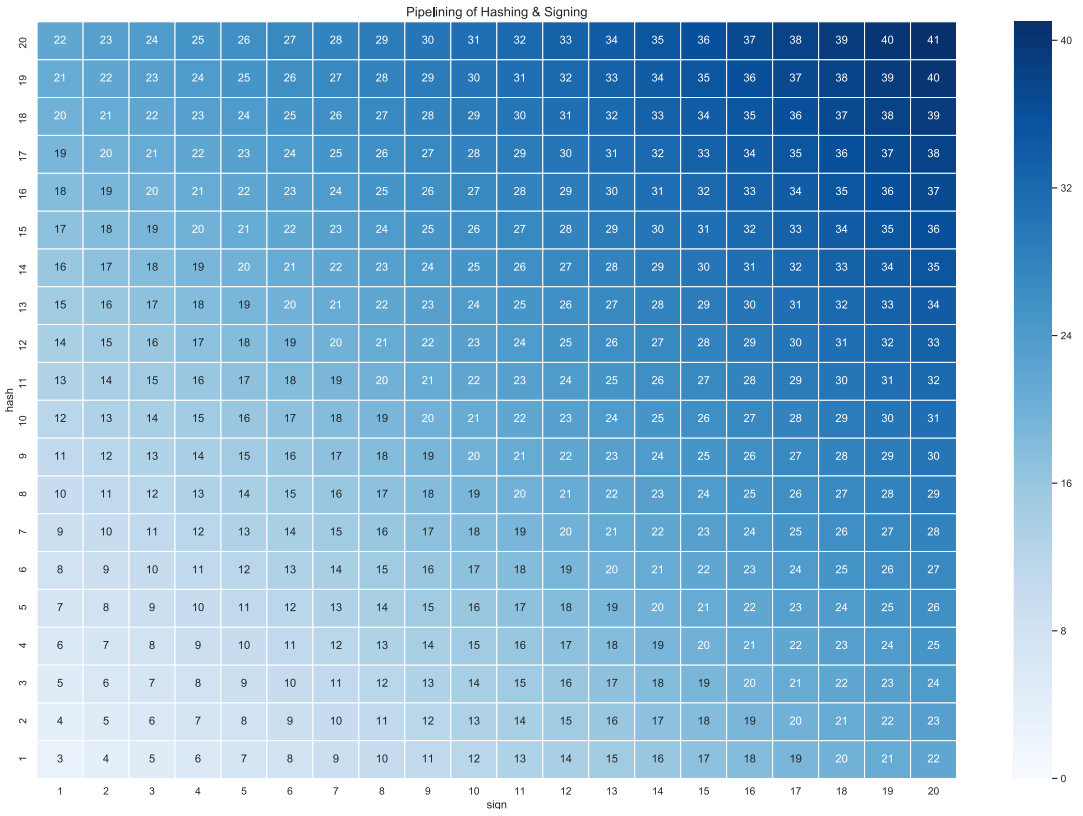


Figure 67 - Hash And Signature Pipelining - Long Transactions Heatmap

5.2.4 Technique 4: Concurrency Through Pipelining All Operations

5.2.4.1 Small (X) Transaction Duration

The heatmap in Figure 68 shows a pattern that is slightly different from the patterns above. Both increase in hashing time and in signing time equally impact the performance. There is a difference though between the impact if the hashing or signing time are equal (or close to one another) or if they are different. The more similar they are, the more stably increasing the time is.

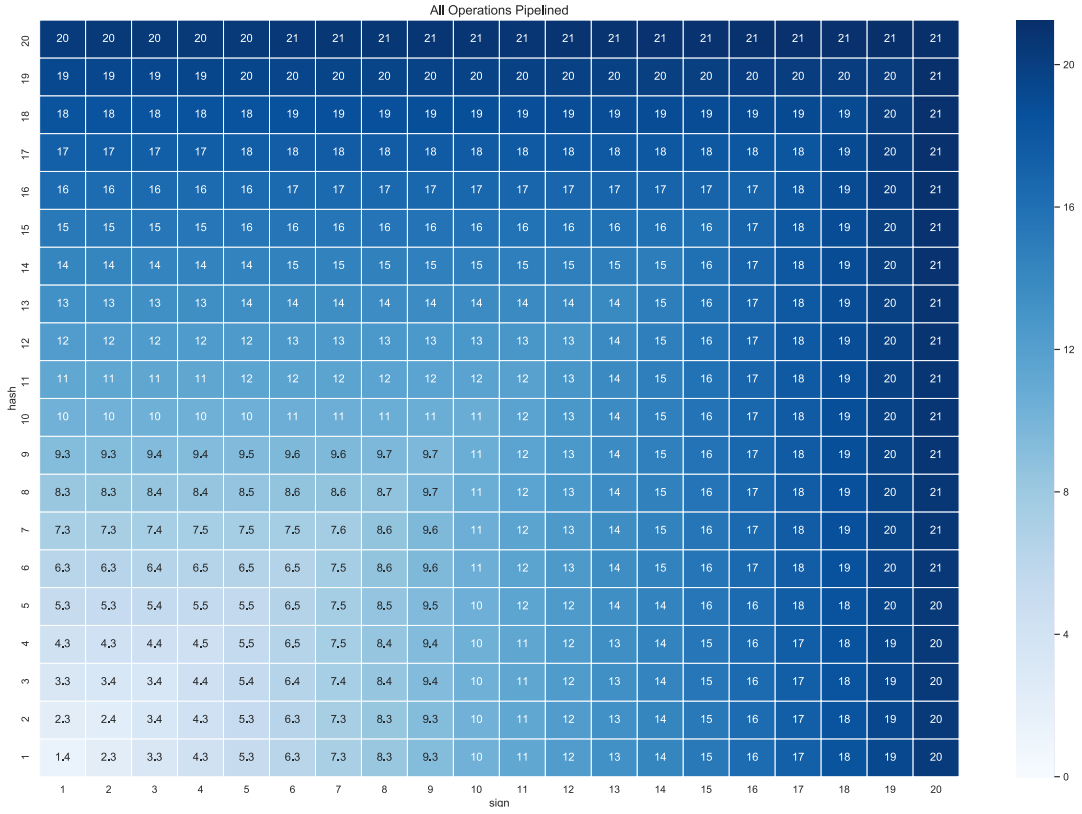


Figure 68 - Pipelining All - Short Transactions Heatmap

5.2.4.2 Long (4X) Transaction Duration

The second (longer) iteration of the all-pipelined technique results in a similar distribution as the short iteration. The noticeable difference however is that the performance per transaction has reflected a very slight increase as opposed to other techniques. This will be clearer in the comparisons below.

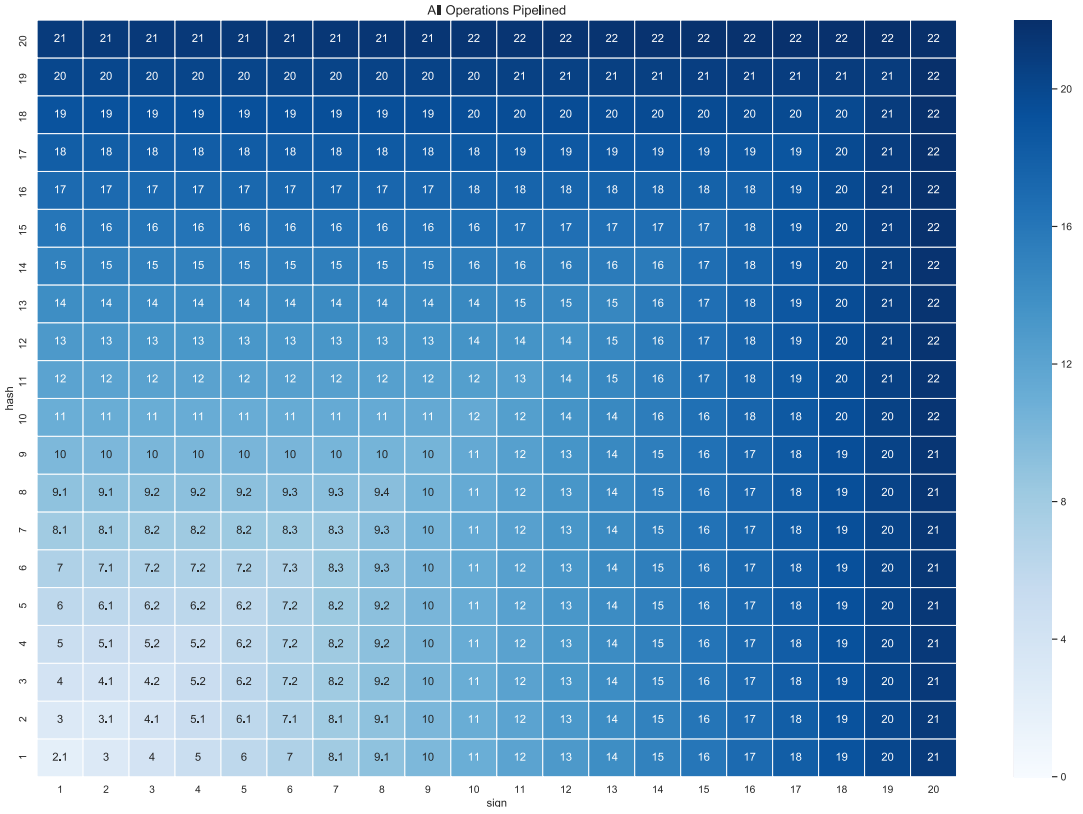


Figure 69 - Pipelining All - Long Transactions Heatmap

5.2.5 Comparison

The two comparison sets of heatmaps below show an interesting observation, pipelining does enhance performance in most cases. The following is a summary of the pipelining results:

- All Inline
  - Base performance
  - Increase in record hashing or signing time results in equal impact on performance.
- Pipeline Signing
  - Better overall performance
  - Increase in signing time results in less performance degradation than increase in hashing time due to parallelism
- Pipeline Signing & Hashing
  - Slight performance improvement from the signing-only pipelining
  - Equal impact of increase in hashing and signing time on the total duration.

- Pipeline All
    - Significantly better performance.
    - Performance is slightly better when hashing and signing time are similar

5.2.5.1 Comparison Of Short (X) Duration Records

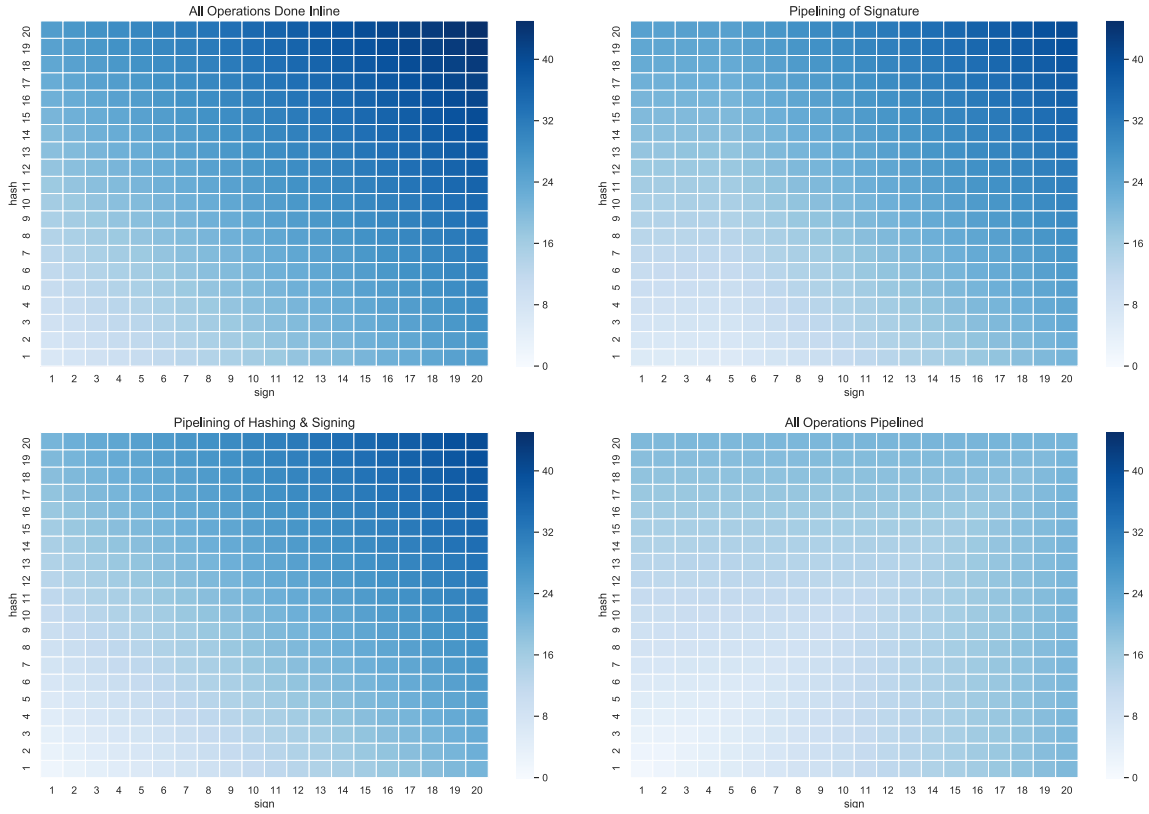


Figure 70 - Comparison of Short Transactions Heatmap

5.2.5.2 Comparison of Long (4X) Duration Records

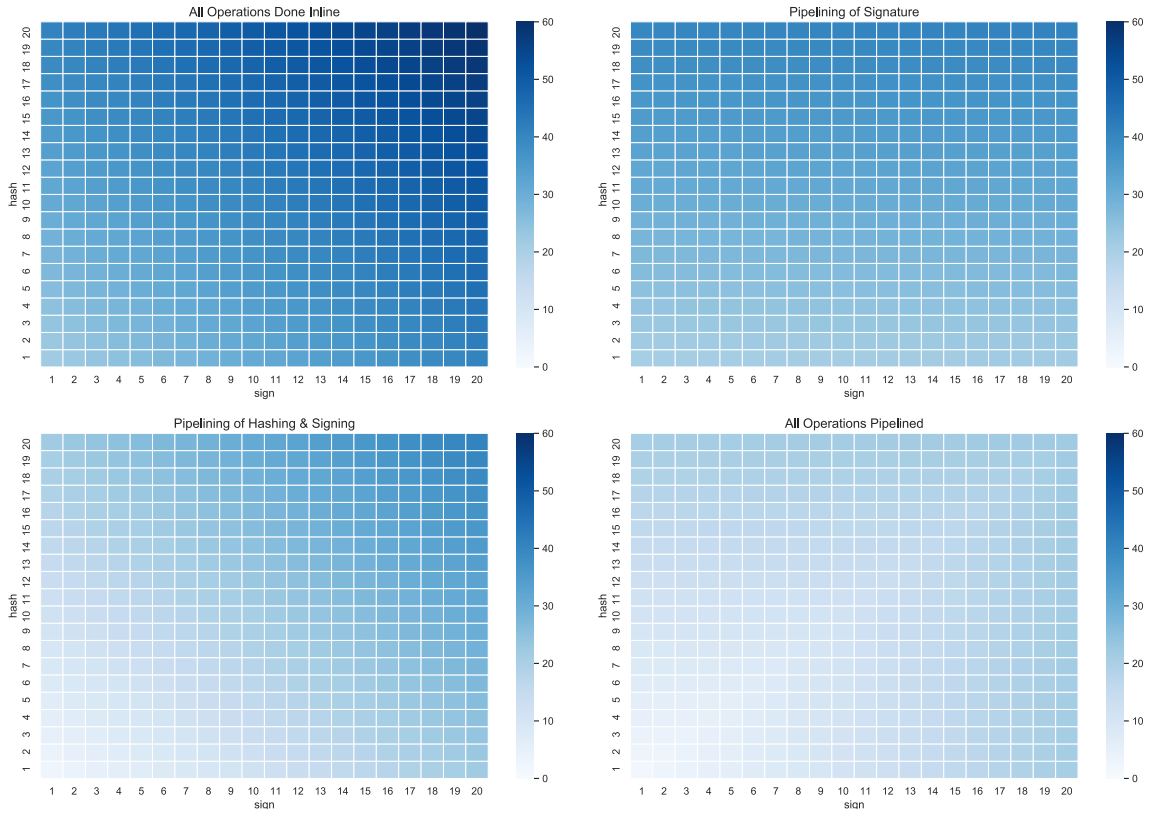


Figure 71 - Comparison of Long Transactions Heatmap

5.2.6 Experimentation Results:

This sections results of mass insert of stock market quote data.

Tests are performed by inserting 1K, 10K, 100K, and 1M records.

Please note that all experiments are done in the sequential version of DBKnot without the usage of any pipelining or parallelism techniques.

The chart below shows the relative time taken to hash and sign transactions during transaction time.

As we can see, the hashing time is almost the same as the transaction time almost doubling the overall transaction time. With more data being inserted however, the overhead per transaction is not increased. In fact, according to the charts below, the overhead slightly declines with more data.

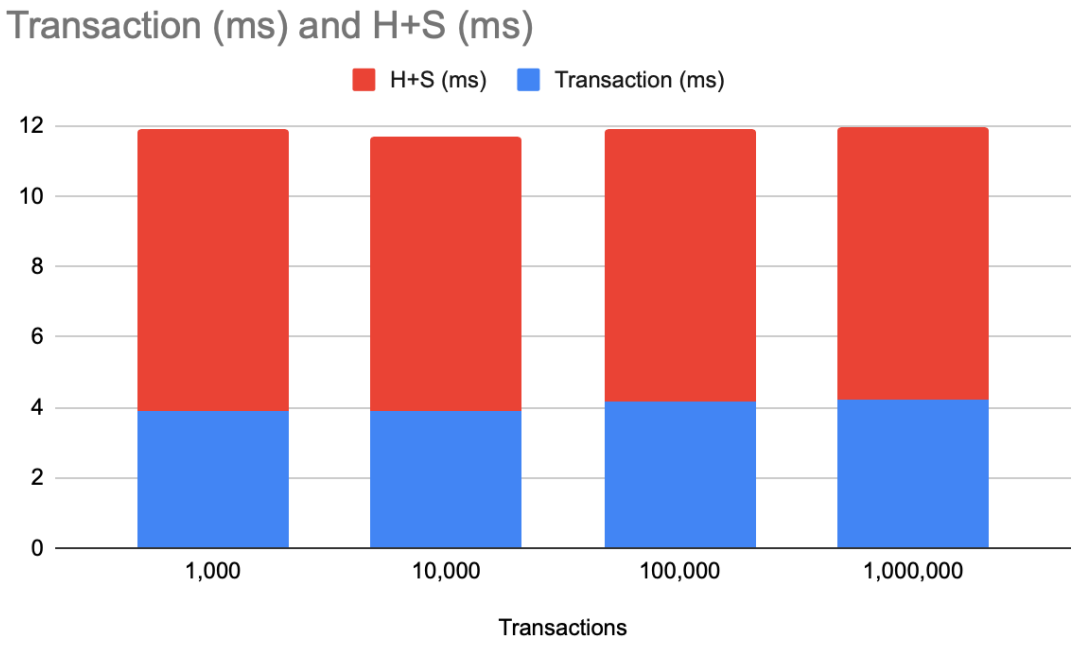
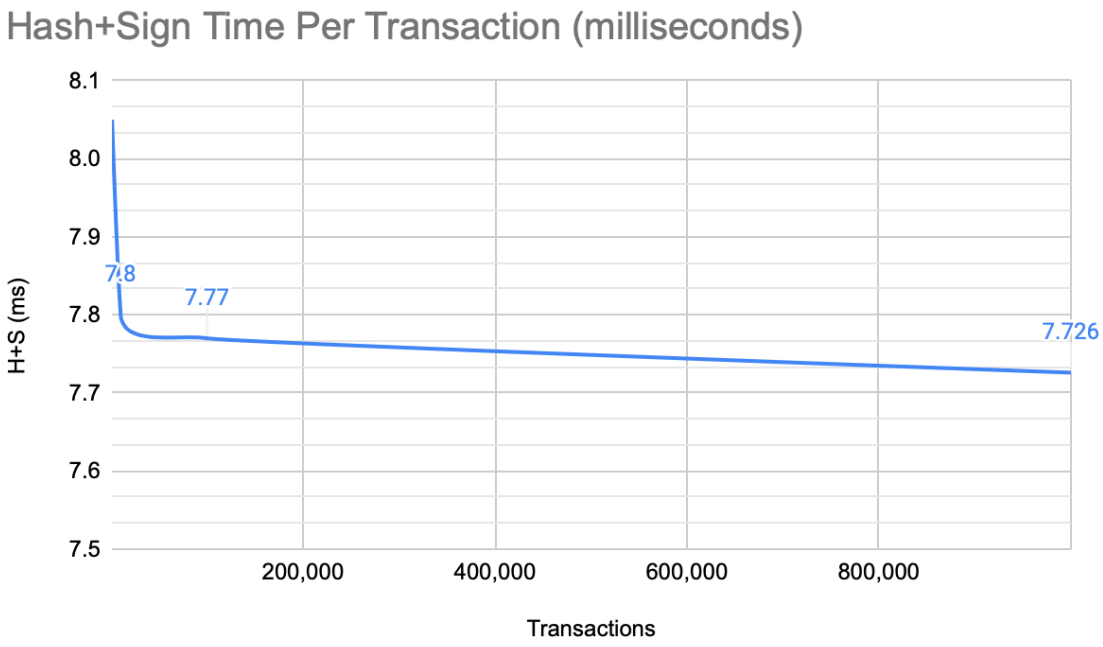
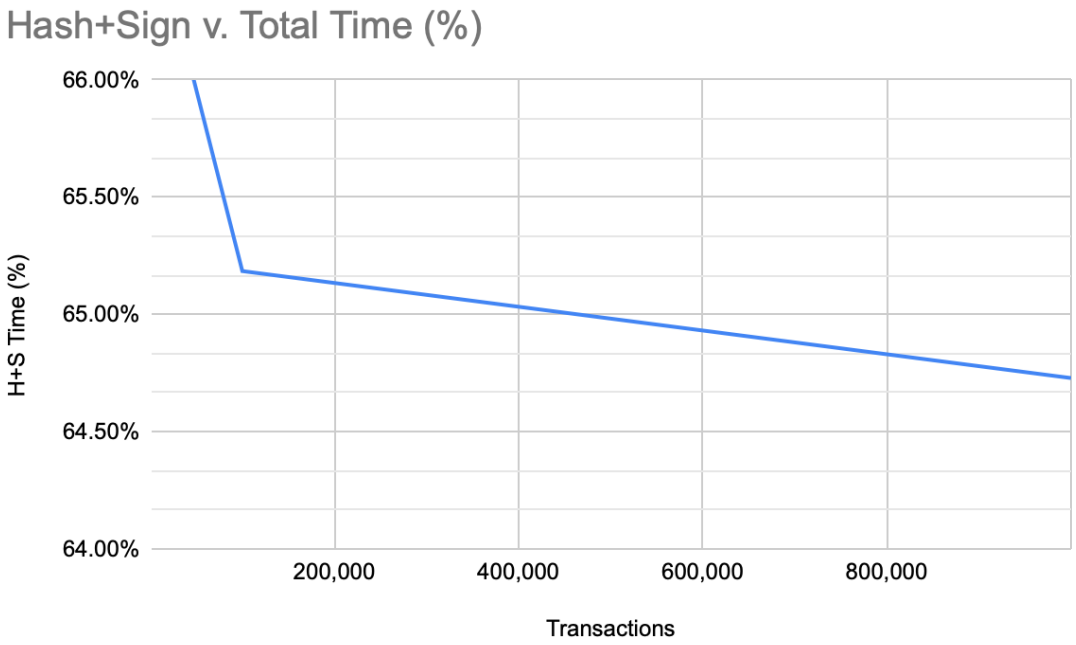


Figure 72 - Stocks Comparison of Transaction vs. Signing Time in milliseconds

The chart below shows the declining transaction hashing and signing time.



The graph below shows the same declining trend but as a percentage of the overall transaction time.


Hash+Sign v. Total Time (%)
66.00%
65.50%
65.00%
64.50%
64.00%
H+S Time (%)
200,000
400,000
600,000
800,000
Transactions

6 Conclusions and Future Work

As a conclusion, and after going through related work in the same area, we believe we have added a new solution for tamper detection for a certain class of problems. The solution is designed to be very lightweight, easy to retrofit into existing systems, as well as adding almost zero steps requiring handing data either in transit or in new storages.

We designed a tamper-evident architecture called DBKnot that detects database tampering in most cases. An external signer is being used to further protect the database from tampering even by an insider who has full authority and access rights over the whole system, including operating systems, databases, network, and firewalls. DBKnot enables tracking of individual tables that are immutable – like accounting systems, banking systems, and system logs. A chain of records inspired by blockchain is used to interlink records together through linking their hashes. Each hash link is signed using an external signer or a hardware-security-module.

We showed how the techniques could apply in 3 different modes of integration: 1) Embed inside a database management system, 2) embed inside an Object Relational Mapping framework, or 3) Implement as an external reverse-proxy for multiple web-services and even multiple totally different servers.

We have performed tests using publicly available stockmarket data. As expected, the tests showed an increased overhead for the hashing and sighing operations. The overhead though was almost constant when prorated to a transaction level, meaning that it would scale up with the same level of performance.

Performance overhead could be significantly reduced by using different parallelization and pipelining techniques to reduce to reduce the synchronicity of hashing and signing.

We have explored different parallelization by testing 4 techniques of parallelization. The first approach was zero parallelization where everything is run in series, and then incrementally started parallelizing step by step until we reached an all parallel scenario. The testing showed that parallelization will lead to a significant performance leap.

The following are some areas that could be enhanced or features that could be added in upcoming related work:

The current work assumes that data being tracked is immutable. Further work can be done by finding different techniques or approaches that would enable catering to database systems that change through updates and deletes with reasonable optimality while utilizing the same technique of relying on external signers for security against internal tampering.

The area of Merkel Trees could be studied further. Verification algorithms utilizing a Merkel-Tree like approach could result in more efficient verification of tracked records.

More studies need to be done to see how the system can be adapted to changes in database structure. This would enable, not only established and mature systems in production, but also dynamic and changeable systems that are undergoing constant development.

The REST web-service approach provided to cater to microservice architectures needs to be elaborated to find the most simple implementation approach that requires little intervention in deployment.

DBKnot is designed to as much as possible detect any tampering with data inside the database. There are however two cases that are not covered. The first case is where the fraudster has access to the application source code. In this case the data is tampered in transit before reaching the database. So the database has no knowledge that the application data has been tampered with. The second vulnerability is the small window between the transaction and the hashing of the transaction. This window could be controlled (shortened or extended) by changing the signing granularity or eliminating block signing altogether and enabling per-transaction signing. It is a tradeoff between window-size and performance.

7 Appendix – Experiment Hardware Performance & Specifications

Data Drive:

```
Sdb: Data, 2T, WDC WD20EZRZ-00Z5HB0 (80.00A80)
sudo hdparm -tT /dev/sda
/dev/sdb:
 Timing cached reads:   25754 MB in  1.99 seconds = 12948.46 MB/sec
 Timing buffered disk reads: 472 MB in  3.00 seconds = 157.19 MB/sec


Sdc: System, 256G, Samsung SSD 850 EVO 250GB (EMT03B6Q)
/dev/sdc:
 Timing cached reads:   21802 MB in  1.99 seconds = 10956.45 MB/sec
 Timing buffered disk reads: 1480 MB in  3.00 seconds = 493.24 MB/sec
```

System Drive:

```
Sdc: System, 256G, Samsung SSD 850 EVO 250GB (EMT03B6Q)
/dev/sdc:
 Timing cached reads:   21802 MB in  1.99 seconds = 10956.45 MB/sec
 Timing buffered disk reads: 1480 MB in  3.00 seconds = 493.24 MB/sec
```

CPU:

/proc/cpuinfo

```
processor    : 0
vendor_id    : GenuineIntel
cpu family    : 6
model        : 60
model name    : Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
stepping    : 3
microcode    : 0x27
cpu MHz    : 1377.436
cache size    : 8192 KB
physical id    : 0
siblings    : 8
core id        : 0
cpu cores    : 4
apicid        : 0
```

```
initial apicid    : 0
fpu               : yes
fpu_exception     : yes
cpuid level       : 13
wp                : yes
flags             : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
                    pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
                    rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology
                    nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx
                    smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic
                    movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm
                    cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi
                    flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
                    invpcid xsaveopt dtherm ida arat pln pts md_clear flush_l1d
bugs              : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs
                    itlb_multihit
bogomips          : 7184.00
clflush size      : 64
cache_alignment   : 64
address sizes     : 39 bits physical, 48 bits virtual
power management:

processor         : 1
vendor_id         : GenuineIntel
cpu family        : 6
model             : 60
model name        : Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
stepping          : 3
microcode         : 0x27
cpu MHz           : 1678.534
cache size        : 8192 KB
physical id       : 0
siblings          : 8
core id           : 1
cpu cores         : 4
apicid            : 2
initial apicid    : 2
fpu               : yes
fpu_exception     : yes
cpuid level       : 13
wp                : yes
flags             : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
                    pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
                    rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology
                    nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx
```

```
                smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic
                movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm
                cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi
                flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
                invpcid xsaveopt dtherm ida arat pln pts md_clear flush_l1d
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs
                itlb_multihit
bogomips    : 7184.00
clflush size    : 64
cache_alignment    : 64
address sizes    : 39 bits physical, 48 bits virtual
power management:

processor    : 2
vendor_id    : GenuineIntel
cpu family    : 6
model        : 60
model name    : Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
stepping    : 3
microcode    : 0x27
cpu MHz    : 1517.972
cache size    : 8192 KB
physical id    : 0
siblings    : 8
core id        : 2
cpu cores    : 4
apicid        : 4
initial apicid    : 4
fpu            : yes
fpu_exception    : yes
cpuid level    : 13
wp             : yes
flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
                pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
                rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology
                nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx
                smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic
                movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm
                cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi
                flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
                invpcid xsaveopt dtherm ida arat pln pts md_clear flush_l1d
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs
                itlb_multihit
bogomips    : 7184.00
clflush size    : 64
```

```
cache_alignment    : 64
address sizes    : 39 bits physical, 48 bits virtual
power management:

processor    : 3
vendor_id    : GenuineIntel
cpu family    : 6
model        : 60
model name    : Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
stepping    : 3
microcode    : 0x27
cpu MHz    : 1419.534
cache size    : 8192 KB
physical id    : 0
siblings    : 8
core id        : 3
cpu cores    : 4
apicid        : 6
initial apicid    : 6
fpu            : yes
fpu_exception    : yes
cpuid level    : 13
wp            : yes
flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
                 pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
                 rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology
                 nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx
                 smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic
                 movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm
                 cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi
                 flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
                 invpcid xsaveopt dtherm ida arat pln pts md_clear flush_l1d
bugs           : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs
                 itlb_multihit
bogomips    : 7184.00
clflush size    : 64
cache_alignment    : 64
address sizes    : 39 bits physical, 48 bits virtual
power management:

processor    : 4
vendor_id    : GenuineIntel
cpu family    : 6
model        : 60
model name    : Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
```

```
stepping    : 3
microcode    : 0x27
cpu MHz     : 1535.382
cache size    : 8192 KB
physical id    : 0
siblings    : 8
core id        : 0
cpu cores    : 4
apicid         : 1
initial apicid    : 1
fpu            : yes
fpu_exception    : yes
cpuid level    : 13
wp             : yes
flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
                 pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
                 rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology
                 nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx
                 smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic
                 movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm
                 cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi
                 flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
                 invpcid xsaveopt dtherm ida arat pln pts md_clear flush_l1d
bugs           : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs
                 itlb_multihit
bogomips    : 7184.00
clflush size    : 64
cache_alignment    : 64
address sizes    : 39 bits physical, 48 bits virtual
power management:

processor    : 5
vendor_id    : GenuineIntel
cpu family    : 6
model        : 60
model name    : Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
stepping    : 3
microcode    : 0x27
cpu MHz     : 1377.134
cache size    : 8192 KB
physical id    : 0
siblings    : 8
core id        : 1
cpu cores    : 4
apicid         : 3
```

```
initial apicid    : 3
fpu               : yes
fpu_exception     : yes
cpuid level       : 13
wp                : yes
flags             : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
                    pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
                    rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology
                    nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx
                    smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic
                    movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm
                    cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi
                    flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
                    invpcid xsaveopt dtherm ida arat pln pts md_clear flush_l1d
bugs              : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs
                    itlb_multihit
bogomips          : 7184.00
clflush size      : 64
cache_alignment   : 64
address sizes     : 39 bits physical, 48 bits virtual
power management:

processor         : 6
vendor_id         : GenuineIntel
cpu family        : 6
model             : 60
model name        : Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
stepping          : 3
microcode         : 0x27
cpu MHz           : 1369.484
cache size        : 8192 KB
physical id       : 0
siblings          : 8
core id           : 2
cpu cores         : 4
apicid            : 5
initial apicid    : 5
fpu               : yes
fpu_exception     : yes
cpuid level       : 13
wp                : yes
flags             : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
                    pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
                    rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology
                    nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx
```

```
                smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic
                movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm
                cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi
                flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
                invpcid xsaveopt dtherm ida arat pln pts md_clear flush_l1d
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs
                itlb_multihit
bogomips    : 7184.00
clflush size    : 64
cache_alignment    : 64
address sizes    : 39 bits physical, 48 bits virtual
power management:

processor    : 7
vendor_id    : GenuineIntel
cpu family    : 6
model        : 60
model name    : Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
stepping    : 3
microcode    : 0x27
cpu MHz    : 1355.393
cache size    : 8192 KB
physical id    : 0
siblings    : 8
core id        : 3
cpu cores    : 4
apicid        : 7
initial apicid    : 7
fpu            : yes
fpu_exception    : yes
cpuid level    : 13
wp             : yes
flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
                pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
                rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology
                nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx
                smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic
                movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm
                cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi
                flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
                invpcid xsaveopt dtherm ida arat pln pts md_clear flush_l1d
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs
                itlb_multihit
bogomips    : 7184.00
clflush size    : 64
```

```
cache_alignment    : 64
address sizes    : 39 bits physical, 48 bits virtual
```

Bibliography

[1] "Report to the Nations - 2018 Global Study on Occupational Fraud and Abuse," Association of Certified Fraud Examiners, 2019. Accessed: Apr. 17, 2019. [Online]. Available: https://www.acfe.com/report-to-the-nations/behind-the-numbers/.

[2] D. M. Upton and S. Creese, "The Danger from Within," *Harvard Business Review*, no. September 2014, Sep. 01, 2014.

[3] "Cost of Cibercrime - Accenture." Accessed: Nov. 07, 2019. [Online]. Available: https://www.accenture.com/_acnmedia/pdf-96/accenture-2019-cost-of-cybercrime-study-final.pdf.

[4] Weltwirtschaftsforum and Zurich Insurance Group, *Global risks 2019: insight report*. 2019.

[5] "RFC 4810 - Long-Term Archive Service Requirements." https://datatracker.ietf.org/doc/rfc4810/ (accessed May 01, 2019).

[6] K. Zeng, "Publicly Verifiable Remote Data Integrity," in *Information and Communications Security*, 2008, pp. 419–434.

[7] "Gramm-Leach-Bliley Act," *Federal Trade Commission*. https://www.ftc.gov/tips-advice/business-center/privacy-and-security/gramm-leach-bliley-act (accessed Oct. 12, 2019).

[8] M. G. Oxley, "H.R.3763 - 107th Congress (2001-2002): Sarbanes-Oxley Act of 2002," Jul. 30, 2002. https://www.congress.gov/bill/107th-congress/house-bill/3763 (accessed Oct. 12, 2019).

[9] O. for C. Rights (OCR), "Summary of the HIPAA Security Rule," *HHS.gov*, Nov. 20, 2009. https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html (accessed Oct. 12, 2019).

[10] J. D. Ultra and S. Pancho-Festin, "A simple model of separation of duty for access control models," *Comput. Secur.*, vol. 68, pp. 69–80, Jul. 2017, doi: 10.1016/j.cose.2017.03.012.

[11] "Control System Firewall | CISA." https://us-cert.cisa.gov/ics/Control_System_Firewall-Definition.html (accessed Aug. 21, 2020).

[12] "Firewall (computing)," *Wikipedia*. Aug. 15, 2020, Accessed: Aug. 21, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Firewall_(computing)&oldid=973146168.

[13] "What Is a Firewall?," *www.kaspersky.com*, Mar. 24, 2020. https://www.kaspersky.com/resource-center/definitions/firewall (accessed Aug. 21, 2020).

[14] "What is a Firewall?," *Forcepoint*, Aug. 09, 2018. https://www.forcepoint.com/cyber-edu/firewall (accessed Aug. 21, 2020).

[15] "About firewalls." https://kb.iu.edu/d/aoru (accessed Aug. 21, 2020).

[16] "What is a firewall and do you need one?" https://us.norton.com/internetsecurity-emerging-threats-what-is-firewall.html (accessed Aug. 21, 2020).

[17] K. Pavlou and R. Snodgrass, "DRAGOON: An Information Accountability System for High-Performance Databases," *Proc. - Int. Conf. Data Eng.*, pp. 1329–1332, Apr. 2012, doi: 10.1109/ICDE.2012.139.

[18] "Security by Design Principles - OWASP." https://www.owasp.org/index.php/Security_by_Design_Principles (accessed May 01, 2019).

[19] "Fundamentals of Database Systems, 7th Edition." /content/one-dot-com/one-dot-com/us/en/higher-education/program.html (accessed Jul. 26, 2020).

[20] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *ACM Commun.*, vol. 13, no. 6, p. 11, 1970.

[21] C. Mohan, “History repeats itself: sensible and NonsenSQL aspects of the NoSQL hoopla,” in *Proceedings of the 16th International Conference on Extending Database Technology - EDBT ’13*, Genoa, Italy, 2013, pp. 11–16, doi: 10.1145/2452376.2452378.
[22] A. Nayak, A. Poriya, and D. Poojary, “Article: Type of nosql databases and its comparison with relational databases,” *Int. J. Appl. Inf. Syst.*, vol. 5, pp. 16–19, Jan. 2013.
[23] “Neo4j Graph Platform – The Leader in Graph Databases,” *Neo4j Graph Database Platform*. https://neo4j.com/ (accessed Jul. 26, 2020).
[24] “SQLAlchemy - The Database Toolkit for Python.” https://www.sqlalchemy.org/ (accessed Jul. 26, 2020).
[25] “Django ORM.” https://www.fullstackpython.com/django-orm.html (accessed Jul. 26, 2020).
[26] “PostgreSQL: The world’s most advanced open source database.” https://www.postgresql.org/ (accessed Jul. 26, 2020).
[27] “Multi-Master Replication.” https://www.percona.com/doc/percona-xtradb-cluster/LATEST/features/multimaster-replication.html (accessed Aug. 21, 2020).
[28] “Multi-Master Replication.” https://www.percona.com/doc/percona-xtradb-cluster/LATEST/features/multimaster-replication.html (accessed Aug. 21, 2020).
[29] “Optimistic concurrency control,” *Wikipedia*. Jun. 20, 2020, Accessed: Aug. 21, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Optimistic_concurrency_control&oldid=963488784.
[30] “The Difference Between MySQL Multi-Master and Multi-Source Replication | Severalnines.” https://severalnines.com/database-blog/difference-between-mysql-multi-master-and-multi-source-replication (accessed Aug. 21, 2020).
[31] “What is database sharding?,” *Educative: Interactive Courses for Software Developers*. /edpresso/what-is-database-sharding?aid=5082902844932096&utm_source=google&utm_medium=cpc&utm_campaign=edpresso-dynamic&gclid=Cj0KCQjwg8n5BRCdARIsALxKb95Djg5UFkI-FgkeOHWlnLmP5fS6jCU20mhVgaNqC_PbXbLHZbyIaikaAlBJEALw_wcB (accessed Aug. 12, 2020).
[32] “Hadoop - HDFS Overview - Tutorialspoint.” https://www.tutorialspoint.com/hadoop/hadoop_hdfs_overview.htm (accessed Aug. 21, 2020).
[33] “Apache Hadoop 3.3.0 – HDFS Architecture.” https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html (accessed Aug. 21, 2020).
[34] “What is HDFS? Apache Hadoop Distributed File System,” Aug. 26, 2019. https://www.ibm.com/analytics/hadoop/hdfs (accessed Aug. 21, 2020).
[35] “What is REST.” https://restfulapi.net/ (accessed Jul. 26, 2020).
[36] M. Belshe, M. Thomson, and R. Peon, “Hypertext Transfer Protocol Version 2 (HTTP/2).” https://tools.ietf.org/html/rfc7540 (accessed Aug. 21, 2020).
[37] “NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy,” *NGINX*. https://www.nginx.com/ (accessed Jul. 26, 2020).
[38] “Reverse Proxy Guide - Apache HTTP Server Version 2.4.” https://httpd.apache.org/docs/2.4/howto/reverse_proxy.html (accessed Jul. 26, 2020).
[39] “HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer.” http://www.haproxy.org/ (accessed Aug. 21, 2020).
[40] “squid : Optimising Web Delivery.” http://www.squid-cache.org/ (accessed Jul. 26, 2020).
[41] “Message-Queues,” May 06, 2020. https://www.ibm.com/cloud/learn/message-queues (accessed Aug. 24, 2020).

[42] "Message queue," *Wikipedia*. Aug. 15, 2020, Accessed: Aug. 24, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Message_queue&oldid=973170986.
[43] "What is message queuing? - CloudAMQP." https://www.cloudamqp.com/blog/2014-12-03-what-is-message-queuing.html (accessed Aug. 24, 2020).
[44] "What is a Message Queue?," *Amazon Web Services, Inc.* https://aws.amazon.com/message-queue/ (accessed Aug. 24, 2020).
[45] "Oracle Database Product Options." https://www.oracle.com/database/technologies/ (accessed Jul. 26, 2020).
[46] N. Emmadi and H. Narumanchi, "Reinforcing Immutability of Permissioned Blockchains with Keyless Signatures' Infrastructure," in *Proceedings of the 18th International Conference on Distributed Computing and Networking*, New York, NY, USA, 2017, p. 46:1–46:6, doi: 10.1145/3007748.3018280.
[47] "Blockchain," *Wikipedia*. Apr. 21, 2019, Accessed: May 01, 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Blockchain&oldid=893385950.
[48] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," p. 9.
[49] B. Cohen and K. Pietrzak, "Simple Proofs of Sequential Work," in *Advances in Cryptology – EUROCRYPT 2018*, 2018, pp. 451–467.
[50] T. M. Press, "The Blockchain and the New Architecture of Trust | The MIT Press." https://mitpress.mit.edu/books/blockchain-and-new-architecture-trust (accessed Jul. 26, 2020).
[51] A. Tar, "Proof-of-Work, Explained," *Cointelegraph*, Jan. 17, 2018. https://cointelegraph.com/explained/proof-of-work-explained (accessed May 11, 2019).
[52] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling Blockchain: A Data Processing View of Blockchain Systems," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 7, pp. 1366–1385, Jul. 2018, doi: 10.1109/TKDE.2017.2781227.
[53] M. Vukolić, "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication," in *Open Problems in Network Security*, 2016, pp. 112–125.
[54] G. Becker, "Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis," p. 28.
[55] "Merkle tree," *Wikipedia*. Apr. 04, 2019, Accessed: May 01, 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Merkle_tree&oldid=890979020.
[56] "Auditing to Keep Online Storage Services Honest." https://www.usenix.org/legacy/event/hotos07/tech/full_papers/shah/shah_html/hotos11web.html (accessed May 01, 2019).
[57] N. Singh, "IPFS and Merkle Forest," *Hacker Noon*, Apr. 15, 2018. https://hackernoon.com/ipfs-and-merkle-forest-a6b7f15f3537 (accessed May 02, 2019).
[58] M. S. Ali, K. Dolui, and F. Antonelli, "IoT Data Privacy via Blockchains and IPFS," in *Proceedings of the Seventh International Conference on the Internet of Things*, New York, NY, USA, 2017, p. 14:1–14:7, doi: 10.1145/3131542.3131563.
[59] A. Tenorio-Fornés, S. Hassan, and J. Pavón, "Open Peer-to-Peer Systems over Blockchain and IPFS: An Agent Oriented Framework," in *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*, New York, NY, USA, 2018, pp. 19–24, doi: 10.1145/3211933.3211937.
[60] *ipfs/ipfs*. IPFS, 2020.
[61] Juan Benet, "IPFS - Content Addressed, Versioned, P2P File System," *GitHub*. https://github.com/ipfs/ipfs (accessed Jul. 26, 2020).
[62] "What is Data Governance? - LightsOnData," *LightsOnData*, Jan. 2018, [Online]. Available: http://www.lightsondata.com/what-is-data-governance/.
[63] "Agile/Lean Data Governance: Proven Strategies." http://agiledata.org/essays/dataGovernance.html (accessed Nov. 07, 2019).

[64] D. Y. Chan and M. A. Vasarhelyi, "Innovation and practice of continuous auditing," *Int. J. Account. Inf. Syst.*, vol. 12, no. 2, pp. 152–160, Jun. 2011, doi: 10.1016/j.accinf.2011.01.001.
[65] M. A. Vasarhelyi, M. G. Alles, and A. Kogan, "Principles of Analytic Monitoring for Continuous Assurance," *J. Emerg. Technol. Account.*, vol. 1, no. 1, pp. 1–21, Dec. 2004, doi: 10.2308/jeta.2004.1.1.1.
[66] K. E. Pavlou and R. T. Snodgrass, "Forensic Analysis of Database Tampering," *ACM Trans Database Syst*, vol. 33, no. 4, p. 30:1–30:47, Dec. 2008, doi: 10.1145/1412331.1412342.
[67] K. E. Pavlou and R. T. Snodgrass, "Forensic Analysis of Database Tampering," *ACM Trans Database Syst*, vol. 33, no. 4, p. 30:1–30:47, Dec. 2008, doi: 10.1145/1412331.1412342.
[68] "Amazon QLDB," *Amazon Web Services, Inc.* https://aws.amazon.com/qldb/ (accessed May 02, 2019).
[69] "BigchainDB 2.0 Whitepaper • • BigchainDB," *BigchainDB*. https://www.bigchaindb.com/whitepaper/ (accessed May 11, 2019).
[70] "BigchainDB 2.0 is Byzantine Fault Tolerant - The BigchainDB Blog." https://blog.bigchaindb.com/bigchaindb-2-0-is-byzantine-fault-tolerant-5ffdac96bc44 (accessed Dec. 09, 2019).
[71] R. Hasan, R. Sion, and M. Winslett, "The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance," in *Proccedings of the 7th Conference on File and Storage Technologies*, Berkeley, CA, USA, 2009, pp. 1–14, Accessed: Oct. 11, 2019. [Online]. Available: http://dl.acm.org/citation.cfm?id=1525908.1525909.
[72] "Oracle Label Security Whitepaper." Accessed: Mar. 12, 2018. [Online]. Available: http://www.oracle.com/technetwork/wp-dbsec-ols-201702-3634252.pdf.
[73] D. Terry, D. Goldberg, D. Nichols, and B. Oki, "Continuous Queries over Append-only Databases," in *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 1992, pp. 321–330, doi: 10.1145/130283.130333.
[74] "Designing better file organization around tags, not hierarchies." https://www.nayuki.io/page/designing-better-file-organization-around-tags-not-hierarchies#git-version-control (accessed Oct. 12, 2019).
[75] J. Zhang, A. Chapman, and K. Lefevre, "Do you know where your data's been?--Tamper-evident database provenance," in *Workshop on Secure Data Management*, 2009, pp. 17–32, Accessed: Oct. 11, 2019. [Online]. Available: https://eprints.soton.ac.uk/411445/.
[76] "A Robust Tamperproof Watermarking for Data Integrity in Relational Databases," *Science Alert*. https://scialert.net/fulltext/?doi=rjit.2009.115.121 (accessed Dec. 13, 2019).
[77] G. Miklau and D. Suciu, "Implementing a Tamper-Evident Database System," in *Advances in Computer Science – ASIAN 2005. Data Management on the Web*, 2005, pp. 28–48.
[78] L. Lavaire, "Immutable systems: how they work and why should we care.," *Medium*, Jul. 10, 2019. https://medium.com/nitrux/immutable-systems-how-they-work-and-why-should-we-care-39e567a59f28 (accessed Aug. 12, 2020).
[79] "Welcome to Flatpak's documentation! — Flatpak documentation." https://docs.flatpak.org/en/latest/ (accessed Aug. 12, 2020).
[80] "Snapcraft - Snaps are universal Linux packages," *Snapcraft*. https://snapcraft.io/ (accessed Aug. 12, 2020).
[81] "Canonical's Snap: The Good, the Bad and the Ugly," *The New Stack*, Jul. 07, 2016. https://thenewstack.io/canonicals-snap-great-good-bad-ugly/ (accessed Aug. 12, 2020).
[82] "OSTree." https://ostree.readthedocs.io/en/latest/ (accessed Aug. 12, 2020).
[83] By, "What's The Deal With Snap Packages?," *Hackaday*, Jun. 24, 2020. https://hackaday.com/2020/06/24/whats-the-deal-with-snap-packages/ (accessed Aug. 23, 2020).

[84] A. Goel, Wu-chang Feng, D. Maier, Wu-chi Feng, and J. Walpole, “Forensix: A Robust, High-Performance Reconstruction System,” in *25th IEEE International Conference on Distributed Computing Systems Workshops*, Columbus, OH, USA, 2005, pp. 155–162, doi: 10.1109/ICDCSW.2005.62.
[85] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen, “ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay,” p. 14.
[86] T. Schwarz and E. Miller, “Store, Forget, and Check: Using Algebraic Signatures to Check Remotely Administered Storage,” Feb. 2006, vol. 2006, pp. 12–12, doi: 10.1109/ICDCS.2006.80.
[87] D. Luiz Gazzoni Filho and P. Barreto, “Demonstrating Data Possession and Uncheatable Data Transfer,” *IACR Cryptol. EPrint Arch.*, vol. 2006, p. 150, Jan. 2006.
[88] F. Sebe, J. Domingo-Ferrer, A. Ballesté, Y. Deswarte, and J.-J. Quisquater, “Efficient Remote Data Possession Checking in Critical Information Infrastructures,” *Knowl. Data Eng. IEEE Trans. On*, vol. 20, pp. 1034–1038, Sep. 2008, doi: 10.1109/TKDE.2007.190647.
[89] L. Quest, A. Charrie, L. du C. de Jongh, and S. Roy, “The Risks and Benefits of Using AI to Detect Crime,” *Harvard Business Review*, Aug. 09, 2018.
[90] C. Xia, G. Yu, and M. Tang, “Efficient Implement of ORM (Object/Relational Mapping) Use in J2EE Framework: Hibernate,” Jan. 2010, pp. 1–3, doi: 10.1109/CISE.2009.5365905.
[91] “Object-relational Mappers (ORMs).” https://www.fullstackpython.com/object-relational-mappers-orms.html (accessed Aug. 23, 2020).
[92] “What is Object/Relational Mapping? - Hibernate ORM.” https://hibernate.org/orm/what-is-an-orm/ (accessed Aug. 23, 2020).
[93] “Object-relational mapping,” *Wikipedia*. Aug. 20, 2020, Accessed: Aug. 23, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Object-relational_mapping&oldid=974070664.
[94] “MD5, SHA-1, SHA-256 and SHA-512 speed performance – Automation Rhapsody.” https://automationrhapsody.com/md5-sha-1-sha-256-sha-512-speed-performance/ (accessed Aug. 23, 2020).
[95] “10 New Ways to Download Historical Stock Quotes for Free,” *QuantShare Trading Software*. https://www.quantshare.com/sa-620-10-new-ways-to-download-historical-stock-quotes-for-free (accessed Aug. 11, 2020).