A Multi-Modal, Pluggable Transaction Tamper Evident Database Architecture

Islam Khalil*, Sherif El-Kassas*, and Karim Sobh*
The American University in Cairo, Cairo, Egypt

Abstract

Fraud and data tampering is one of the key security risks of computer systems in general and in particular, sophisticated architecture that involves a wide array of heavily interdependent systems that communicate data using microservices, as well as simple normal user-facing systems.

The evolving risks of security threats as well as regulatory compliance are important driving forces for achieving better integrity and detecting any possible data tampering by either internal or external malicious perpetrators. We present the architecture for a multi-modal tamper detection solution with a primary goal of being easily retrofittable into existing systems with minimal intervention required from system developers or system administrators in large organizations. Our focus in this work is append-only databases like financial transactions, auditing systems, as well as technical system logs. We also pay attention to data confidentiality by making sure that the data never leaves the organization's premises. We leverage designs like chains of record hashes to achieve the target solution. After illustrating different ways of integrating DBKnot into existing architecture, we then go through how to leverage existing web service configuration and definition standards to increase the seamlessness and ease of retrofitting into existing applications by automatically detecting and learning about the target web service semantics without much need for manual human intervention.

Key Words: Database, security, tamper evident, chaining, lock-chain, and hash chaining.

1 Introduction

With the increasing use and ubiquity and multiple ways to use and access data across systems and as system architectures get more sophisticated and their interdependence is increasing while the range of technologies being used is widening, the need for more security and detecting fraud also increases. We propose 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.

Such systems manage information like bank transactions, medical information, government records, as well as other critical information. Such systems often fall prey to perpetrators who are insiders or collude with insiders to commit their fraud crimes. External malicious actors are in many cases the players responsible for committing fraud and tampering with sensitive databases. Many cases involve tampering with existing systems and making fraudulent transactions that go unnoticed because 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 [26], $7 Billion of losses were incurred due to internal fraud alone with an average fraud scheme going for 16 months unnoticed. Small businesses 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 [31], more than 80 million insider security breaches occur every year costing tens of billions of dollars in the US alone. In one incident $350,000 was stolen from 4 Citibank customers by employees of a software and service company that Citibank had contracted [31]. According to Accenture [6] and The World Economic Forum (WEF) [35], the cost of insider malicious activity constitutes 15% of all cybercrime. The IETF's RFC 4810 [28, 39] 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 [31]. 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 broadening complexity and use of IT solutions and its corresponding increase in the number of internal and external stakeholders needed to operate such systems.

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) [11] which mandates that companies engaging in financial services 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 [22] (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.

* Emails: {ikhalil, sherif, kmsobh}@aucegypt.edu.

This act was made as a reaction to a number of major corruption scandals including Enron and WorldCom. The Health Insurance Portability and Accountability Act [18] (HIPAA) by the US Department of Health and Human Services (HHS) is also an example which regulates access and changes to medical records.

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 [24, 126, 129].

In the process of coming up with such a solution, a number of different technologies are examined, in addition to related work.

Example append-only applications that could benefit from our proposed solutions are server security logs, banking transactions, accounting ledgers in enterprises, notary and real-estate records, birth and death records, time and attendance systems, and many others.

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 they have performed since he/she is the one responsible for all system permissions, logs, monitoring, etc.

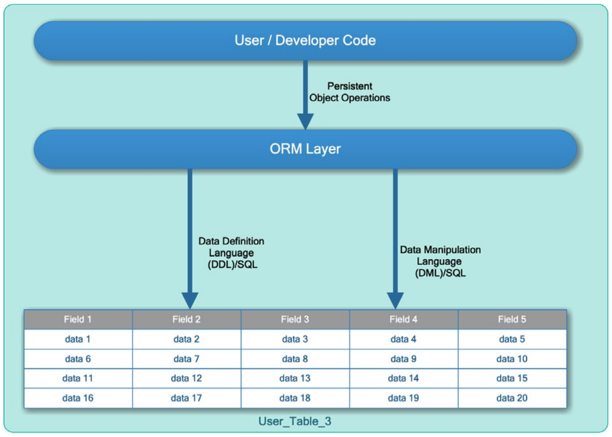
We start this paper by giving a background on some of the technologies used, then we briefly mention different related approaches to the same problem and a comparison of their corresponding features. Afterwards we go through our proposed solution, then we show some results of our experimentation followed by a conclusion. This paper is a continuation of the work done in [13].

2 Background

2.1 Object Relational Mapping (ORM)

Object Relational Mapping (ORM) frameworks [16–17, 36, 38] sit between developer applications and databases. They provide developers with full object-oriented semantics to the database allowing 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 during database creation and definition, transactions, as well as querying.

Figure 1: 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.


User / Developer Code
Persistent Object Operations
ORM Layer
Data Definition Language (DDL)/SQL
Data Manipulation Language (DML)/SQL
User_Table_3

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

Figure 1: Standard ORM operations

2.2 Web Services

Web services provide a standard mechanism of integrating different software systems or subsystems while abstracting away all implementation details and technologies. Web services usually provide the functionality to make database transactions as well as queries through formats like the REST API [37].

Figure 2: REST API request and response is an example of how web services work.

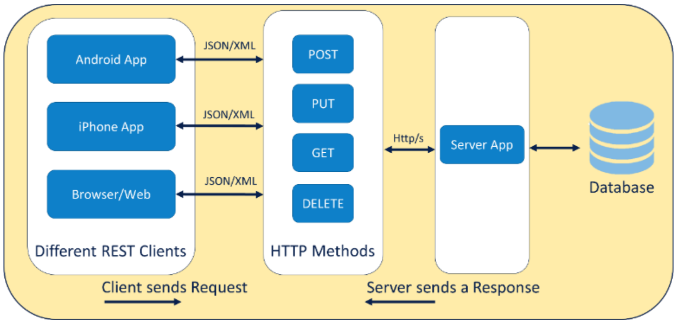


Figure 2: REST API request and response

2.2.1 REST. The definition of REST according to [27] is “Representational State Transfer”.

REST is defined to be a standardized HTTP based communication scheme for systems to invoke web services across hybrid technologies without relying on any technology specific integration and thus, decouple implementations from internal technologies.

REST depends on standard HTTP methods (GET, POST, HEAD, DELETE, etc.) and uses standard HTTP return codes to communicate meaningful responses.

Contents of a REST message are usually written in formats like JSON (a javascript notation representation of data), but also other formats could be used like XML and YAML.

2.2.2 CRUD. CRUD (Create, Read, Update, Delete) are standard database operations. They are however often mapped very closely to REST API calls [32] (REST APIs have many other non-CRUD uses as well). The concept of CRUD was coined long ago before web services APIs were used and is very database specific.

2.2.3 Scenarios of REST and CRUD Mapping. With the creation of REST, there started to be many use cases that tend to show semantic similarities between parts of the two concepts.

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. Some of them use a similar technique of chained hashes. All the solutions surveyed failed to provide a seamless and non-invasive way to get retrofitted into existing solutions with little or zero changes necessary. Another important difference is the requirement that data does not leave the users' premises.

DRAGOON [1, 23-24] 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.

Amazon Quantum Ledger Database (QLDB) [4], a blockchain based database, solves part of the problem addressed in our work.

QLDB provides the ledger database service based on the premise that there is a "central" and "trusted" authority which in this case is Amazon. In this case Amazon 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. The difference though is that it requires data to be stored at Amazon premises meaning that Amazon needs to be depended on as a trusted host of the data.

BigchainDB [12] leverages a blockchain network to provide decentralized and an immutable database. However, due to its sophisticated setup, it does not allow seamless retrofitting into existing systems.

There are other research work like [24] that focus on documents rather than data. Some of which are designed to track documents provenance throughout their lifecycle.

Several other research works have catered to a similar problem in the domains of operating systems and file systems. Examples are [5, 8, 10, 14-15, 20, 30, 34]. But most of them either depend on a local trusted administrator or use mechanisms that require data to be moved to outside the local premises.

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 in-house and within the control of the internal DBA team. This comes with the implicit assumption that the internal top-most system administrators with the 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 eliminate the need for transferring an organization's data and keep it completely in-house.
- **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, 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 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 middleware 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 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. Another goal is to provide a solution that does not impractically require total change in an underlying infrastructure.

4 Proposed Solution

4.1 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 regenerated following any tampering attempt, an external source is used for time stamped signing of hashes. The external time-stamp 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).

In our work, we propose three integration architectures. One is used for Object Relational Mapping frameworks (ORM), the second is for direct database integration, and the third is implementing microservice solutions by a totally transparent reverse proxy.

4.2 The Hasher and The Time-Stamping 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 time-stamp is also added to the signed data to protect against future signing replay attacks.

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.

4.3 Externalized Signer/Stamper

As illustrated in Figure 4, the signer is by design to be external and to serve (as a service) multiple unrelated organization. This adds more security and dramatically reduces the possibility of collusion among system administrators of all the organizations serviced by the signer.

We introduced in Figure 5, an independent signer and 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.

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- **Simplicity of design:** Reduces the range of possible attack vectors making it less vulnerable to attacks.

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 loads across many signer nodes as needed if 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).

The hasher is the first step of the process. As soon as a 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.

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 time-stamp 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 time-stamp will be

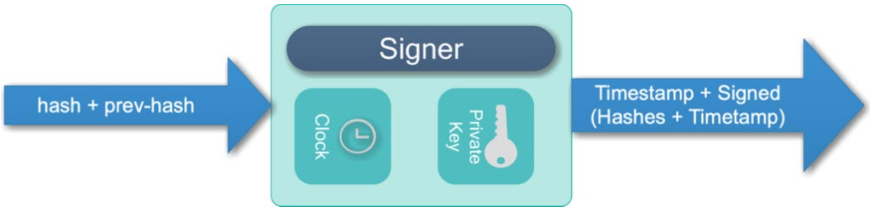


Figure 3: Signer service

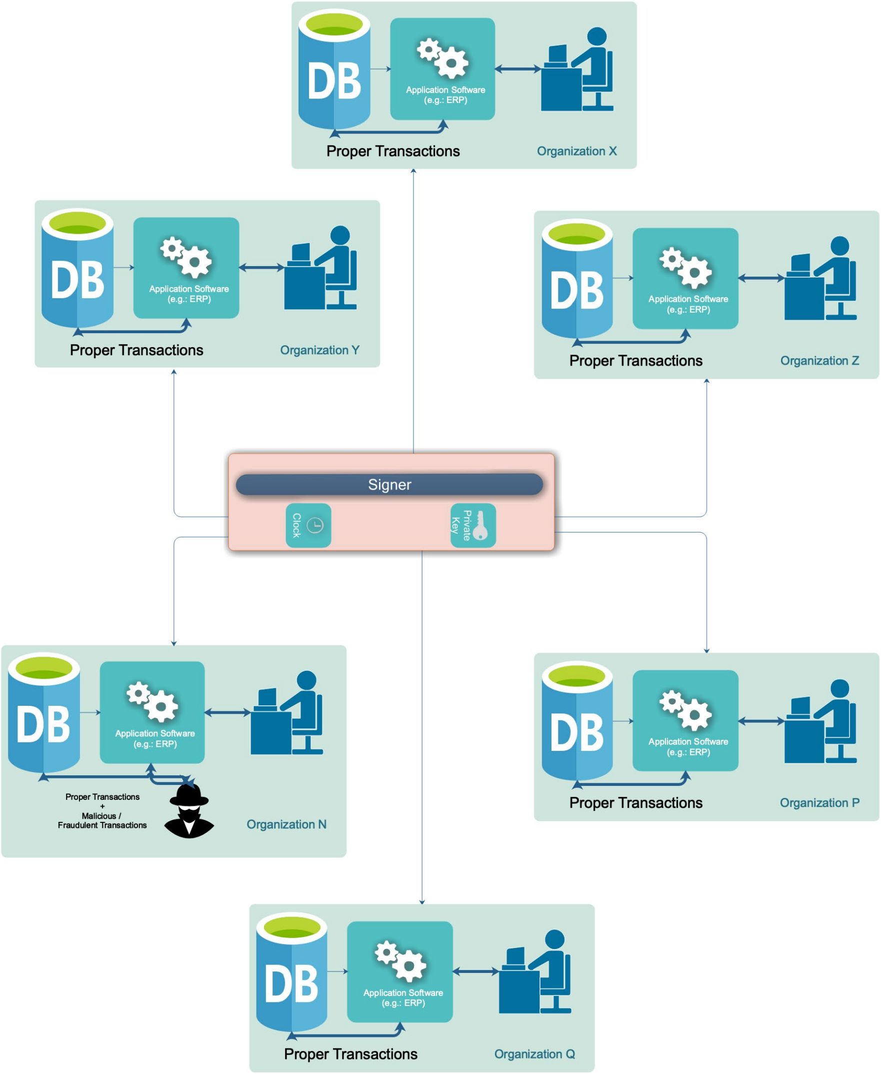


Figure 4: Detailed introduction of a third-party external signing authority

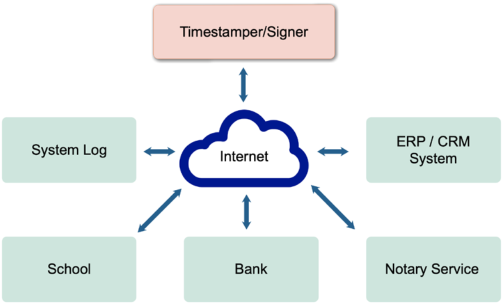


Figure 5: Introduction of a third-party signing service

returned to the database server and stored inside the hash table. The signature saved in the hash table will be used for verification.

4.4 Integration Models

Three different strategies are provided for integrating into existing systems.

The first technique is to design the DBKnot as an embedded layer inside Object Relational Mapping (ORM) systems so application developers can use it seamlessly in a declarative way as detailed below. The second approach is to implement it as a hook into existing databases. This requires less intervention from the user side and only requires an action from the database system administrator. The third and relatively more challenging approach is to be implemented in the form of a REST web service reverse proxy.

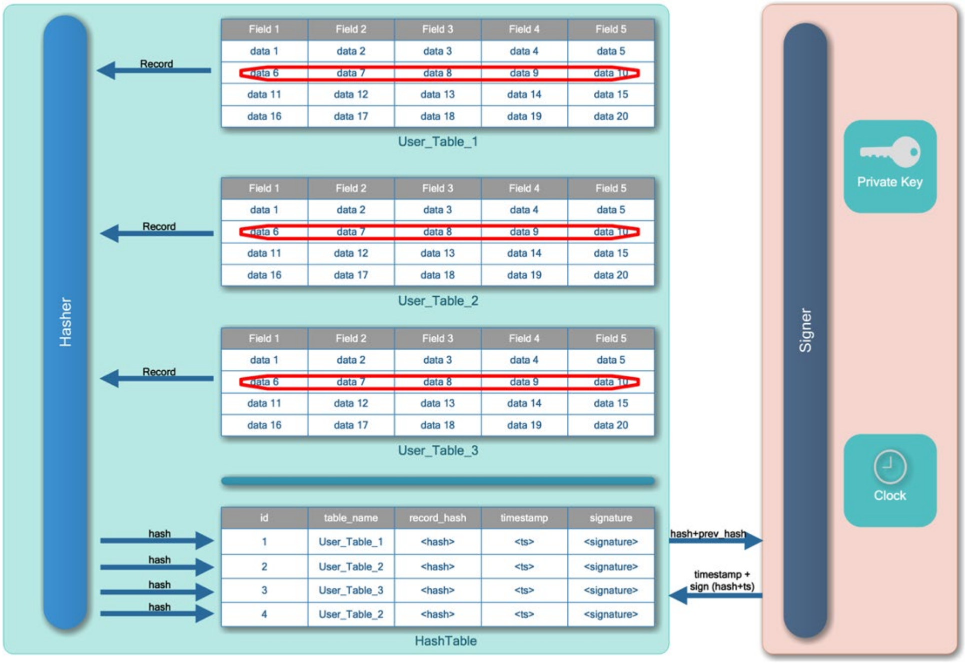
4.4.1 ORM Level Integration. Object Relational Mapping (ORM) frameworks [16-17, 36, 38] sit between developer applications and databases. They provide developers with full object oriented semantics to interfacing with the database.

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, the 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 individual "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 be reflected immediately into the database structure while preserving all data. This is a process that some implementations call "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.

In addition to the declarative semantics and ease of use by developers, embedding a tamper-detection layer inside the



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

User_Table_1

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

User_Table_2

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

User_Table_3

id	table_name	record_hash	timestamp	signature
1	User_Table_1	<hash>	<ts>	<signature>
2	User_Table_2	<hash>	<ts>	<signature>
3	User_Table_3	<hash>	<ts>	<signature>
4	User_Table_2	<hash>	<ts>	<signature>

HashTable

Figure 6: Signer and time-stamper

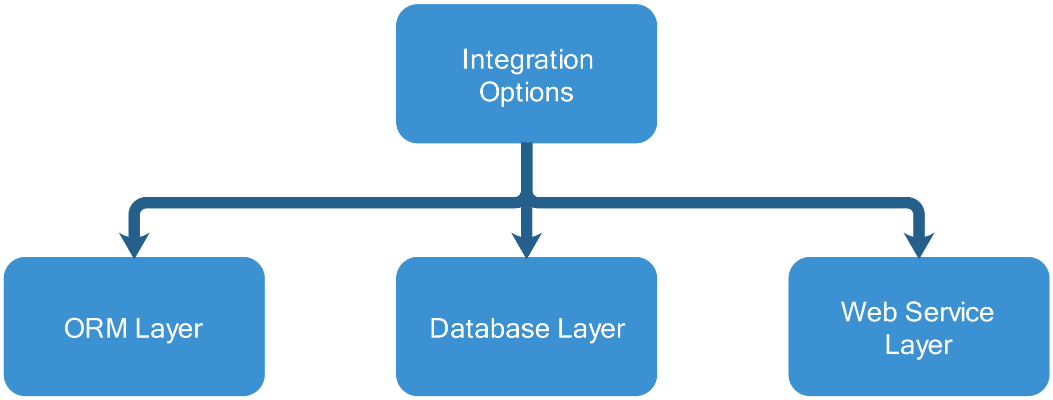


Figure 7: Integration options

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 changes.

Figure 8 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.

In the following section, two different techniques are outlined for integration into ORM systems.

The first one is through an application level ORM interceptor, and the second one is through implementing a framework level global middleware to perform the ORM functionality.

4.4.1.1 ORM Technique 1: ORM Interceptor. To retrofit DBKnot functionality into an ORM application, 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 10 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.

The integration layer is designed to provide a completely seamless user experience to developers. In the current implementation, as illustrated in Figure 11, all a user (developer) needs to do is to have his/her model classes extend a class (a mixin) that provides all needed functionality.

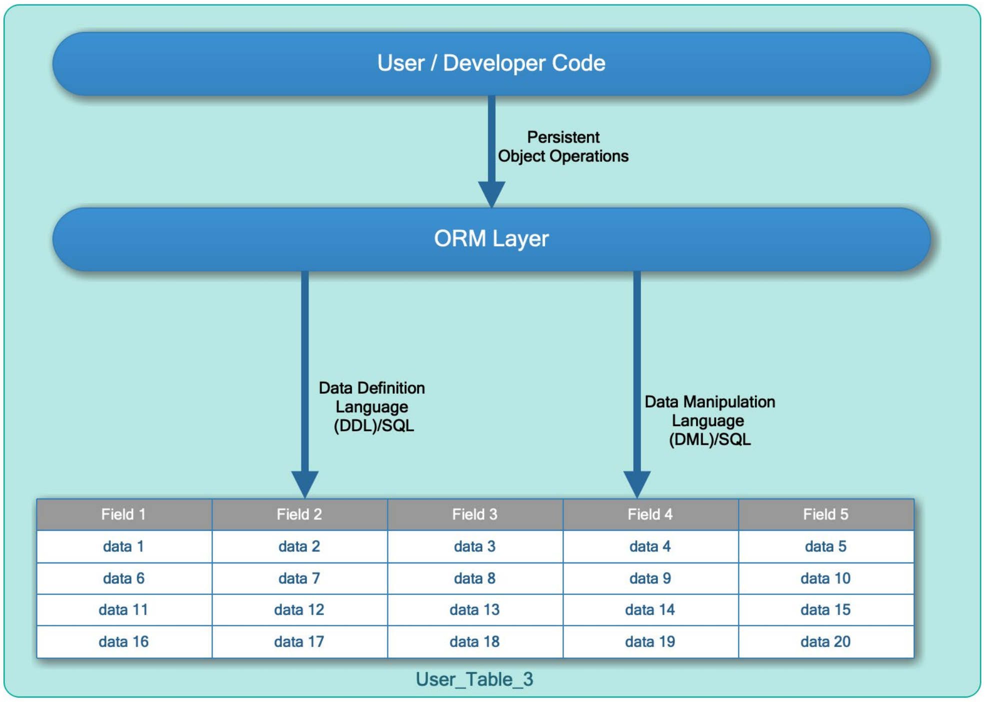
4.4.1.2 ORM Technique 2: Framework-wide Global Middleware. A second approach to integrating into ORM systems is to integrate in the form of a middleware that is embedded into the ORM framework itself. The advantage of this approach is that it is completely transparent and will not even require the declarative approach of extending a "trackable" class in system code. The side effect however of this approach is that it will give application developers less control to selectively track specific models (tables) while ignoring the tracking of other models. This could be mitigated through the implementation of an external configurator that could be managed separately to disable universal tracking and allow selective tracking of data models.

4.4.1.3 More Efficient ORM Tracking through Parallel Pipelining. The efficiency of the previously outlined ORM tracking could be increased through the introduction of a level of parallelism. Such parallelism in signing and stamping is not going to be as simple as just creating a parallel thread due to the fact that the feature of "chaining" introduces dependencies. Due to this level of dependency, a pipelining technique is introduced as detailed in Chapter 4.7.

Figure 12 shows an adapted version of the activity diagram after adding the parallel tracking.

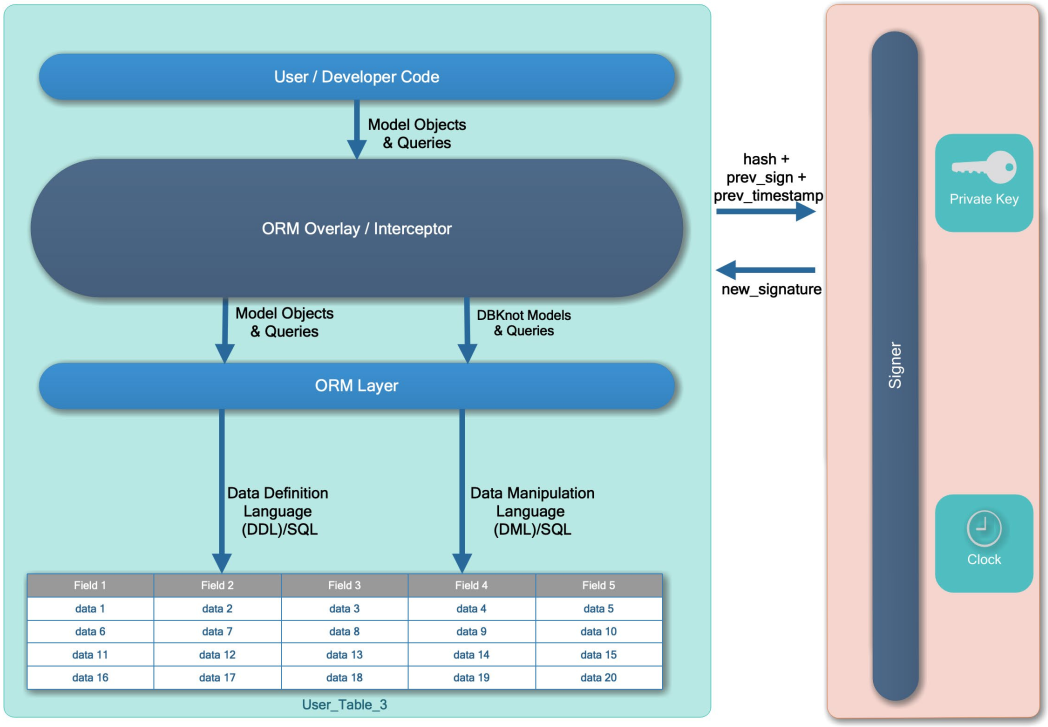
4.4.2 Database Level Integration

DBKnot also supports database level integration. This is done



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

Figure 8: Standard ORM operations



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

Figure 9: ORM interceptor

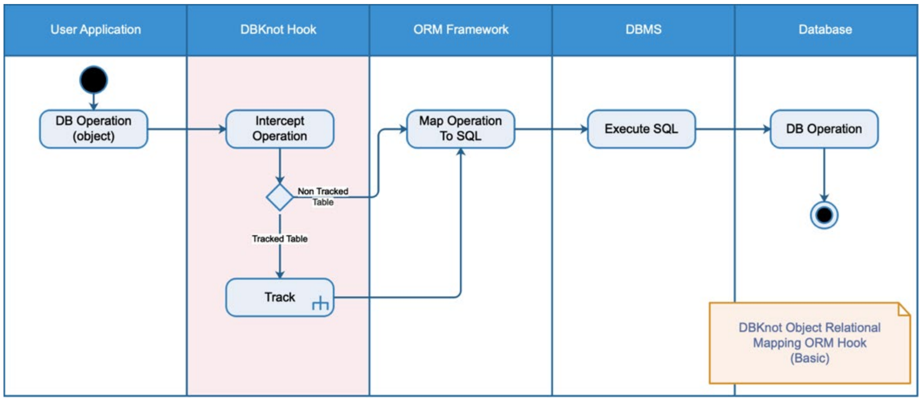


Figure 10: Adding DBKnot ORM hook basic activity diagram

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

Figure 11: ORM simple mixing implementation

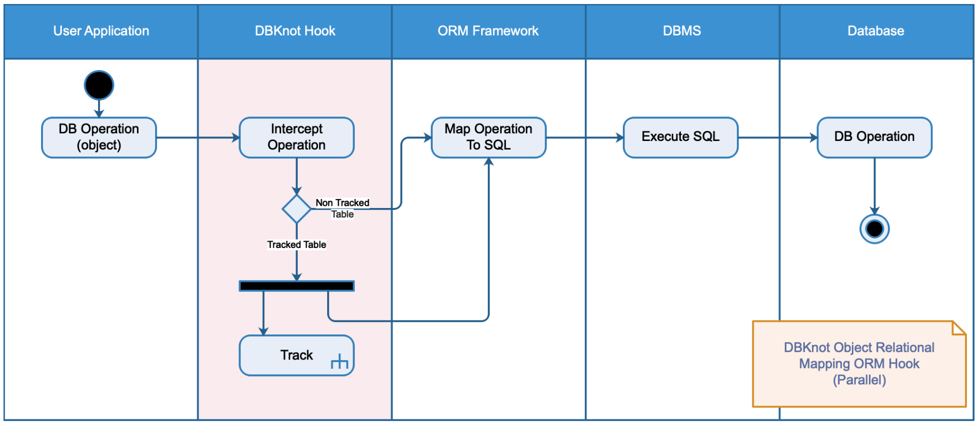


Figure 12: Adding DBKnot ORM hook parallel activity diagram

by embedding triggers on tracked tables. When a record is inserted in a tracked table, the trigger will be fired and will perform all the needed tracking functionality.

The default behavior in Figure 13 is changed by adding the DBKnot layer. The DBKnot layer is called a database trigger that tracks desired tables.

Figure 14 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.

4.4.2.1 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 time-stamp is also added to the signed data in order to protect against future signing replay attacks.

4.4.2.2 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 time-stamps. 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.

As illustrated in Figure 16, 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

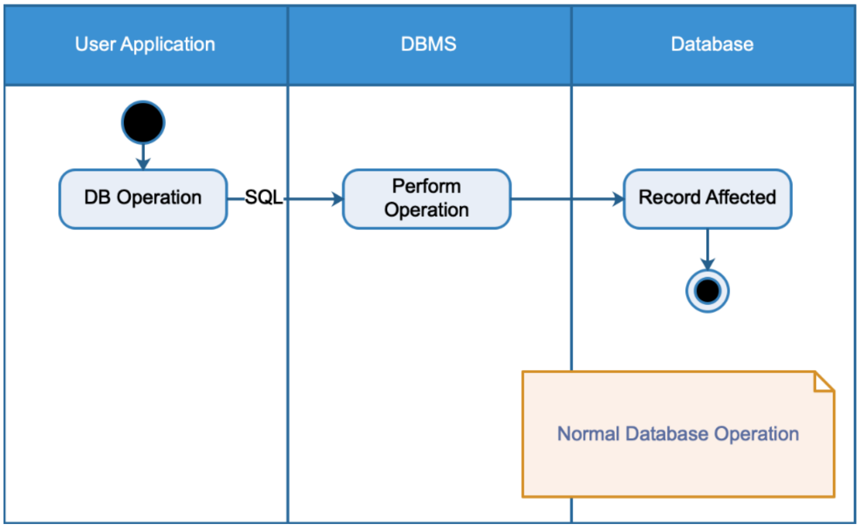


Figure 13: Normal database operation

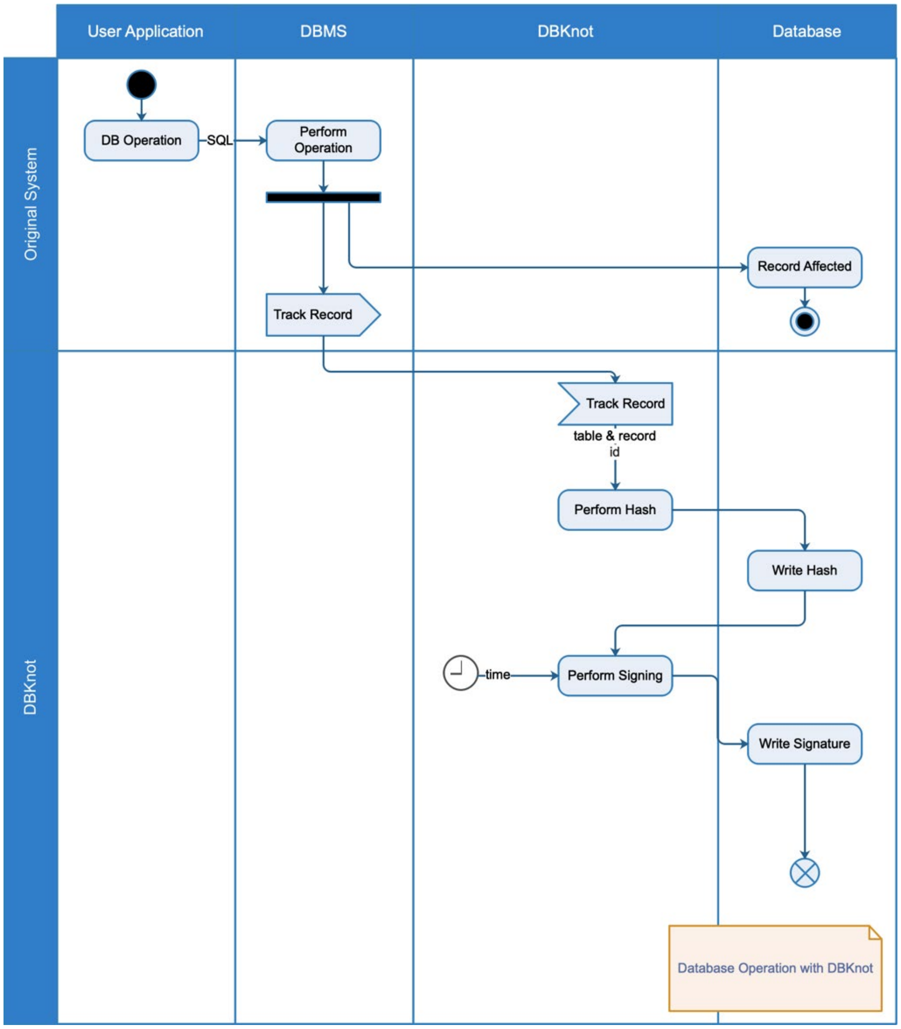


Figure 14: Database level DBKnot integration

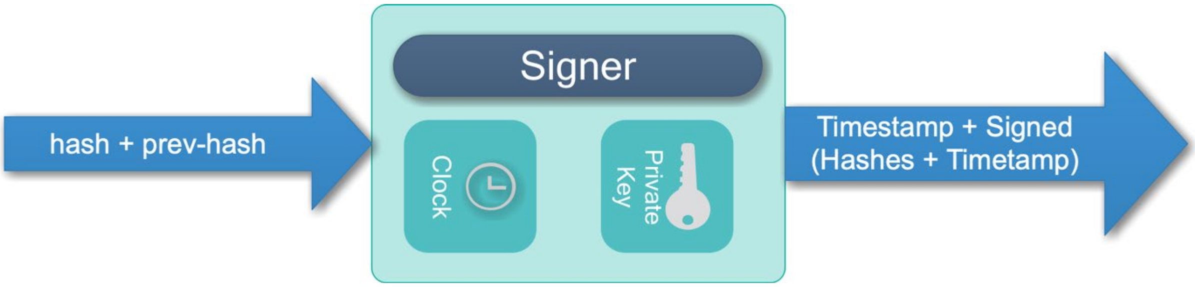


Figure 15: Signer service

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 [9, 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 in any meaningful data. Again, it could be configurable and left to be decided on a case-by-case basis.

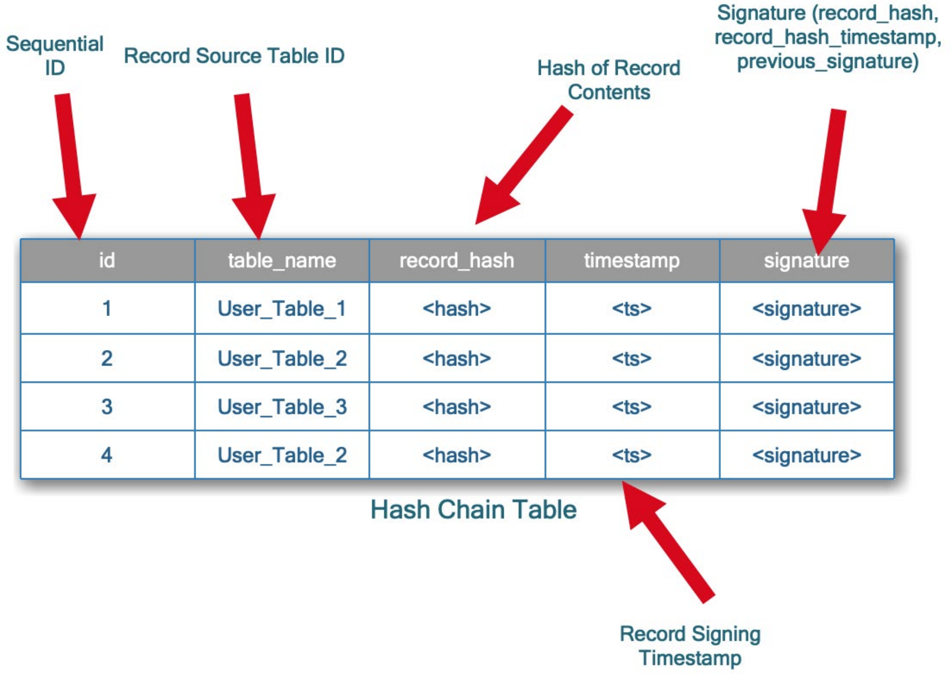
4- **Time-stamp:** This field is filled by the data returned from the signer. It is the signature time-stamp.

5- **signature:** In this field, the signature itself is stored as returned by the signer.

4.4.2.3 The Hasher. The hasher is the first step of the process. As soon as a 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 4.4.2.2.

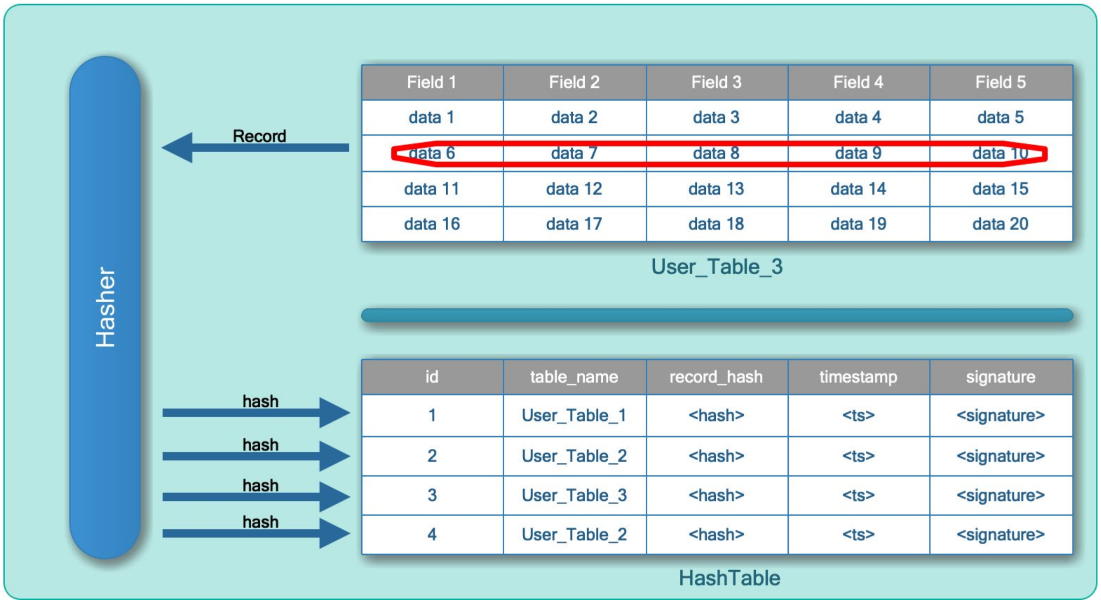
Parallelizable Hashing: By nature, the hashing process is parallelizable. This will utilize any available parallelism infrastructure present at the database server to optimize signing performance. In addition, it could be done by any external server that has access to the same database or a live replica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

4.4.2.4 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



id	table_name	record_hash	timestamp	signature
1	User_Table_1	<hash>	<ts>	<signature>
2	User_Table_2	<hash>	<ts>	<signature>
3	User_Table_3	<hash>	<ts>	<signature>
4	User_Table_2	<hash>	<ts>	<signature>

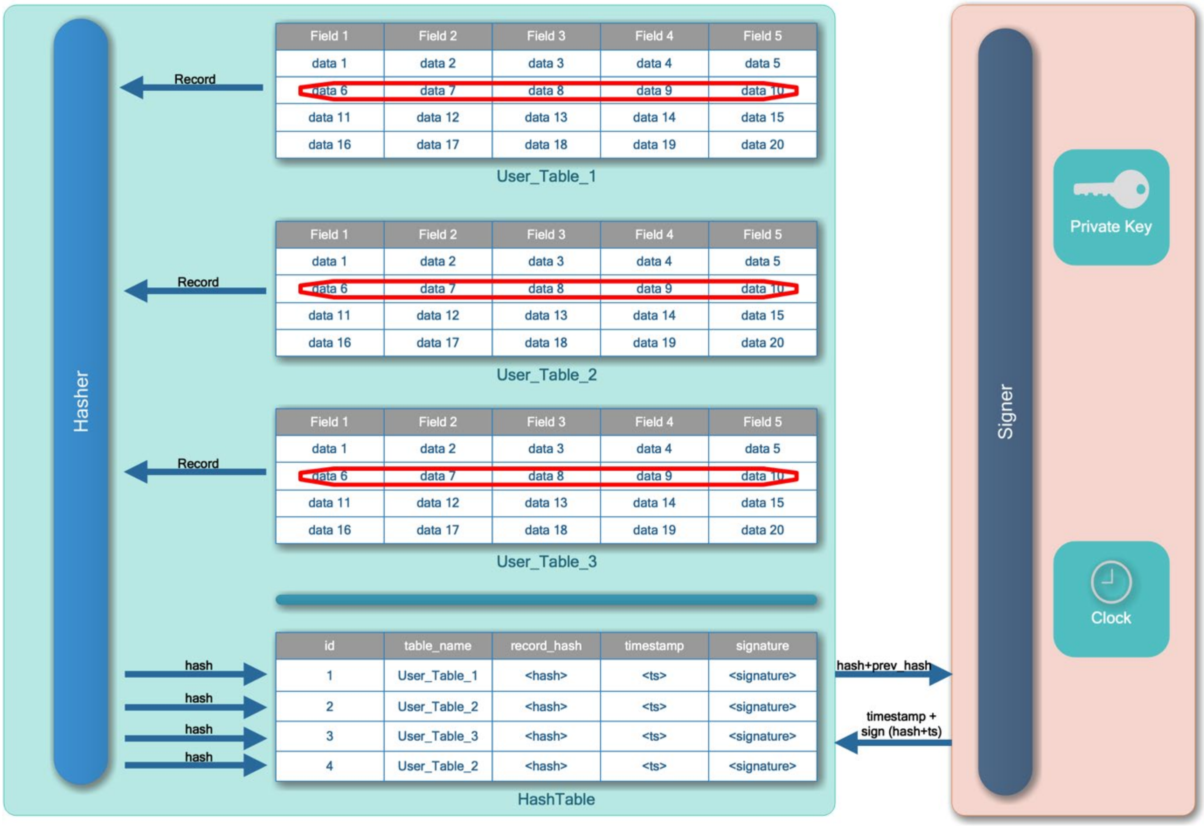
Figure 16: Chain of hashes



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

id	table_name	record_hash	timestamp	signature
1	User_Table_1	<hash>	<ts>	<signature>
2	User_Table_2	<hash>	<ts>	<signature>
3	User_Table_3	<hash>	<ts>	<signature>
4	User_Table_2	<hash>	<ts>	<signature>

Figure 17: Hasher



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

User_Table_1

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

User_Table_2

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

User_Table_3

id	table_name	record_hash	timestamp	signature
1	User_Table_1	<hash>	<ts>	<signature>
2	User_Table_2	<hash>	<ts>	<signature>
3	User_Table_3	<hash>	<ts>	<signature>
4	User_Table_2	<hash>	<ts>	<signature>

HashTable

Figure 18: Signer and time-stamper

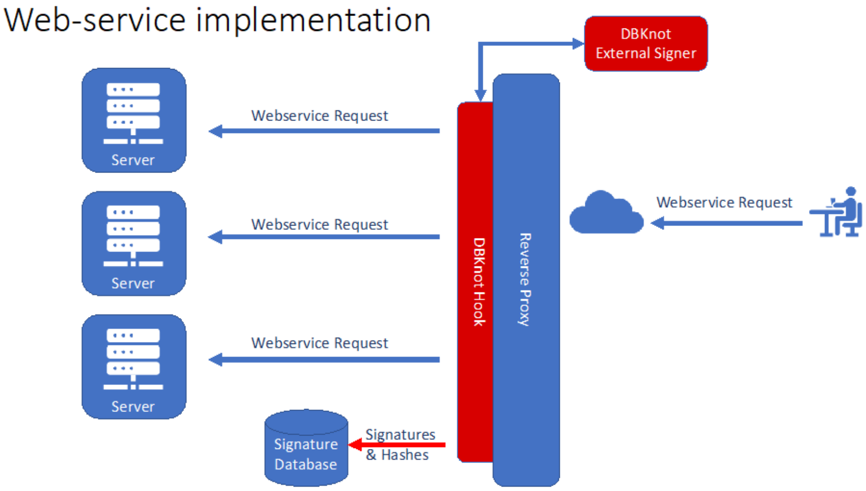


Figure 19: Web service implementation

signer. The signer will take the hash record, add to it the preceding record together with a time-stamp 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 time-stamp 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.

4.4.3 Web-Service/API Microservices Architecture. 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 while 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.

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.

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 software/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.
- **Non-relational Database:** Relying on REST web services for tracking database CRUD operations opens the way to cater to other non-relational database models directly without being limited to a particular ORM framework or a database management system.

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.

As mentioned in Section 2.2, our approach is to try and base record chaining on the semantics of using the REST API to do CRUD functionality. This is a good entry point to the implementation of this technique. The technique could be taken a step further into covering other REST semantics but will require more implementation specific configuration and will be less transparent.

The following are some REST methods that are based on standard HTTP methods: [21, 32].

As we see in Table 1, HTTP (REST) methods automatically lend themselves to data operations.

Additionally, most of the HTTP (REST) response codes match standard database operations. [32]

4.4.4 REST API Based Definition. To be able to track a microservice based request, in most cases a specific configuration is required. Fortunately, there are new industry standards [3] for performing such configurations. Examples are OpenAPI [2, 19, 33] and RAML [25].

As we can see, a number of the details of the possible web service operation is specified in YAML format.

4.5 Verification Steps

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

Table 1: HTTP methods and REST

Method	Use
GET	Retrieve a particular record of data
HEAD	Get a summary of record data
PUT	Add a record
POST	Possibly update a data record
DELETE	Delete a data record

Table 2: HTTP (and REST) return codes

HTTP Return Code	Meaning
200 OK	Operation performed correctly
201 Created	Record added correctly
400 Bad Request	There is a problem with the request
401 Unauthorized	Authentication Required
403 Forbidden	User permission problem
404 Not Found	Item being queried does not exist

This is an example service definition using OpenAPI:

```
tags:
- pet
summary: Updates a pet in the store with form data
operationId: updatePetWithForm
parameters:
- name: petId
  in: path
  description: ID of pet that needs to be updated
  required: true
  schema:
    type: string
requestBody:
  content:
    'application/x-www-form-urlencoded':
      schema:
       type: object
       properties:
          name:
            description: Updated name of the pet
            type: string
          status:
            description: Updated status of the pet
            type: string
       required:
         - status
responses:
  '200':
    description: Pet updated.
    content:
      'application/json': {}
      'application/xml': {}
  '405':
    description: Method Not Allowed

    content:

      'application/json': {}

      'application/xml': {}

security:

- petstore_auth:

  - write:pets

  - read:pets
```

1- Malicious addition of a record: results in a record that does not have a corresponding signed hash in the hash/signature table.
2- Malicious deletion of existing records: results in an existing hash/signature without a corresponding record.
3- Malicious tampering with hashes or signatures: results in a scenario that is a combination of the two tampering situations above.

Figure 21 shows an example of the inconsistencies resulting from maliciously adding a record to the database.

There are two cases when a verification is triggered. The first one is at data read or insertion time where one 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.

4.6 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 forms of parallelism into the design.

4.6.1 Signing Distribution. In this design illustrated in Figure 22: 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 **“$\boldsymbol{n}$”** shards, then each record **“$\boldsymbol{i}$”** will be chained with distributed to shard **“$\boldsymbol{s = i \,\%\, n}$”**. 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.

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, several insert statements together with their corresponding hashes could be done in parallel without having to wait for each other to finish.

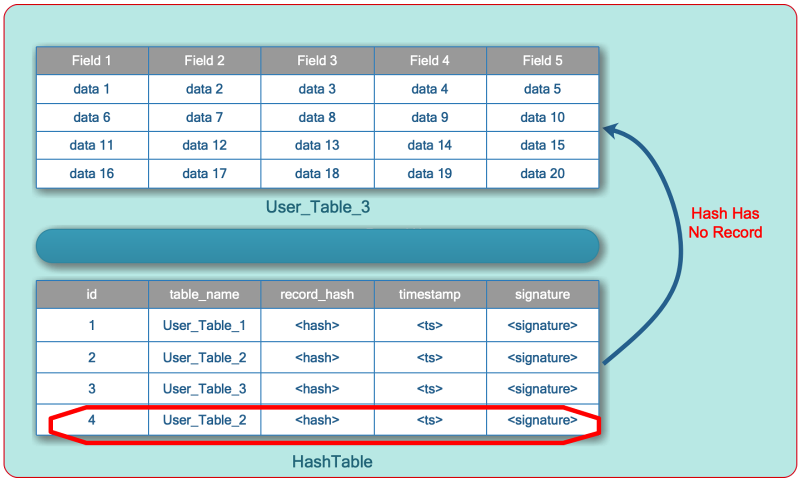
The tradeoff in this approach is that database verification is divided into “n” independent chunks which makes the chaining process less complex. One mitigation for that is to introduce occasional inter-shard linkages to tightly intertwine them together and eliminate that independence.

Figure 23 illustrates how consecutive transactions are linke, hashed, chained, and signed together and how they are split into groups.

4.6.2 Coarse Grained Block Signing. 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.

The figure below (Figure 24) shows how transaction 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 hash table with an entry for every record, a smaller hash table 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. To mitigate for this problem, a variable size block could be implemented (illustrated in Figure 24: 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 time-stamp 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



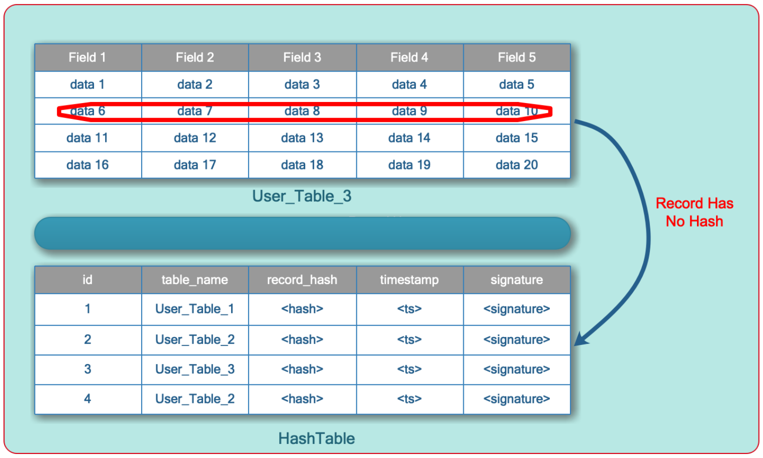
Field 1	Field 2	Field 3	Field 4	Field 5
data 1	data 2	data 3	data 4	data 5
data 6	data 7	data 8	data 9	data 10
data 11	data 12	data 13	data 14	data 15
data 16	data 17	data 18	data 19	data 20

User_Table_3

id	table_name	record_hash	timestamp	signature
1	User_Table_1	<hash>	<ts>	<signature>
2	User_Table_2	<hash>	<ts>	<signature>
3	User_Table_3	<hash>	<ts>	<signature>
4	User_Table_2	<hash>	<ts>	<signature>

HashTable

Figure 20: Detection of a maliciously deleted record



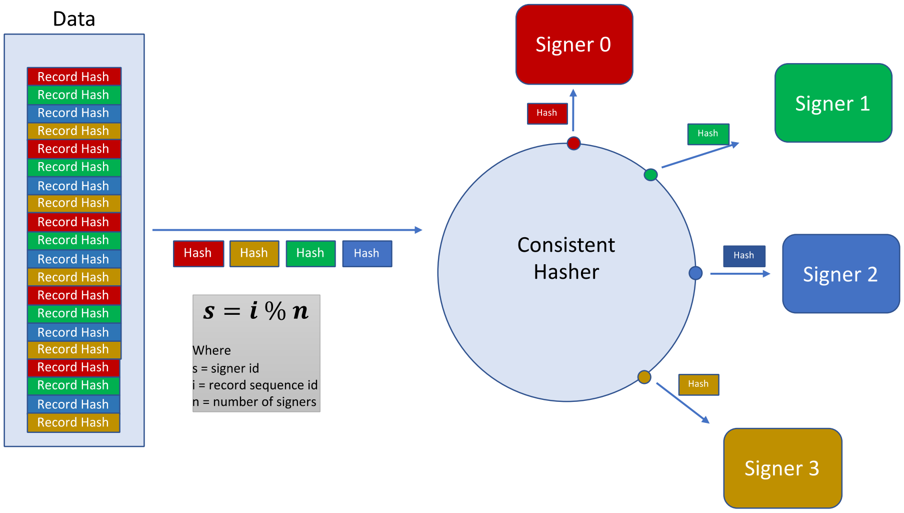
Field 1	Field 2	Field 3	Field 4	Field 5
data 1	data 2	data 3	data 4	data 5
data 6	data 7	data 8	data 9	data 10
data 11	data 12	data 13	data 14	data 15
data 16	data 17	data 18	data 19	data 20

User_Table_3

id	table_name	record_hash	timestamp	signature
1	User_Table_1	<hash>	<ts>	<signature>
2	User_Table_2	<hash>	<ts>	<signature>
3	User_Table_3	<hash>	<ts>	<signature>
4	User_Table_2	<hash>	<ts>	<signature>

HashTable

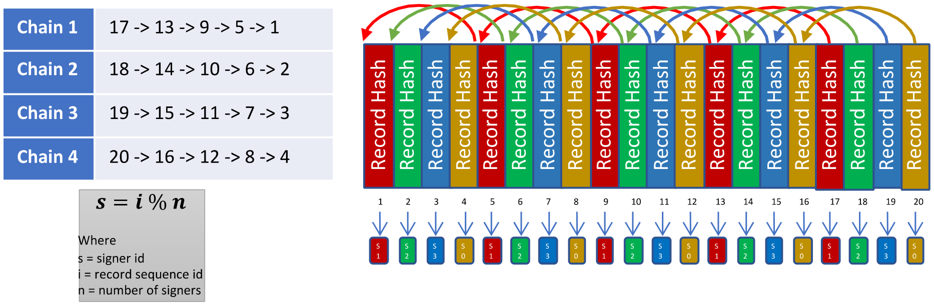
Figure 21: Detection of a maliciously added record



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

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

Figure 22: Parallel signers - consistent hashing



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 23: Parallel signers - linking of hashes

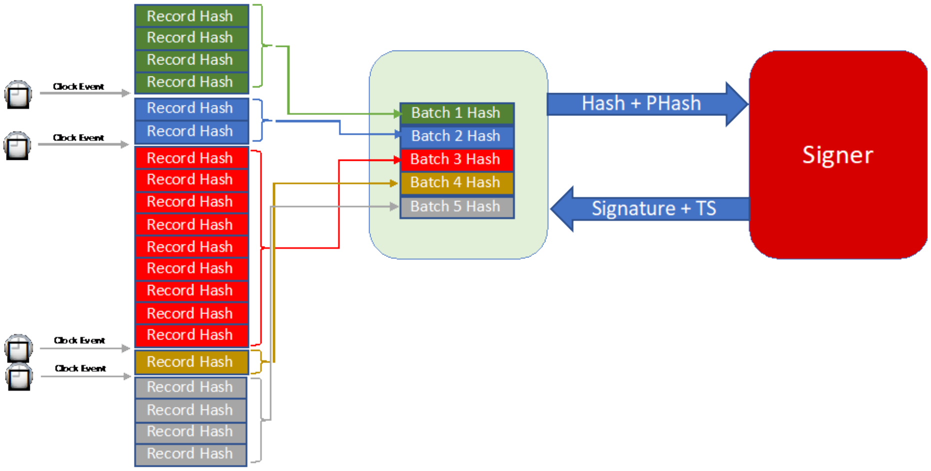


Figure 24: Coarse grained signing - variable block size

with intermittent or unreliable connectivity.

4.7 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.

4.7.1 Parameters. For each of the techniques used, there are three 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

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

Figure 25: 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 three numbers.
- **Hashing Bound:** In these scenarios, the hashing time is the longest of the three numbers.
- **Signing Bound:** In these scenarios, the signing time is the longest of the three numbers.

All tests are done on two 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.

4.7.2 Technique 1: Inline Hashing & Signing. The first technique 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 Figure 26.

There are three scenarios of implementing the “all-inline” sequential method. Such scenarios are used in comparison of different techniques under varying conditions.

The formula in Figure 27 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.

4.7.3 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. Please note that the transaction and hashing in this approach remain sequential.

4.7.4 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. 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.

4.7.5 Technique 4: Concurrency Through Pipelining All Operations. This technique is different from all the others above. In this technique we separate each of the three steps (transaction, hashing, and pipelining) into its own pipeline and let them run asynchronously while preserving sequence dependencies.

In this solution everything runs in parallel. Where a hasher is separate from a signer and separate from the main transaction thread of execution.

5 Experimentation and Results

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 displaying 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.

The two comparison sets of heatmaps below show that 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.

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 handling 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 such as accounting systems, banking systems, and system logs. A chain of records inspired

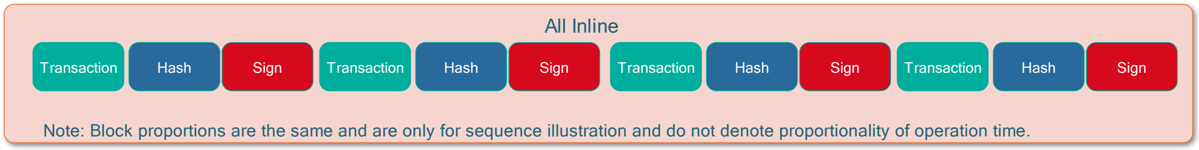

All Inline
Transaction
Hash
Sign
Transaction
Hash
Sign
Transaction
Hash
Sign
Transaction
Hash
Sign
Note: Block proportions are the same and are only for sequence illustration and do not denote proportionality of operation time.

Figure 26: Inline hashing & signing

All inline formula:

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

Figure 27: Formula for "all inline"

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 three 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.

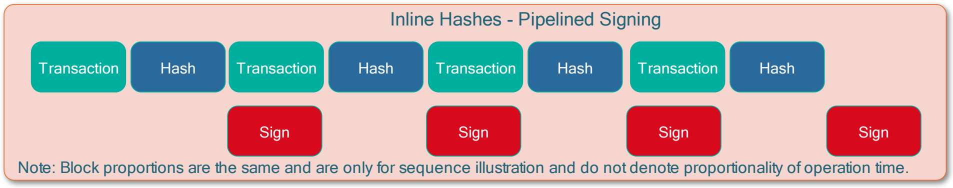
We have illustrated how DBKnot could be implemented in a web service model and how new web service definition languages can be used to facilitate the DBKnot web service configuration process for systems that adhere to the standard and properly define their services. In that case, this can be done with much less intervention from the system admin than if nothing was defined at all.

We have performed tests using generated workloads. As expected, the tests showed an increased overhead for the hashing and signing 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 the synchronicity of hashing and signing.

We have explored different parallelization by testing four 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.



Note: Block proportions are the same and are only for sequence illustration and do not denote proportionality of operation time.

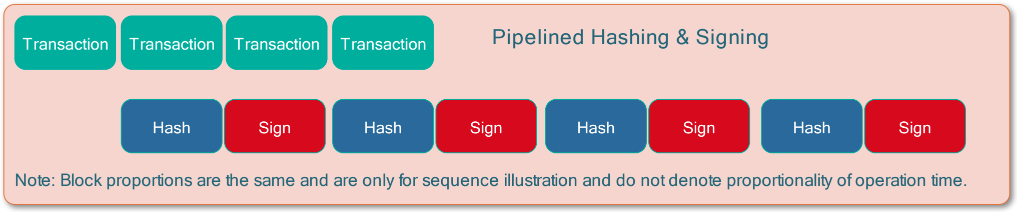
Figure 28: Partial concurrency through signature pipelining

Formula for signature pipelining:

$$v1 = s + \sum_{i=0}^{n} t + h \qquad v2 = t + \sum_{i=0}^{n} s + h$$

$$v = \max\,(v1, v2)$$

Figure 29: Formula for signature pipelining



Note: Block proportions are the same and are only for sequence illustration and do not denote proportionality of operation time.

Figure 30: Concurrency through hash and signature pipelining

Formula for hash and signature pipelining:

$$v1 = s + \sum_{i=0}^{n} t + h \qquad v2 = t + \sum_{i=0}^{n} s + h$$

$$v = \max\,(v1, v2)$$

Figure 31: Formula for signature and hash pipelining

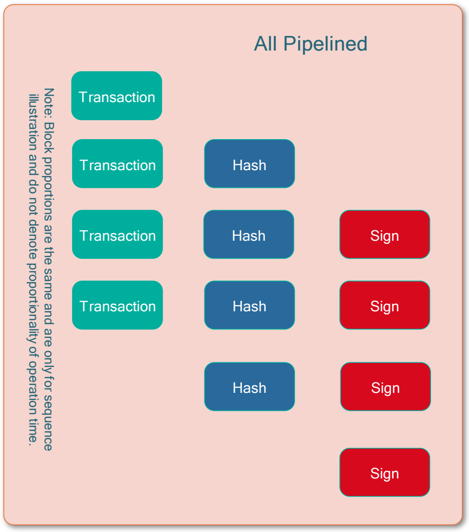

All Pipelined
Note: Block proportions are the same and are only for sequence illustration and do not denote proportionality of operation time.
Transaction
Transaction
Hash
Transaction
Hash
Sign
Transaction
Hash
Sign
Hash
Sign
Sign

Figure 32: Pipelining all operations

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.

DBKnot is designed as much as possible to 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

.

Formula all pipelining:

$$v1 = h + s + \sum_{i=0}^{n} t \qquad v2 = t + s + \sum_{i=0}^{n} h \qquad v3 = t + h + \sum_{i=0}^{n} s$$

$$v = \max\,(v1, v2, v3)$$

Figure 33: Formula for pipelining all operations

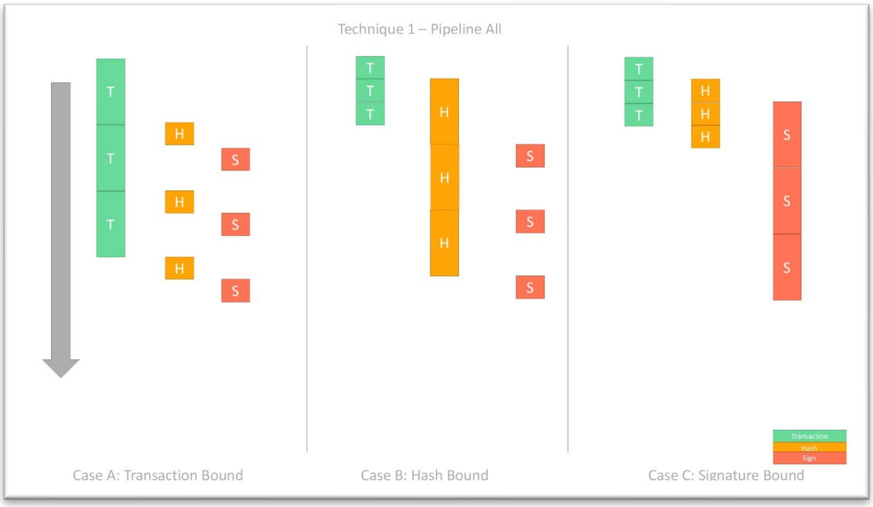


Figure 34: Pipelining all – illustration

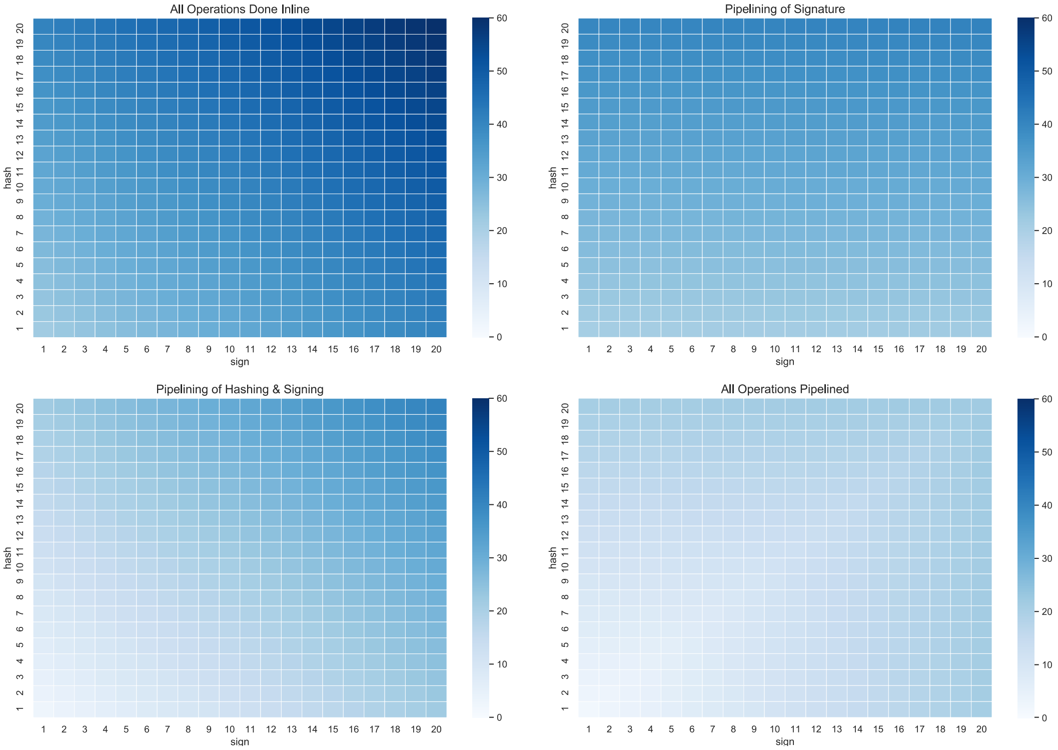


Figure 35: Transaction performance comparison heatmap

References

[1] “Amazon QLDB,” *Amazon Web Services, Inc.* https://aws.amazon.com/qldb/ (accessed May 02, 2019).

[2] Apache Foundation, “OpenAPI Specification v3.1.0 | Introduction, Definitions, & More.” https://spec.openapis.org/oas/v3.1.0 (accessed Jan. 21, 2022).

[3] “API Specifications Conference,” *Linux Foundation Events*. https://events.linuxfoundation.org/openapi-asc/ (accessed Jan. 21, 2022).

[4] “BigchainDB 2.0 Whitepaper • • BigchainDB,” *BigchainDB*. https://www.bigchaindb.com/whitepaper/ (accessed May 11, 2019).

[5] “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).

[6] “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.

[7] "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).

[8] 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, 2002.

[9] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California, Irvine, CA, 2000.

[10] A. Goel, Wu-chang Feng, D. Maier, Wu-chi Feng, and J. Walpole, "Forensix: A Robust, High-Performance Reconstruction System," *25th IEEE International Conference on Distributed Computing Systems Workshops*, Columbus, OH, USA, pp. 155-162, 2005. doi: 10.1109/ICDCSW.2005.62.

[11] "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).

[12] R. Hasan, R. Sion, and M. Winslett, "The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance," *Proccedings of the 7th Conference on File and Storage Technologies*, Berkeley, CA, USA, pp. 1-14, 2009. Accessed: Oct. 11, 2019. [Online]. Available: http://dl.acm.org/citation.cfm?id=1525908.1525909

[13] I. Khalil, S. El-Kassas, and K. Sobh, "DBKnot: A Transparent and Seamless, Pluggable, Tamper Evident Database," *EPiC Series in Computing*, 77:90-103, Oct. 2021. doi: 10.29007/7l81.

[14] 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).

[15] "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).

[16] "Object-relational Mappers (ORMs)." https://www.fullstackpython.com/object-relational-mappers-orms.html (accessed Aug. 23, 2020).

[17] "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

[18] 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).

[19] "OpenAPI Specification," *Wikipedia*. Nov. 27, 2021. Accessed: Jan. 21, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=OpenAPI_Specification&oldid=1057453112

[20] "OSTree." https://ostree.readthedocs.io/en/latest/ (accessed Aug. 12, 2020).

[21] "Overview of RESTful API Description Languages," *Wikipedia*. Jan. 17, 2022. Accessed: Jan. 21, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Overview_of_RESTful_API_Description_Languages&oldid=1066320532

[22] 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).

[23] K. E. Pavlou and R. T. Snodgrass, "Forensic Analysis of Database Tampering," *ACM Trans Database Syst*, 33(4)30:1-30:47, Dec. 2008, doi: 10.1145/1412331.1412342.

[24] 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.

[25] "RAML (Software)," *Wikipedia*. Oct. 14, 2021. Accessed: Jan. 21, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=RAML_(software)&oldid=1049813969

[26] "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/

[27] "Representational State Transfer," *Wikipedia*. Jan. 06, 2022. Accessed: Jan. 21, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=1064071285

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

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

[30] "Snapcraft - Snaps Are Universal Linux Packages," *Snapcraft*. https://snapcraft.io/ (accessed Aug. 12, 2020).

[31] D. M. Upton and S. Creese, "The Danger from Within," *Harvard Business Review*, no. September 2014, Sep. 01, 2014. Accessed: May 09, 2019. [Online]. Available: https://hbr.org/2014/09/the-danger-from-within

[32] S. Watts, "REST vs CRUD: Explaining REST & CRUD Operations," *BMC Blogs*. https://www.bmc.com/blogs/rest-vs-crud-whats-the-difference/ (accessed Jan. 21, 2022).

[33] "Welcome," *RAML*. https://raml.org/ (accessed Jan. 21, 2022).

[34] "Welcome to Flatpak's Documentation! — Flatpak Documentation." https://docs.flatpak.org/en/latest/ (accessed Aug. 12, 2020).

[35] Weltwirtschaftsforum and Zurich Insurance Group, *Global risks 2019: insight report*. 2019. Accessed: Nov. 07, 2019. [Online]. Available: http://www3.weforum.org/docs/ WEF_Global_Risks_Report_2019.pdf

[36] "What is Object/Relational Mapping? - Hibernate ORM." https://hibernate.org/orm/what-is-an-orm/ (accessed Aug.

23, 2020).

[37] "What is REST." https://restfulapi.net/ (accessed Jul. 26, 2020).

[38] C. Xia, G. Yu, and M. Tang, "Efficient Implement of ORM (Object/Relational Mapping) Use in J2EE Framework: Hibernate," pp. 1–3, Jan. 2010, doi: 10.1109/CISE.2009.5365905.

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

Islam Khalil has received his BSc and MSc degrees in computer science from The American University in Cairo. He is currently pursuing his PhD with a primary focus on database systems and security. On the professional side, Khalil is the co-founder of companies that provide data analytics, business intelligence, and cloud services for various enterprises in the areas of telecommunication, AgTech, utilities, defense, and others worldwide. Khalil's company has been recognized as one of the top 5 companies worldwide in applying artificial intelligence to the field of agriculture and one of the top 3 worldwide most influential companies in data analytics in some specific agriculture verticals. Khalil has been appointed by the minister of industry on the board of directors of various semi-governmental organizations focused on industry and export development.

Sherif El-Kassas is a Professor of Computer Science and Engineering at the American University in Cairo. El-Kassas' research interests are focused on Security Engineering, the application of formal methods in Software engineering and Computer Security, and Open Source technologies.

El-Kassas is also a consultant for various organizations; Member of the board of e-finance (leading provider of governmental and payment services), former board member of the Information Technology Industry Development Agency (ITIDA); Member of the board of trustees of the Egyptian e-signature center of excellence; Founding member of the Egyptian Open Source NGO (OpenEgpt) and Internet Masr (Egyptian Chapter of Internet Society); Founding partner, former broad member, and former CTO of SecureMisr (leading Egyptian Information Security services providers, recently acquired Cysiv a trend micro company); Founder of new startup, QuiverLabs, focusing on innovative threat modeling and incident response technologies; and Member of various professional computing societies.

El-Kassas received his Ph.D. from the Eindhoven University of Technology in the Netherlands.

Karim Sobh received the B.Sc., M.Sc., and Ph.D. degrees in computer science from The American University in Cairo. He worked at the American University in Cairo (AUC) as a Full time Assistant Professor for three academic years in the Department of Computer Science and Engineering. Prior to that he worked at Nile University (NU), Cairo and as an Assistant Professor and the University of California at Santa Cruz (UCSC) as a Visiting Lecturer. He is currently the Chief Technology Officer (CTO) of Blnk Consumer Finance, an emerging FinTech startup in Egypt. He also founded Code-Corner, a software development firm providing software development, subcontracted services, cloud deployment services, consultation services, and turn-key solutions using open source technologies. He also worked as a Systems Architecture Consultant at IBM Egypt. His role included and was not limited to providing system architecture consultations and implementation services for large projects. His specialization is in operating systems, networks, distributed systems, and cloud computing, and his Ph.D. topic is cloud environments metering.