EPiC Series in Computing

Volume 77, 2021, Pages 90–103

Proceedings of ISCA 30th International Conference on Software Engineering and Data Engineering

DBKnot
A Transparent and Seamless, Pluggable, Tamper Evident Database

Islam Khalil, Sherif El-Kassas, and Karim Sobh

The American University in Cairo, Cairo, Egypt
{ikhalil,sherif,kmsobh}@aucegypt.edu

Abstract

Database tampering is a key security threat that impacts the integrity of sensitive information of crucial businesses. The evolving risks of security threats as well as regulatory compliance are important driving forces for achieving better integrity and detecting possible data tampering by either internal or external malicious perpetrators. We present DBKnot, an architecture for a tamper detection solution that caters to such problem while maintaining seamlessness and ease of retrofitting into existing append-only database applications with near-zero modifications. 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. A set of preliminary experiments have been conducted that resulted in DBKnot adding an overhead equal to the original transaction time. We have run the same experimemts experiments with different parallelization and pipelining versions of DBKnot which resulted in cutting approximately 66% of the added overhead.

1 Introduction

With the growing need for digitalization by various industries and the pervasiveness of sophisticated software systems, there is a matching growing need for securing such systems. Systems mange 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 external attackers colluding with insiders to commit their fraud crimes using their legitimately assigned access rights. According to the Association of Certified Fraud Examiners (ACFE) 2018 report[14], $7 Billion of losses were incurred due to internal fraud alone with an average fraud scheme going for 16 months unnoticed.

According to Harvard Business Review[23], more than 80 million insider security breaches occur every year costing tens of billions of dollars every year in the US alone. For example, an incident of $350,000 that were stolen from 4 Citibank customers by employees of a software and service company that Citibank had contracted. According to Accenture[23] and The World Economic Forum (WEF)[24], the cost of insider malicious activity constitutes 15% of all cybercrime. The IETF's RFC 4810[8] 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.

Various governments have put into place different regulations to reduce such risks by governing the collection, retention, and disclosure of data. Among such regulations are the Gramm-Leach-Bliley Act[5] for financial institutes, Sarbanes-Oxley Act[17] (SOX), and Health Insurance Portability and Accountability Act[22] (HIPAA) for medical records.

The goal of our work is to design an architecture that enables traditional database systems to be tamper-evident. Such architecture can be implemented at different levels; namely the ORM level, database level, or web-service level. The primary goal is to eliminate the need for relying completely on trust inside the organization while minimizing the overhead added by the solution. Ease of integration is a key feature while requiring near-zero changes to existing systems. Our proposed architecture targets primarily append-only systems like 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.

The rest of the paper is organized as follows: we present the needed background in section 2 followed by a selection of the related work in section 3. We formulate our problem statement in section 4 coupled with our motivation behind the proposed tamper detection architecture. In Section 5 we present our proposed solution followed by experiments and results in section 6. Finally we conclude and present possible future work venues in section 7.

2 Background

2.1 Object Relational Mapping

Object Relational Mapping (ORM) frameworks[6][11] 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, definition, transactions, as well as querying. Figure 2 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.

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[12]. Figure 1 is an example of how web-services work.

2.3 Transaction Chaining

The idea behind blockchain is that 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 3. 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 and the immutability of such transactions because a modification of any of the transactions will invalidate the rest of them

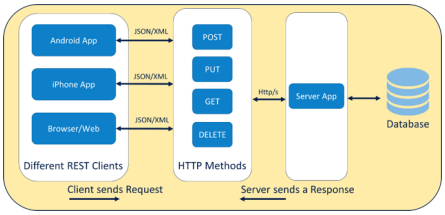


Figure 1: REST API Request and Response

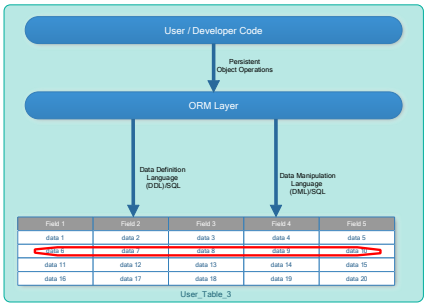


Figure 2: Standard ORM Operation

and thus, "break the chain". The first block in a blockchain is called the "Genesis block". Each blockchain has its own genesis block.

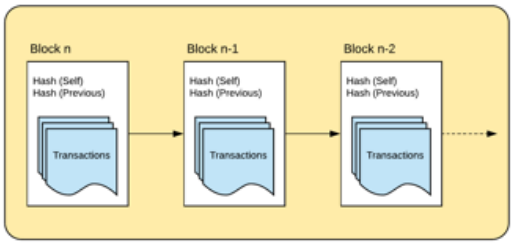


Figure 3: Chaining of Hashes into Blocks

3 Related Work

A number of different solutions have been proposed to target the problem we are addressing. Solutions vary in the way they tackle the problem. Some of them use a similar technique of chained hashes. All of 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[18][20][19][21] 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)[1][2] – a blockchain based database - solves part of the problem addressed in our work. QLDB provides the clock-chaining functionality in a proprietary database that is hosted on Amazon cloud and provided a Software-As-A-Service.

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

There are other research work that focus on documents rather than data. Some of which are designed to track documents provenance throughout their lifecycle. A number of other research work has catered to a similar problem in the domains of operating systems and filesystems. Examples are [4][16][10][9][13][7][15]. 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.

4 Problem Statement and Motivation

Database environments are sometimes vulnerable to illegitimate tampering by internal administrators who have full access to the data by virtue of their administration roles. The need for a tamper detection mechanism is important as an initial counter measure against the disastrous effects of such actions.

By looking at the related work, we highlight the primary gaps in existing solutions. Some solutions address one or more of the gaps but most of them fail to address all the critical ones simultaneously. For example: eliminating the need to base all security on trusting an insider administrator, ensuring that all organization's data remains inside, and ease of retrofitting into existing applications with near-zero intervention. This has motivated us to work on a solution that tries to address a number of the important goals simultaneously with a primary focus on applicability to retro-fitting to existing systems.

5 Proposed Solution

5.1 Solution Overview

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 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. 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 microservice solutions by being implemented as a totally transparent reverse proxy.

5.2 The Hasher and The External Timestamping Signer

The direction adopted is to introduce a third-party externalized time-stamper/signer that is outside the boundaries of the organization as shown in Figure 4 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 4.

Being external, the signer is outside the reach of organization insiders which reduces and ideally eliminates the possibility of collusion among internal and external stakeholders. We introduced in Figure 4, an independent signer + time-stamper service (in red). The signer service could cater to different organizations as illustrated in the diagram.

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: Such statelessness reduces possible attack vectors, ensures confidentiality of the tracked data since it is not stored on the signer server, saves bandwidth and storage that would have been required in cases where a data replica is stored, and makes it simpler to implement load-balancing, redundant signing servers for CDN-like servers.

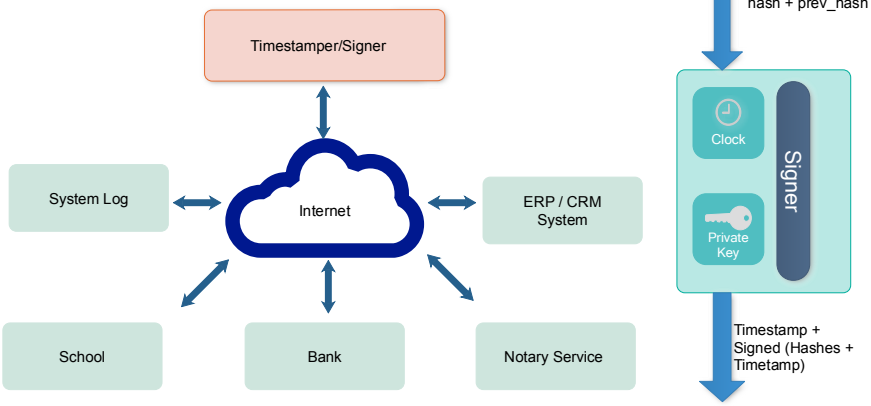


Figure 4: Introduction of Third Party Signing Service

Figure 5: Signer Service

As explained before in section 2.3, Figure 6 shows how the whole architecture fits together to form a single immutable chain of transactions. This is implemented by using an external signer. Transactions are chained at record insert time. The private key of the external signer is used to sign transactions and the public key is used for verification.

5.3 Integration Models

5.3.1 ORM Level Integration

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

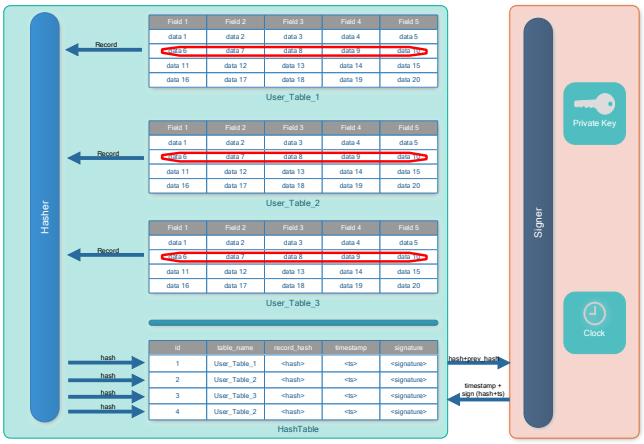


Figure 6: Signer and Timestamper

the same implementation will work on any database as long as it is supported by the used ORM layer without any change at all.

As the user code initiates any persistent database 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 work.

Figure 7 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

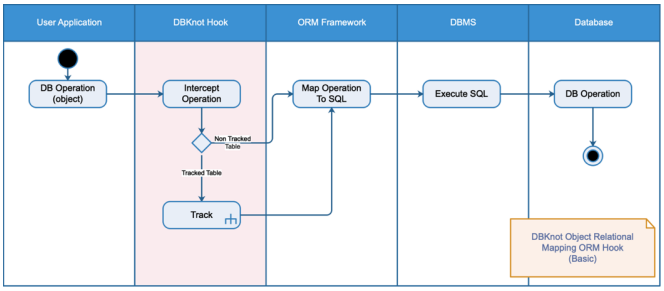


Figure 7: Adding DBKnot ORM Hook - Activity Diagram

completely seamless user experience to developers. In the current implementation, as illustrated in the python/django code below, all a user (developer) needs to do is to have the model classes extend a class (a mixin) that provides all needed functionality. This is an example of the DBKnot mixin usage written in python.

```
class Test(DBKnotMixin):    # Embedded in the form of a simple mixin
    name=models.CharField("Name", max_length=50)
    def __str__(self)
        return self.name
```

5.3.2 Database Level Integration

Database-level integration is done using the exact same methodology as the ORM but on a database-level. 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 than through the ORM.

5.3.3 Web Service API Level Integration

DBKnot functionality could be implemented inside a reverse-proxy 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 and microservices with doing zero changes to the existing application. This way existing applications can benefit from DBKnot and secure their data seamlessly. The primary challenge is that it will require an easy-to-use mini language/syntax for application developers to define their application web-service's semantics. The primary advantage however is that it is totally non-invasive and could be totally external to server inside a reverse proxy. Advantages if this approach include being agnostic with regards to the technology in use, supporting any mix of hybrid microservices as well as supporting load-balancing multi-server architectures.

5.4 Verification Steps

Verification of records and thus, the detection of possible tampering is done by checking for the validity of signatures as well as the consistency of signatures and hashes with the actual data. The validation algorithm will look for dangling hashes, missing hashes, or inconsistent record-hash pairs.

Figure 8 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

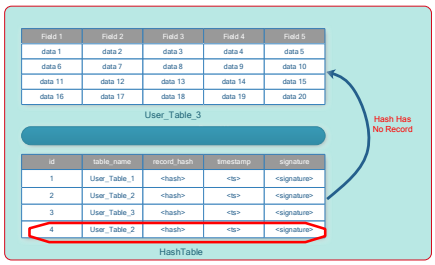


Figure 8: Detection of Maliciously Added Record

data-read or insertion time where a 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. Patrolling tasks would adaptively work when resource utilization is low so they do not impact the performance of the day-to-day system transactions.

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.

5.5 Performance Optimization

To minimize the performance impact of the tracking, signing, and hashing layer, 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 introduce different forms of parallelism into the design.

5.5.1 Signing Distribution

In this design illustrated in Figure 9, 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 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. The advantage of this technique

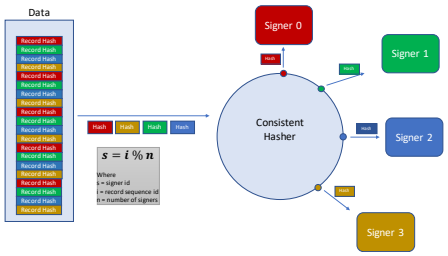


Figure 9: Why one should use EasyChair

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.

Figure 10 illustrates how consecutive transactions are linked, hashed, chained, and signed together and how they are split into groups.

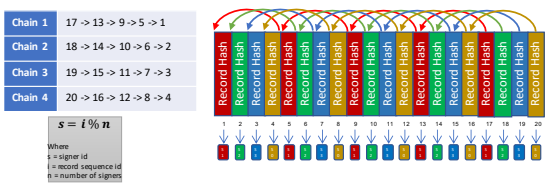


Figure 10: Why one should use EasyChair

5.5.2 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.

To optimize the signing process, 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. To mitigate for this problem, a variable-size block could be implemented as illustrated in Figure 11. 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.

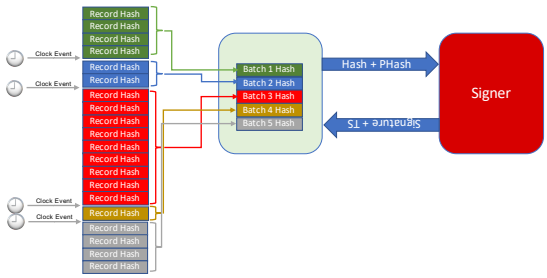


Figure 11: Coarse Grined Signing - Variable Block Size

5.6 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. maliciously-added-record

5.6.1 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, Hashing Time, and Signature Time.

All Variables:

- n=number of transactions
- t=transaction time (t1→short transaction,t2→long [4X] transaction)
- h=hashing time
- s=signing time

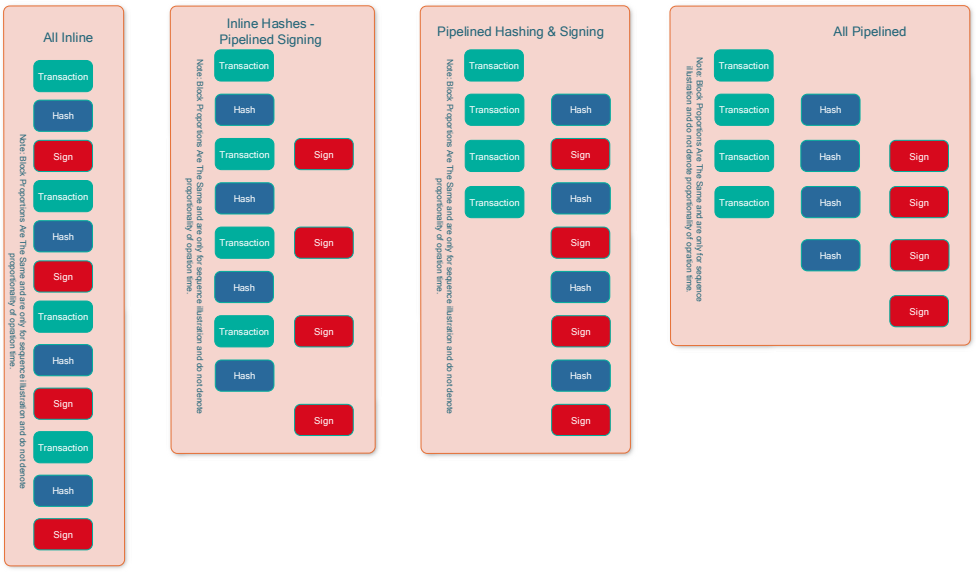


Figure 12: Comparison of the 5 Pipelining Techniques

- v=total batch duration

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×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.

5.6.2 Technique 1: Inline Hashing and 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 Figure 12.

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

5.6.3 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 to gain some performance. Please note that the transaction and hashing in this approach remain sequential.

5.6.4 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. This time taken for a transaction is summarized as follows: $v = max(s + \sum_{i=0}^{n} t + h, t + \sum_{i=0}^{n} s + h)$

5.6.5 Technique 4: Concurrency Through Pipelining All Operations

This technique is different from all the others above. In this technique we separate each of the 3 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. The time taken for a transaction is equivalent to: $v = max(h + s + \sum_{i=0}^{n} t, t + s + \sum_{i=0}^{n} h, t + h + \sum_{i=0}^{n} s)$

6 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 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.

Figure 13 shows the classification of test data based on a comparison of time taken by the actual transaction, the hashing process, and the signing process. As illustrated, the generated test data ensures that all different combination scenarios are taken into consideration. This includes transaction-bound operations, hashing-bound operations, as well as signing-bound operations.

Figure 13 also shows a visual comparison of how the 4 different techniques of parallelism visually compare in the 3 different scenarios of workloads.

The comparison sets of heatmaps below shows 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 and 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

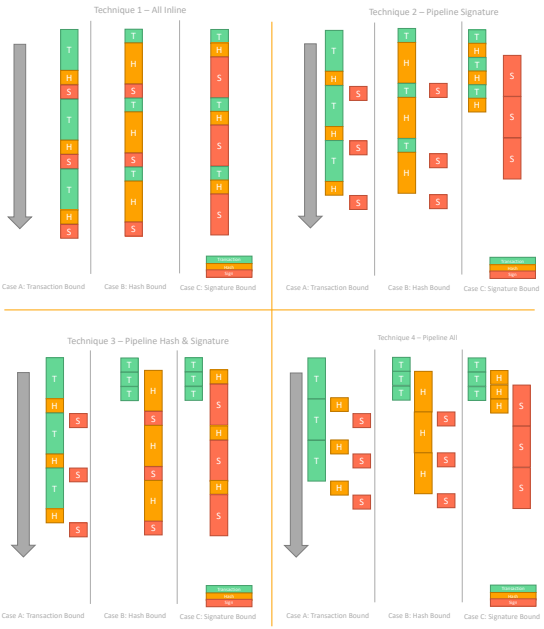
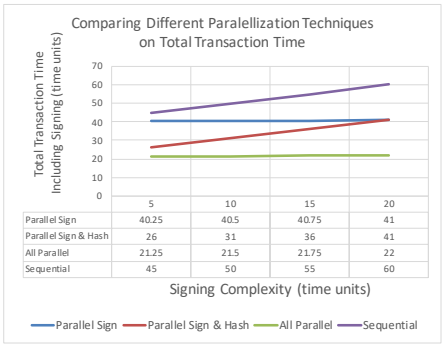


Figure 13: Classification of Generated Test Data

More details are outlined in Figure 14 and Figure 15 with a numerical comparative indication of relative performance speedup. We have also assessed the impact of the proposed hashing,



	5	10	15	20
Parallel Sign	40.25	40.5	40.75	41
Parallel Sign & Hash	26	31	36	41
All Parallel	21.25	21.5	21.75	22
Sequential	45	50	55	60

Figure 14: Comparison of Pipelining Techniques Results

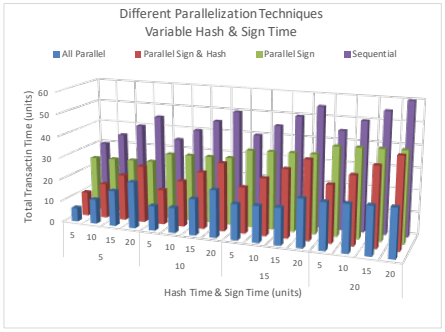


Figure 15: Comparison of Pipelining Techniques Results

signing, and chaining architecture on the performance of the transactions in comparison with the plain transactions without any additional tracking. Our solutions did add an overhead that is slightly higher than transaction time that is slightly more than the double of the transaction-only time. As illustrated however by Figure 17, the overhead is reduced with time. The overhead time however was cut down to approximately 33% using the pipelining techniques we implemented.

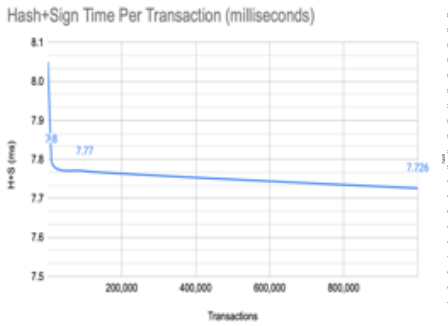


Figure 16: Diminishing Performance Overhead

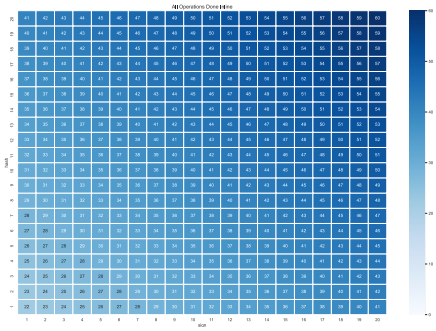


Figure 17: Total Time Taken Using Sequential Hashing and Signing

7 Conclusions and Future Work

We designed a tamper-evident architecture called DBKnot for detecting database tampering. An external signer is being used to provide stronger detection of database tampering even by an insider who has full authority and access rights over the whole system including operating systems, databases, networks, and firewalls. DBKnot enables tracking of individual immutable tables by chaining transaction hashes and signing them by an external signer or a hardware-security-module. Based on our experiments, performance overhead was significantly reduced by using different parallelization and pipelining techniques that reduced the synchronicity of hashing and signing.

Based on the related work presented above, we believe that our proposed architecture achieves tamper detection for a certain class of database applications (i.e.: Append only use-cases). The architecture is designed to be lightweight, easy to retrofit into existing systems, as well as being easy to integrate with near-zero modifications.

DBKnot was able to achieve initially the target functionality with an extra overhead that is approximately equal to the plain transaction time without tracking. The overhead varies according to different workload scenarios outlined in the experiment results chapter. At the second round of repeated experiments and as per the normalized numbers, Figure 14 and Figure 15 show that the fully parallel approach to our hashing and signing design leads to 66% performance speedup than the non-parallel approach.

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

- Expand beyond immutable transactions to support update and delete
- More advanced verification algorithms
- Work on supporting web-services and IoT
- Cater to database structural changes

References

[1] Amazon gets into the blockchain with Quantum Ledger Database & Managed Blockchain. URL: http://social.techcrunch.com/2018/11/28/amazon-gets-into-the-blockchain-with-quantum-ledger-database-managed-blockchain/.

[2] Amazon QLDB. URL: https://aws.amazon.com/qldb/.

[3] BigchainDB 2.0 Whitepaper • • BigchainDB. URL: https://www.bigchaindb.com/whitepaper/.

[4] Designing better file organization around tags, not hierarchies. URL: https://www.nayuki.io/page/designing-better-file-organization-around-tags-not-hierarchies#git-version-control.

[5] Gramm-Leach-Bliley Act. URL: https://www.ftc.gov/tips-advice/business-center/privacy-and-security/gramm-leach-bliley-act.

[6] Object-relational Mappers (ORMs). URL: https://www.fullstackpython.com/object-relational-mappers-orms.html.

[7] OSTree. URL: https://ostree.readthedocs.io/en/latest/.

[8] RFC 4810 - Long-Term Archive Service Requirements. URL: https://datatracker.ietf.org/doc/rfc4810/.

[9] Snapcraft - Snaps are universal Linux packages. URL: https://snapcraft.io/.

[10] Welcome to Flatpak's documentation! — Flatpak documentation. URL: https://docs.flatpak.org/en/latest/.

[11] What is Object/Relational Mapping? - Hibernate ORM. URL: https://hibernate.org/orm/what-is-an-orm/.

[12] What is REST. Library Catalog: restfulapi.net. URL: https://restfulapi.net/.

[13] Canonical's Snap: The Good, the Bad and the Ugly, July 2016. Section: Development. URL: https://thenewstack.io/canonicals-snap-great-good-bad-ugly/.

[14] Report to the Nations - 2018 Global Study on Occupational Fraud and Abuse. Technical report, Association of Certified Fraud Examiners, 2019. URL: https://www.acfe.com/report-to-the-nations/behind-the-numbers/.

[15] George W Dunlap, Samuel T King, Sukru Cinar, Murtaza A Basrai, and Peter M Chen. ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay. page 14.

[16] Luis Lavaire. Immutable systems: how they work and why should we care., July 2019. URL: https://medium.com/nitrux/immutable-systems-how-they-work-and-why-should-we-care-39e567a59f28.

[17] Michael G. Oxley. H.R.3763 - 107th Congress (2001-2002): Sarbanes-Oxley Act of 2002, July 2002. URL: https://www.congress.gov/bill/107th-congress/house-bill/3763.

[18] Kyriacos Pavlou and Richard Snodgrass. DRAGOON: An Information Accountability System for High-Performance Databases. *Proceedings - International Conference on Data Engineering*, pages 1329–1332, April 2012. doi:10.1109/ICDE.2012.139.

[19] Kyriacos Pavlou and Richard T. Snodgrass. Forensic Analysis of Database Tampering. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 109–120, New York, NY, USA, 2006. ACM. URL: http://doi.acm.org/10.1145/1142473.1142487, doi:10.1145/1142473.1142487.

[20] Kyriacos E. Pavlou and Richard T. Snodgrass. Generalizing Database Forensics. *ACM Trans. Database Syst.*, 38(2):12:1–12:43, July 2013. URL: http://doi.acm.org/10.1145/2487259.2487264, doi:10.1145/2487259.2487264.

[21] Kyriacos E. Pavlou and Richard T. Snodgrass. Generalizing Database Forensics. *ACM Trans. Database Syst.*, 38(2):12:1–12:43, July 2013. URL: http://doi.acm.org/10.1145/2487259.2487264, doi:10.1145/2487259.2487264.

[22] Office for Civil Rights (OCR). Summary of the HIPAA Security Rule, November 2009. URL: https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html.

[23] David M. Upton and Sadie Creese. The Danger from Within. *Harvard Business Review*, (September 2014), September 2014. URL: https://hbr.org/2014/09/the-danger-from-within.

[24] Weltwirtschaftsforum and Zurich Insurance Group. *Global risks 2019: insight report*. 2019. OCLC: 1099890423. URL: http://www3.weforum.org/docs/WEF_Global_Risks_Report_2019.pdf.