Thesis no: MECS-2015-15

Evaluation of Data Integrity Methods in Storage: Oracle Database

Ognjen Tomanović
Oliver Posse

Dept. Computer Science & Engineering
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

This thesis is submitted to the Department of Computer Science & Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the Degree of Master of Science in Engineering: Computer Security. The thesis is equivalent to 20 weeks of full-time studies.

Ericsson AB was a collaborative partner during this theses. The company is a leading global company offering solutions in the area of telecommunication and multimedia. Such solutions include mobile banking, multimedia solutions and network solutions. The company is ISO 9001:2000 certified. The market in which the company operates can be characterized as highly dynamic with high innovation in products and solutions. The development model is market-driven, meaning that the requirements are collected from a large base of potential end-customers without knowing exactly who the customer will be. Furthermore, the market demands high integrity, specifically due to the sensitive nature of the stored data.

Contact Information:
Authors:
Ognjen Tomanović
E-mail: ogto09@student.bth.se

Oliver Posse
E-mail: olpo10@student.bth.se

External advisor:
Ericsson AB

University advisor:
Prof. Stefan Axelsson
Dept. Computer Science & Engineering

Dept. Computer Science & Engineering
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Context. It is very common today that e-commerce systems store sensitive client information. The database administrators of these types of systems have access to this sensitive client information and are able to manipulate it. Therefore, data integrity is of core importance in these systems and methods to detect fraudulent behavior need to be implemented.
Objectives. The objective of this thesis is to implement and evaluate the features and performance impact of different methods for achieving data integrity in a database, Oracle to be more exact.
Methods. Five methods for achieving data integrity were tested. The methods were tested in a controlled environment. Three of them was tested and performance evaluated by a tool emulating a real life e-commerce scenario. The focus of this thesis is to evaluate the performance impact and the fraud detection ability of the implemented methods.
Results. This paper evaluates traditional Digital signature, Linked timestamping applied to a Merkle hash tree and Auditing performance impact and feature impact wise. Two more methods were implemented and tested in a controlled environment, Merkle hash tree and Digital watermarking. We showed results from the empirical analysis, data verification and transaction performance. In our evaluation we proved our hypothesis that traditional Digital signature is faster than Linked timestamping.
Conclusions. In this thesis we conclude that when choosing a data integrity method to implement it is of great importance to know which type of operation is more frequently used. Our experiments show that the Digital signature method performed better than Linked timestamping and Auditing. Our experiments did also conclude that application of Digital signature, Linked timestamping and Auditing decreased the performance by 4%, 12% and 27% respectively, which is a relatively small price to pay for data integrity.

Keywords: Data Integrity, Database, Performance.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

Databases are organized collections of data that are used for storing information. There are many different database management systems and the most popular is the relationship database Oracle [1]. An important requirement of a database is to be able to guarantee the integrity of the stored data, that no tampering is done on the data. The purpose of this paper is to implement and measure the performance of different methods that can be used to detect tampering of the data.
This paper will be organized as follows.
In the next section of this chapter we will explain the fundamental knowledge needed for understanding the paper. In chapter 2 we will discuss related work. The methods that we will evaluate will be presented in chapter 3. The scientific method, approach and lab environment will be presented in chapter 4. The test results are presented in chapter 5 and analyzed in chapter 6. Finally, chapter 7 concludes our paper with a conclusion and provides possible future work.

Scenario

For the purpose of this paper we are assuming a scenario where a electronic commerce (e-commerce) product is used by several thousand users daily to make and receive payments. All data is stored in a relational database and the administrator of this e-commerce system has access to all sensitive user information. This allows the administrator to freely manipulate the data and possibly make unauthorized transaction of funds. These kinds of actions are also known as internal fraud.

Scope and Research questions

The threat model for this e-commerce system we described above is limited to the unauthorized action of an administrator where he manipulates data to his, or the gain of others. Given that the data has been compromised as described in Scenario, this thesis tries to solve the data integrity problem by applying different solutions and answering the research questions below.

- What is the feature and performance impact between the tested methods?
- What is the performance difference between not having and having data integrity implemented?

The scenario we are describing does not allow the restriction of rights the administrator has, therefore we are looking into a solution that will not prohibit malicious behavior but instead detect it.

Contribution

To be able to answer the research questions presented we chose to test five different methods for achieving data integrity: *Digital signature, Auditing, Merkle hash tree, Linked timestamping, and Digital watermarking.* The implemented methods were then tested, both off-line by us and by a performance evaluation tool provided by our external adviser. The performance evaluation tool emulated a real life e-commerce system (see section 1.1.1) and provided detailed results of how the different methods behaved in different types of transactions. In this thesis we provide theoretical overviews in addition to implementational changes and performance evaluations for each of the above mentioned methods.

1.1 Background

Cryptography is an important tool to manage integrity even though most people associate it with secrecy. Data integrity and authentication are only some of the examples where cryptography is not used to achieve secrecy, but for data fingerprinting, indexing in hash tables, in combination with a secret cryptographic key to verify data integrity and message authentication, etc [2]. With the technology development and large availability and accessibility of data, cryptography has become crucial to its security.

In this chapter we present what we believe is information of core importance for understanding this thesis.

1.1.1 Electronic commerce system

As previously mentioned the scope of this thesis is fraud detection and prevention in a e-commerce system.

Electronic commerce also known as e-commerce, is commonly described as doing business using computer networks, on the internet. It involves selling and buying of products and services and financial transactions [3].

In a e-commerce system there are several different participants and they are hierarchically divided. The participants in a e-commerce system are system dependent and can vary from system to system. E.g. a transaction system may

consist of *Issuers, Agents and Users.* A Issuer is holding the capital, like a bank. They are responsible for transferring the money to Agents. An Agent can simulate an ATM machine. They receive/transfer their capital from/to Users. And lastly the Users are the lowest entity of the hierarchy and they have the possibility to interact with other Users and Agents.

E.g. in a transaction a issuer has an physical bank account that is shared by all the agents. The capital in that account is of injective type, which means that all the funds are mapped one-to-one and one agent can not send more money than it has been allocated for him on that account. An agent can be a grocery store and a user can enter that store and ask for a cash withdrawal. The cashier in the store sends an invoice to the users mobile phone and when the user accepts that invoice the cashier can then physically hand him the money. In this example the store cashier simulates an ATM machine and this type of transaction is called *cash out.* This type of transaction can be reversed. The user hands cash to the cashier and the cashier transfers the money digitally to to the users account. This transaction is called *Cash in.* If the user only wants to pay for his groceries he will receive an invoice on his phone and upon acceptation funds will be transferred from user to agent account. The user will then receive a receipt that the transaction is complete. This transaction is called *Payment.* If a user wants to send funds to another user he can do so by SMS or via an application on his phone. By doing so he is performing a *P2P transfer*[1]. After the P2P transfer, or at any other time, the users can check their account balance or transaction history by performing the *Get balance* and *Get transaction history* respectively.

All this transaction and user information needs to be stored somewhere. Data as the one previously described is usually stored in a database of some sort. These databases need to hold a lot of sensitive information like names, funds and other transaction data and it could lead to catastrophic consequences if data was corrupted from a database crash, malfunction, data loss or even tampered with. For those reasons security measures should be taken to protect the integrity of data.

1.1.2 Databases and data integrity

Databases are used by companies and individuals to store organized collections of data. If the information stored is of the sensitive kind it would not be desirable if unauthorized manipulation of that data was performed, that is the reason why data integrity is of great importance.

"The quality of correctness, completeness, wholeness, soundness and compliance with the intention of the creators of the data. It is achieved by preventing accidental or deliberate but unauthorized insertion, modification or destruction of data in a database. Data integrity is one of the six fundamental components

[1] A peer to peer transfer.

of information security" [4].

Mark Rhodes-Ousley [5] divides integrity risks into four categories:

Malfunctions - There may be a failure in either the computer or the storage that corrupts the data and leads to an integrity loss.

Data Deletion and Data loss - Data may be destroyed either accidentally or intentionally.

Data corruption and Data tampering - Data may be changed either by a malfunction in the computer or storage system, or by a malicious attacker.

Accidental Modification - One of the most common cases for loss of data integrity is accidental modifications. This could be that the wrong data is inserted or that the data might be inserted at the wrong place.

Chapter 2

Related Work

In this chapter we are going to present a paper we have found on performance evaluation of data integrity methods. We are looking into a paper we have found that performs evaluation of data integrity methods and what the differences and similarities are from the work we have done. However, the paper we present does not contain any required information for understanding this thesis.

Everything discussed in this chapter is for the purpose of applying it to our, previously mentioned, scenario, everything else falls outside of the scope of this thesis.

2.1 Performance evaluation of data integrity methods

The only paper we were able to find that does something very similar to what we did is Gunupudis paper [6]. He is trying to solve the same problem we are but with different data integrity methods. The methods he presents all use different algorithms to compute unique representation of the initial data. None of the methods presented in his paper are very efficient at detecting which entry has been manipulated.

The performance evaluation performed in his paper is different from ours because of the evaluated data integrity methods. He tested *Partial Result Authentication Codes, Hash Chaining, Set Authentication Code* and *Modified Set Authentication Code*.

The authentication code methods presented by Gunapundi use Message Authentication Code (MAC) for generating the message code. MAC is similar to one of the methods we evaluated, Digital signature. The similarity lies in the usage for achieving data integrity and not in the input requirements for creating hashes. Their results show that methods performed differently in different types of transactions but they also state "It should be noted, however, that the cost of the data integrity mechanisms is a noticeable overhead, when compared with the baseline, and differences between the methods are minor in comparison. Thus, if data integrity is required, using the Modified Set Authentication Code method

which gives the widest flexibility and strong security guarantees is a good generic solution" [6].

Based on the fact that the used methods, the testing approach, the measurements and the performance evaluation tool is different our test results are not comparable.

Chapter 3

Data integrity methods to be evaluated

Hash functions have been used for decades to ensure that data stored in untrusted data structures has not been manipulated. By calculating an entry's digest upon insertion into the data structure and storing that hash in a secure location you make it infeasible for someone to manipulate the data and produce the same hash.

The methods presented in this chapter can all use different types of storage for storing and generating integrity keys. We will discuss our implementation and other possibilities in a later chapter.

3.1 Prerequisites

In this section we present information about methods and techniques that are used by the methods we later evaluate.

3.1.1 Asymmetric cryptography

The data integrity methods we have chosen to look into are built on asymmetric cryptography where the verifier has one private key he uses to verify his data and there is a public key that is available to everyone. Only the verifier has access to the private key and misplacement of it makes verification impossible.

As the name suggests, this encryption technique is built on key asymmetry, where the encryption and decryption keys are not identical. Public-key (PK) cryptography, as it is also called, is based on mathematical problems which makes key generation very safe but still fairly easy to generate and use. The strength of asymmetric cryptography lies in the fact that it is computationally infeasible to determine the private key from its corresponding public key. This makes it possible for the key owner to publicly spread his public key so anyone can encrypt data and send to him without the fear that his private key will be compromised [7].

3.1.2 Cryptographic hashes

Cryptographic hashes can be used to verify integrity of data or demonstrate the authenticity of a message and is commonly used for solving the data integrity problem.

Cryptographic hash functions take strings as input and map them to short, fixed length (128-512 bits), strings. The output is called hash, message digest or digest. Any changes in the input will produce a different output. A Cryptographic hash function needs to satisfy certain conditions, otherwise it will not be able to be used efficiently or safely. Cryptographic hash function requirements.

Collision resistant: Finding two different messages with the same hash should be infeasible. Finding $Hash(m_1) = Hash(m_2)$ should be extremely difficult.

Preimage resistance: Finding any message m such that $h = hash(m)$ should be very difficult, even with a given hash function h.

Second preimage resistance: Finding a second input that produces the same output as a specified input should be infeasible. Such that $m_1 \neq m_2$ and $hash(m1) = hash(m2)$[8].

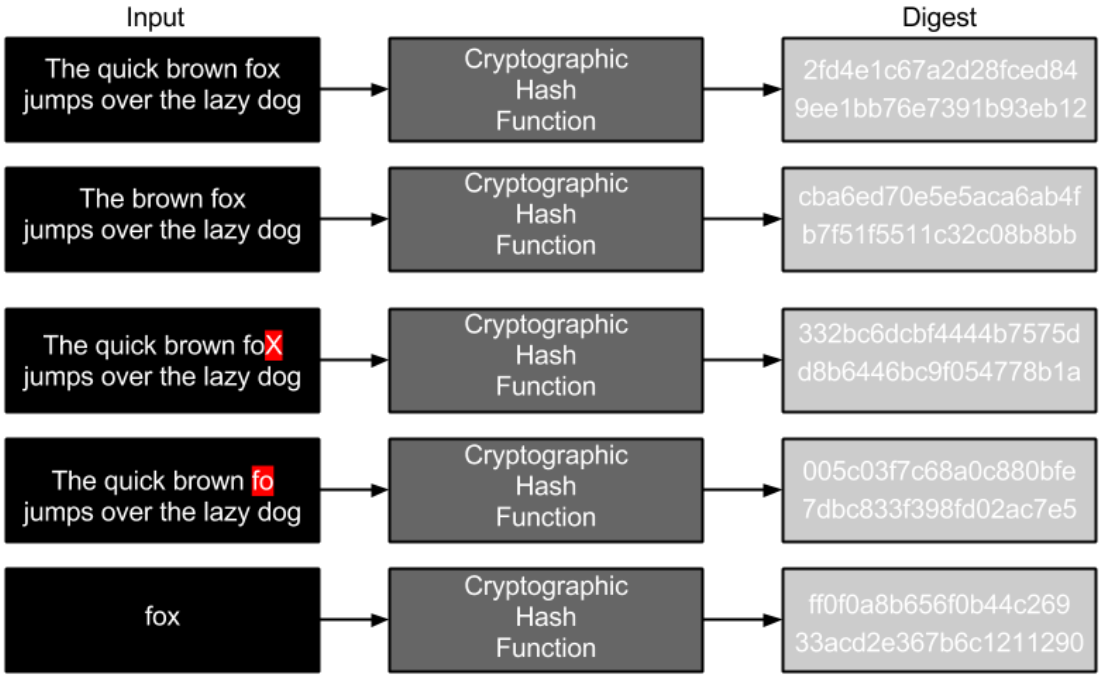

Figure 3.1: Note the small differences in the input result is very different, same length, digests.

3.2 Evaluated methods

All of the research mentioned in this chapter point to different methods and solutions to data integrity problems in different systems and data structures. The research in this chapter has influenced our work to a great extent.

We present one of the most common methods used to achieve data integrity, Digital signature. We briefly discuss how it works and it's usage. We continue on and present a common method for activity logging, Auditing. More precisely Auditing in an Oracle database. Merkle hash tree is the next method we explain. We go in depth on what it is, what it is used for, how it is used and how it is applied to our scenario. We continue on by discussing how Linked timestamping can be applied on the structure of Merkle hash trees to create a chain of hashed trees, and lastly we present one of the newer methods for achieving data integrity, watermarking.

The methods presented in this chapter are chosen based on different reasons. Research and the fact that our external supervisor at Ericsson had greater interest in some methods over others. Digital signature was picked because it is easy to implement and is the predominant method for detecting tampering of data [9]. Auditing is easily integrated and can be used to compare it's results to the ones of the more advanced methods. Due to the fact that Linked timestamping is applied to a binary tree structure we chose to implement and test Merkle hash tree. Digital watermarking was chosen because it is a new method with a lot of potential [10]. It is also the youngest method of the ones presented, previously it has mostly been used in media context for copyright of images. It allows to detect which kind of tampering has concurred, insert, delete, or modification, and also the possibility to locate where the tampering has occurred. However, it is also the most complex method of the ones evaluated.

Ericsson was very much interested in us testing Digital signature, Auditing and Linked timestamping. Their interest in those specific methods influenced us greatly.

We do also present how and for what some of the methods have been used previously, by other researchers.

3.2.1 Digital Signatures

Digital signatures is a way to demonstrate the authenticity of a message and a well known method for data integrity checks. The way a Digital signature works is by computing a hash value of the message, signing the result with a key, and storing it. The integrity can later be verified by comparing the stored digest with the newly computed one. If they are the same one can reach the conclusion that the message has not been altered. This is only true if the hash function used fulfills the requirements specified in the section above [9].

3.2.2 Auditing

Another method we wanted to compare the performance with was to use only the Oracle auditing. What this means is that we will use no signing or verification function but we will turn on full Oracle auditing activity. Full Oracle auditing means that we will audit all SQL operations to a file. For example, selects, inserts, deletes, updates, and procedure calls. The reason we wanted to test this is because it is already supported within Oracle databases and could very well be used to detect tampering. Compared to other methods this one would be used to detect tampering after the action has happened.

3.2.3 Merkle Hash Tree

Related work

A method that has great potential is the *Merkle Signature Scheme*. However, this scheme does have some weaknesses and there are several implementations and variants that exist to solve these. One of the newer implementations available are discussed by Georg Becker [11] where he mentions several different implementations used to improve the original scheme and talks about the major weaknesses of these. The new improvement to the original scheme will address the major weakness of limited amount of keys. We will try to implement this scheme towards an Oracle database and measure the performance.

How it works

Ralph C. Merkle patented a digital signature scheme in 1979. This patented method was called Hash tree or Merkle tree and was based on both hash functions and one-time signatures. The hash tree is of binary structure. Leaves are on the bottom, entities above the leaves are internal nodes, the lone node on the top is the tree root or the public key and all tree entities are nodes, as can be seen in the figure below. The leafs contain hash representations of data and are taken in pairs and hashed together to form a parent node which is then hashed together with its sibling to form a tree root which is the public key of the hash tree. The private keys are the leaves and are authenticated with the help of the publicly available root. The digests can be stored in a database, some secure storage or memory, it all depends on the system. The root (the public key) should be published in a widely witnessed and hard-to-modify media so it is easily accessible and monitored. The size of the Merkle tree is pre-computed which means that only data that is already in the tree can be authenticated. The tree can theoretically be of arbitrary size [12].

We are explaining Merkle hash tree in depth because the next method we present in this chapter is built on the same foundation and principles.

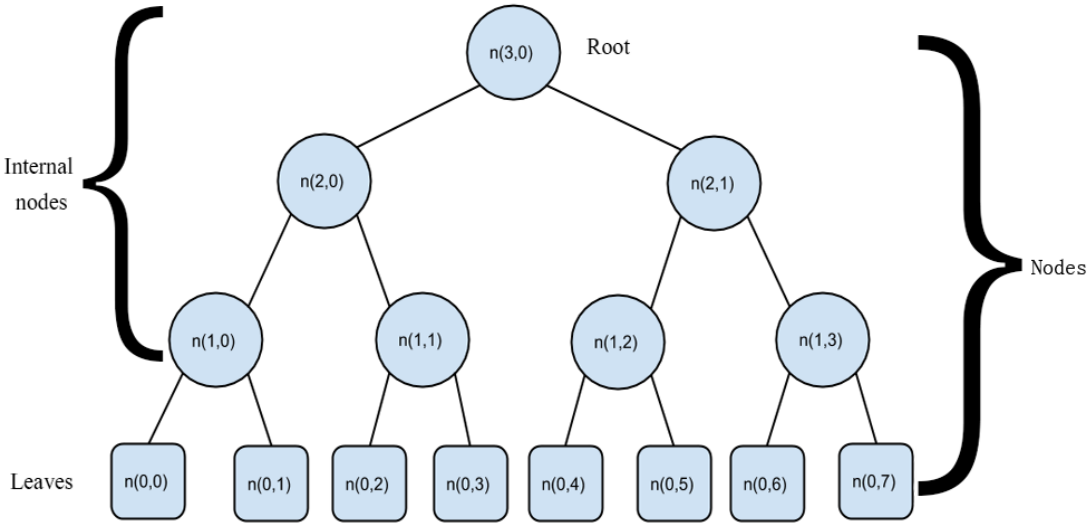

Figure 3.2: In n(y,i) n stands for node, y is the level of the node and i is the number of the node.

Signature generation

For computation of a tree T with height H, 2^H leaves are needed. The tree then has $2^H - 1$ internal nodes and every internal node has two children, 0 and 1. There are in total $2^{H+1} - 1$ nodes N. Each leaf M is hashed with a one-way hash function, SHA-1 for example, producing a one-time signature sig'. The leaves are taken in pairs, n_0 and n_1, n_2 and n_3 and so on. The leaf pairs are concatenated and hashed, the resulting digest becomes their parent node.

$$n_{parent} = Hash(n_{left}||n_{right})$$

As mentioned before, the tree can be of arbitrary size but the largest one we have found had 2^{80} leaves [11, 13].

The *TREEHASH* is a tree generation algorithm. The stack in the algorithm contains the data that is to be stored in the tree. *LEAF-CALC* is a oracle function that takes an leaf index as input and returns a leaf as output.

Algorithm 1 TREEHASH (maxheight)

1. Set $leaf = 0$ and create empty stack.
2. **Consolidate** If top 2 nodes on the stack are at the same height:
   - Pop node value P_{left} from stack.
   - Pop node value P_{right} from stack.
   - Compute $P_{parent} = hash(P_{left}||P_{right})$.
   - If height of P_{parent} = maxheight, output P_{parent} and stop.
   - Push P_{parent} onto the stack.
3. **New Leaf** Otherwise:
   - Compute $P_{leaf} = LEAF\text{-}CALC(leaf)$.
   - Increment $leaf$.
4. Loop to step 2.

Authentication path generation

Every node needs a authentication path A that goes from the node $n_{y,i}$ to the root, where n represents the node, y is tree level in which the leaf is and i is the leaf number. Without this path, verification of data is impossible. There are $y + 1$ nodes in the authentication path where A_0 is the leaf and A_y is the public key, the tree root. All children of the nodes $A_1, ..., A_y$ are needed for the computation of the authentication path A. This means that y nodes are needed, $auth_0, ..., auth_{y-1}$. These nodes and the one-time signature sig' of M make the signature $sig = (sig'||auth_0||auth_1||...||auth_{y-1})$. The signature sig is distributed to the user and only he can verify the validity of the data stored in that leaf [12, 11, 14].

Verification of data

The verifying entity knows the public key, which is the root of the tree, the message M and the signature $sig = (sig'||auth_0||auth_1||...||auth_{y-1})$. The verifier needs to check that sig' is a valid signature of M and if that is the case then the rest of the signature token can be used to verify the validity of the data. The verifier needs to compute the authentication path stored in his signature token. By hashing sig' with the next element in the authentication path $auth_0$, the answer with $auth_1$ and so on the verifier will be left with a digest that, if the data entry in the Merkle tree has not been manipulated, will be identical to the public key of the Merkle signature scheme and thereby valid. The tree traversal requires $O(logN)$ computations and as most $O(log2N)$ space [12, 11].

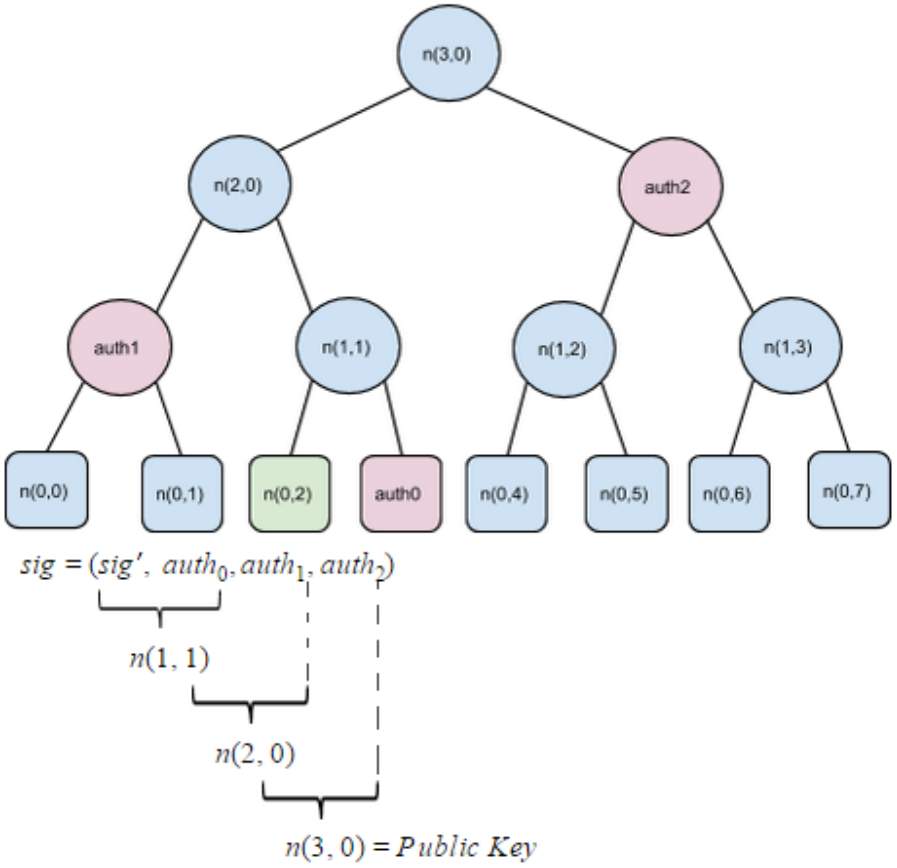

Figure 3.3: The signature token and verification of $n(0,2)$

Adapting the tree

When Ralph C. Merkle constructed the Merkle tree in 1979 he designed it in a way that would make updating and inserting new data in to the tree very inefficient and expensive. The database we are working with is constantly being updated and new entries are being inserted very frequently therefore we needed to improve and adapt the original tree structure to be able to fully test it's efficiency.

Update an entry

When updating an entry the user first needs to verify the integrity of his data and if it's valid he can proceed and update the old entry with a new one. The old entry is overwritten with the hash of the new entry and then the parent, the parents parent and so on need to be recomputed because of the hash chaining dependencies. When the tree has been rebuilt we need to recalculate and redistribute all the signature tokens $sig = (sig'||auth_0||auth_1||...||auth_{n-1})$ to their rightful owners. No user will be able to verify the integrity of his data if the authentication path is not recalculated.

New leaf insertion

When inserting a new leaf to the tree it is not possible to just attach it anywhere on the tree because of it's binary structure. By doing so it will break the concept of having a complete binary tree [15] which a Merkle tree must be in order to work as intended. We chose to solve this problem by generating another tree that is double the size and then transfer all the old entries to the new tree, including the new one we want to insert. When all the values are moved the *TREEHASH* algorithm is called to recalculate the hash values and lastly the authentication path algorithm is called to generate each nodes new authentication path. When everything is done we are left with a Merkle hash tree that is two times larger and has one more entry than the previous tree. All remaining leaves are empty and can later be filled using the insert function.

3.2.4 Linked Timestamping

Related Work

Buldas et al. developed a method [16] based on Merkle Signature Scheme [11]. Their method builds small binary hash trees that are time dependent and connected to each other. Their tops are placed in a publicly available "calendar" which makes the verification of data easier. This method is an improvement over the original Merkle tree method because it dynamically allocates memory and grows depending on how much data needs to be stored unlike its predecessor

that is static. The method is built on Keyless Signatures Infrastructure[1](for a detailed explanation check [16]), which will give many improvements compared to standard asymmetric key cryptography, for example improvement in scaling and quantum immunity. The company behind the method, Guardtime, states: "...signatures are cryptographically linked to the underlying data such that assertions can be made at a later date regarding the time, integrity and provenance of the underlying data." [17].

How it works

We chose to apply the Linked timestamping method on the previously explained Merkle hash tree but instead of generating one large tree for all the leaves small binary trees with four leaves are connected together by their roots. Each tree in the chain is time dependent and the whole chain expands to the right. The root for each Merkle tree will then be connected using *Linear hash chaining*, into a so-called calendar. The public key of the whole structure is a computation of all the hashes in the calendar. The PK is published, preferably in a widely witnessed and hard-to-modify media.
The signing process for many entries involves:

1. *Hashing*: Data is hashed with a cryptographic hash function and the digest is used for representing the data.

2. *Aggregation*: A hash tree is generated each aggregation round to represent all signed entries during the round. All data that is sent to the system during the defined time t is hashed and placed in a tree.

3. *Publication*: All the digests of every aggregation round are stored in another tree, a so-called *hash calendar*, and the top of that tree is publicly published.

Tree aggregation in practice

If $t = 1sek$ and 8 documents enter the system during that time that means that 4 nodes will be placed in tree t and 4 in tree $t+1$. After all documents are placed in trees the system will wait for new ones to enter the queue and do the same.

Verification of data

Whenever a user signs data he receives a verification token from the server. The token contains crucial information that the verifier needs for verification of his data. Depending of the underlying design the token may contain an authentication path, the digest of his signed data, a calendar node etc. Without this unique token the signed data becomes unverifiable.

[1] "A globally distributed system for providing timestamping and server-supported digital signature services."[16]

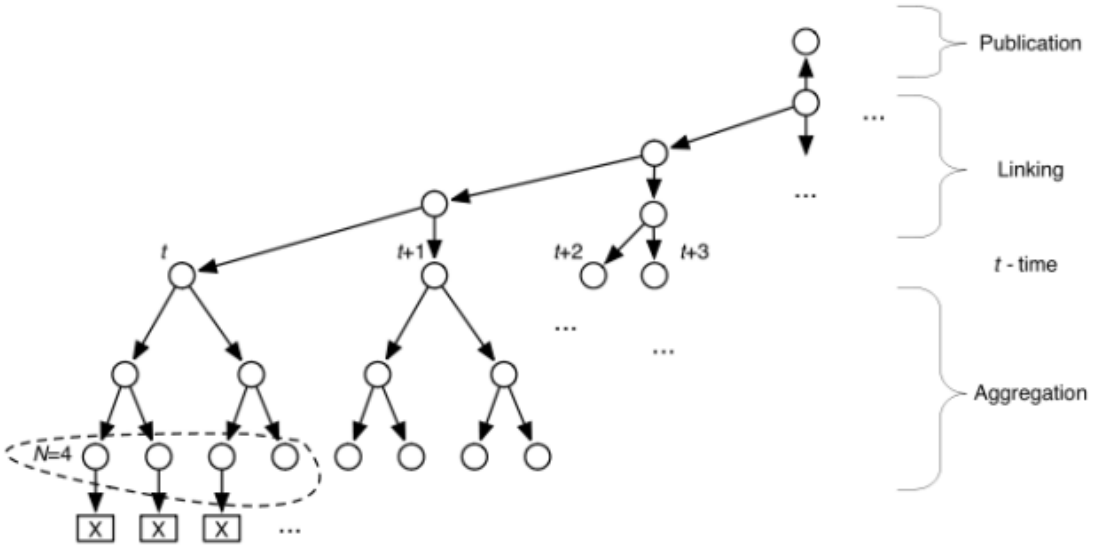

Figure 3.4: Merkle trees connected by their roots forming a timestamp based hash chain structure.

3.2.5 Digital Watermarking

Related work

There are many different methods to generate watermarks on databases but an interesting article is done by Aihab Khan et al. [18] where they state that most watermark schemes introduces distortion in the data. Data distortion makes this method unusable by us due to the fact that we are trying to protect currency and distorting that data is not advised. They introduce their own method that stores the watermark without introducing distortions in the data. However, according to Lancine et al. [10] their scheme is vulnerable to certain attacks. They follow this up by introducing their own scheme that should fill this vulnerability.

How it works

In this section we will talk about Digital watermarking. We will go through what it is, how it works, different types of watermarking, our choice of watermark method and why, and last how this method works.

A Digital watermark is a kind of mark that should only be perceptible to the persons who know it exists and possesses the correct algorithm to see it. This technique works by inputting the data you want to protect into an algorithm, this algorithms varies a lot depending on what classification the mark should possess, more details about this later. After inputting the data into the algorithm one will receive the watermark; this mark should be stored somewhere safe where it

can't be tampered with. When you want to verify that no one has tampered with the data you input the suspicious data into the same algorithm used before and compare this new watermark to the original one stored safely elsewhere. There are many different techniques on how to generate a watermark and the classification of these watermarks differs a lot.
Some watermark classifications are discussed by Raju Halder et al. [19]:
Watermark Information – What kind of information the watermark will add.
Distortion – Whatever the watermark will create distortion in our data or not. For example, a common watermark technique is to change the value of some selected bits in the data. This would introduce distortion since we have to change our original data to implement the watermark.
Cover type – What kind of attribute the mark can be inserted into. For example, some watermarks can only be implemented into integers or text fields.
Granuality Level – What kind of level the watermark will be added at. For example, bit level, character level, attribute level, or tuple level.
Intent of Marking – Tells what the purpose of the mark is. Some common ones are to detect tampering, verify integrity, or traitor detection.

For our scenario we wanted a watermark technique that did not introduce any distortion, since our data is currency; and it should be able to detect tampering and verify integrity. We found two techniques that have these and chose to implement the technique by Lancine et al. [10] over the one by Aihab Khan et al. [18], since the former found a security flaw in the latter's watermark technique.
Matrices - To understand the method by Lancine et al. [10] we first need to understand how the calculate a minor in a matrix. A minor of a cell in a matrix is determined by calculating the determinant of a sub matrix gained by removing that cells row and column. For example, if one has a square matrix A with the dimensions 3x3 and want to calculate the minor of $a_{2,2}$, then this is done by calculating the determinant of the matrix that would be formed if we removed row 2 and column 2 from matrix A. The 2x2 matrix determinant calculation is performed by multiplying the values in one diagonal and subtracting with the product of the other diagonal. The reason behind these calculations is that it is extremely difficult to find two different matrices that have the same diagonal minors and determinant [10].

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} Minor(a_{22}) = Det \begin{vmatrix} a_{11} & X & a_{13} \\ X & X & X \\ a_{31} & X & a_{33} \end{vmatrix} = Det \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix}$$

$$= a_{11}a_{33} - a_{13}a_{31}$$

Lancine et al. [10] divides their Watermark generation algorithm into 4 different steps:

Data Set Partitioning – Where the data set to be protected is partitioned into several different groups of square matrices. Each row in the data set is placed in a group which is decided by doing a hash modulus on the unique primary key of the row. The amount of groups are determined by dividing the amount of rows in the data set with the amount of columns. If any group matrix is missing some rows to form a square matrix then temporary rows are generated.
Watermark Generation - An individual watermark is computed for each group, this is done by sorting the rows in the matrix group in ascending order according to their primary key hash. The watermark is then created with the determinant of the matrix concatenated with the diagonal minors of the group matrix.
Watermark Encryption – All individual watermarks for the groups are concatenated to form a single watermark. This watermark is then encrypted.
Watermark Certification & Registration - The encrypted watermark is concatenated with some meta data, in this case with the owner ID and a universal timestamp. This watermark is later registered at a certification authority (CA) for certification purpose and safe keeping.

Lancine et al. [10] divides the verification process into 4 steps:
Data set partitioning - The first step used in generating the watermark is used again against the suspected data set.
Watermark generation - The second step that is used in generating the watermark is used again against the result from the previous step.
Watermark Extraction - The original watermark is extracted from the CA.
Watermark comparison - The newly created watermark and the original one is compared.

Chapter 4

Method

In this section we will explain our test method, followed by our implementation of the methods to make them work against the system, and an explanation on how the performance evaluation tool works.

We chose to make an experimental approach for evaluating the methods where we try to implement them to be able to sign and validate data in an e-commerce system. To test the performance of the methods we will use an internal performance evaluation tool developed by Ericsson. This tool will simulate real-time use of the system. For our tests we ran simulations with 20000 users, 6000 agents, and 1 issuer; we ran 8 simulations with a duration of 10 minutes for all different methods. To explain why we chose these parameters we must first give a quick summary how the performance evaluation tool works. This will be explained more in-depth in the following section. The performance evaluation tool works in several stages. One of the early stages is when the users and the agents are created and the last stage is when the simulation starts and will run for the given time. During this simulation the tool will perform different transactions with the users and the agents. However, the scope of where we will pick these users and agents is determined based on the simulation time and if this scope stretches beyond the amount of users and agents created then the tool will try to perform transactions with users or agents that do not exist and will so throw errors. Because of this we needed to find a balance where there were enough users and agents for failed transactions not to occur, and after a lot of experimenting we settled for these numbers. The amount of times we ran each simulation was based on running enough tests to get good average results and it was within our time restrictions. All transactions done by the performance evaluation tool were configured, by the creator of the system, to best simulate real-time use of the system, what this means is that a transactions chance to be picked and used by the performance evaluation tool should reflect real-time use of the system, and so, some transactions might be used more than others.

The tests were run on a *Red Hat Enterprise Linux Server release 6.4 (Santiago)* using two *Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz*, 7GB RAM, and an *Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit Production.*

Our hypothesis is that traditional Digital signature are faster in all the differ-

ent operations and transactions compared to Linked timestamping.

4.1 Performance Evaluation Tool

In this section we will talk about the performance evaluation tool. We will try to explain: what the tool is, how it works, which parameters we use, what kind of entities are in the simulation, and what those entities can do.

Our performance tests were done using a performance evaluation tool created and used internally by our external supervisor.

One could say that the tool works in 5 stages from start to finish.

1. First stage involves sending all the code over to the test machine, which is described in the previous section, when the code is over it builds the system.

2. In the second stage the tool clears the database, which means it drops all the tables in the database that might be there from previous test runs.

3. The third stage involves creating entities with the role of being agents for the simulation and filling the database with them.

4. In the fourth stage the tool creates entities as users and fills the database with them.

5. At the last stage the tool starts the simulation. It will run for a given amount of time and will pick users within its scope based on its run time to make transactions with.

Before the tool starts it is possible to change several different parameters. It is possible to change how many minutes the simulation, stage five, is supposed to run, how many agents and users it will create in stage three and four respectively, what kind of transactions the entities can perform in stage 5 and also what their weights are. This means that the higher weight a transaction has the higher chance it has to be picked compared to the other transactions.

In stage three and four it will create several entities that will be assigned different roles, they are to be used to simulate a real life scenario:
Issuer - Takes the role of a bank. They transfer the money in their account between themselves and the agents.
Agent - They use their account to cashin and cashout money for the user. Can be seen as an ATM machine.
User - Normal user of the system. Possible to interact with other users and with the agents.

These are the different transactions the tool will perform during the simulation. Something to take notice of is that the tool will perform several of these transactions using threading; which means that a method that can't handle

threading very well will have a negative performance impact.
Get balance - Simulates an enduser which checks the balance on his account.
Cashout - Simulates an agent initiated cashout, exchanging endusers digital currency for physical.
Cashin - Simulates a cashin by an agent, exchanging endusers physical currency for digital.
Payment - Simulates an end user payment to a service provider.
P2P transfer - Simulates an enduser which transfer money to another enduser.
Get transactionhistory - Simulates an enduser which requests the transaction history on his account.

During the test each transaction is divided into several rounds, where each round performs some action.
Cashin_R0 - Get Accountholder Information.
Cashin_R1 - Cashin.
Cashout_R0 - Get Accountholder Information.
Cashout_R1 - Initiate transfer.
Cashout_R2 - Get Approvals.
Cashout_R3 - Transfer Approvals.
Get balance_R0 - Get Accountholder Information.
Get balance_R1 - Get balance.
Get transactionhistory_R0 - Get Accountholder Information.
Get transactionhistory_R1 - Get Transaction History.
P2P transfer_R0 - Get Accountholder Information.
P2P transfer_R1 - Transfer.
Payment_R0 - Get Accountholder Information.
Payment_R1 - Payment.

There are some things worth noting. First, that *Get balance* and *Get transactionhistory* use only read operations; whereas the other transactions use a combination of read and write operations. This should result in methods with faster verification speed to perform better on *Get balance* and *Get transactionhistory*. Second, in the fifth stage the tool will continue to perform transactions even if there are no users or agents left to work with. An extreme example would be if we use only 10 users in a simulation that is several hours long. After it has used those users it still has several hours left and will try to make transactions on users that have not been created, this will result in failures because it can't find the user, since it doesn't exist, and will ruin the simulation. What this means is that a balance between agents, users, and the simulation time has to be found.

4.2 Performance Measurements

In this section we will try to explain and justify which measurements we chose to use for the methods. For each method we measured the following criteria:
Transactions Per Second - How many transactions can be done each second using the to be evaluated method. This is an important measurement to include for many reasons: it tells us how fast the methods are and it is also a standard measurement unit used when working with databases.
Time for each transaction - The simulation will perform several different transactions, listed and explained in the following section, and each transaction performs several functions that will be measured for each method. The reason we chose to include this measurements is because it might give us an idea how well the methods handle read and writes against the database.

4.3 Implementation

In this section we are going to explain what implementation modifications we have done on the data integrity methods explained in chapter 3 to make them more suitable for the purpose of this thesis.

All the methods tested in this thesis used the same cryptographically secure hashing algorithm with the same key length.

4.3.1 Data verification in Linked timestamping

There are two verification types in the Linked timestamping method, one is performed by the user when he needs to verify his data and the second one is performed by the system to verify the integrity of the whole tree structure.

The user data verification method consists of two steps, the first is one-to-one verification where the digest of his data is compared to the one stored in the tree and the second one is by computing all the chain hashes (authentication path) of the tree where the node is stored. If the first verification step fails the second one will not be performed and if the whole verification process fails it is either because the data integrity of the verifying leaf or any other leaf in the tree has been compromised. This verification process is performed on-line when the integrity of the verifier needs to be proven.

The tree verification method is done by computing the root of every tree and then comparing the digest to the one that its calendar node points to, if all comparisons pass it means that the whole tree is verified. This verification process is triggered periodically, performed on-line and should be performed frequently. If the system has many users that perform many transactions it would be desirable

to schedule a full tree verification once or twice a day, however it can be a time consuming process if the tree aggregation has been going on for a long time.

4.3.2 Linked timestamping tree aggregation

The performance evaluation tool is created in such a way that it creates data hashes and sometimes verifies them immediately, this creates a problem in the original implementation because the aggregation round might not be over and the software wants to verify data that has yet not been placed in a tree and the verification will fail. This problem can be solved by making the performance evaluation software wait the length of one aggregation round. The problem with this solution is that the wait for the aggregation round to finish will lower the transaction per second time drastically and make the test results inconclusive.

To evade this problem we chose to implement our aggregation process a bit different. Instead of waiting t length of time for the aggregation round to finish we chose to make a tree as soon as the testing software wanted to hash new data.

4.3.3 Storage

The company that provides the e-commerce service needs to secure the system and protect the sensitive user information. To be able to do so the company needs different security methods, as the ones described in this thesis, implemented in the system and also secure storage for keys and other relevant data.

Storing sensitive data like fingerprints and keys can be performed in many different ways. In our implementation we did two types of testing. The first one that was performed locally (off-line) stored its keys in a secure *Java KeyStore*[1] and the computed verification digests in the memory, for fast verification. Every time data was hashed the key was fetched from the key store and the resulting hash was stored in memory. The second part of testing that was performed by the performance evaluation tool stored its keys in a *Hardware Secure Module* (HSM) and the digests in a database. Data that needs to be hashed is sent to the HSM for computation, after the computation the HSM answers with a digest that is placed in the database. Upon verification the HSM is asked to compute the digest for the data that needs to be verified and that digest is compared to the one stored in the database.

A HSM is a very secure module that has many security related functions. It can encrypt and decrypt, digitaly sign and hash data and also provide Message Authentication Codes[2] (MAC). E.g. as we earlier described in our scenario it is undesirable if someone was able to manipulate data in the database. Therefore a HSM can be used for key storage and hash computation and because the keys are

[1] A secure repository for certificates and keys.

[2] A hashing function that takes a key and a message as input and produces a digest as output.

maintained by the HSM it is theoretically impossible for anyone else to reproduce a valid digest for that sensitive data. So when verification data is received the HSM is asked to verify the digest and if it fails appropriate measures can be taken. A HSM provides also the functionality of key management. All generated keys should be stored on the device itself and frequently backed up on to another HSM or smart cards. The hardware security module is also tamper resistant. If it detects physical penetration, anomalous temperatures or electrical activity the device will erase all sensitive data from it's storage. This is to protect data stored on the device if someone has gained physical access to it [20].

In summary the keys that are stored off-line in a *Java KeyStore* are retrieved much faster than the ones stored and calculated by the *HSM*. This makes the off-line verification much faster, however the security of the stored keys is much greater when stored in the *HSM*.

4.4 Database auditing

"Auditing is the monitoring and recording of selected user database actions. It can be based on individual actions, such as the type of SQL statement executed, or on combinations of factors that can include user name, application, time, and so on. Security policies can trigger auditing when specified elements in an Oracle database are accessed or altered, including the contents within a specified object."[21]

We are logging all database activity when auditing is turned on. That includes all select, insert, delete, procedures and many more. This means that all the activity is being recorded to the table *SYS.AUD$* and later dumped to file. This file would then be used for integrity and tampering detection.

4.5 Verification

To be able to show that the methods presented in this thesis really solve the data integrity problem we need test if they can detect that database entries have been manipulated. This will be tested by allowing the performance evaluation tool to compute hashes and then inject changes to some of them and see how the tool reacts when it tries to verify. Auditing is not able to verify data integrity by itself, it is however used for monitoring the database and can detect tampering. Methods that we are unable to integrate with the performance evaluation tool will be tested by ourselves. The off-line and performance evaluation tool tests were based on the same principle and performed in the similar manner. 1000 erroneous entries will be injected into storage and later upon verification when/if the tool detects these entries we will note the detected data and where it was detected. These 1000 erroneous entries will be random numerical values in the

form of a string. This is to be able to confirm that the methods indeed could detected fraudulent behavior.

4.6 Journal system

The currency stored for each account is stored as a journal system. What this means is that to get the current balance for an account you traverse through all the transaction entries and add or subtract depending on what is stored and the result is then the current balance for that account. The result of this is that it takes many read operations to get the current balance for one account.

Chapter 5

Results

In this chapter we present the results of our data integrity method evaluation. However, due to the lack of time we were unable to fully test all the implemented methods. Digital signatures, Linked timestamping, and database auditing were integrated and tested with the performance evaluation tool and Merkle hash tree and Digital watermarking were only tested off-line by us in a controlled environment.

5.1 Empirical analysis

In this section we will present the results of our empirical analysis in the form of benefits and drawbacks of each evaluated method.

5.1.1 Digital Signature

Digital signatures are relatively easy to integrate into the system as all standardized methods are already implemented in most programming languages. As the method only computes digest representation of data it is easy to insert, update and verify entries. However if the verification fails you can not know which data has been manipulated, the one coming from the verifier or the one in storage. And with access to the database manipulations can easily be made to single entries which will not be detected until verifiers try to check their integrity.

Benefits

- Easily implemented.
- Fast verification.
- Easy to insert and update values.

Drawbacks

- Mismatch in verification only means that one of the values has been manipulated.

- Possible to manipulate data in storage without affecting other entries.

5.1.2 Auditing

Auditing is easy to use since it is already supported in the Oracle database. All one needs to do is enable it in the Oracle database and decide whether to save the audits on a file or in the database. However, there are some problems with this. The problem is that the Oracle will audit so much that it is almost impossible to go through the file by hand and more sophisticated methods must be created to filter through the file.

Benefits

- Already supported in Oracle databases.
- Capable of detecting tampering.
- Rules for auditing are simple to define.

Drawbacks

- Must filter through the file for anomalies. Either using scripts, AI[1] (Artificial Intelligence), or manually.
- Significant performance loss.
- Must safely manage the audits.

5.1.3 Merkle Hash Tree

The Merkle tree was both easy to understand and implement. The biggest issue with this method is how one should handle insertion of new nodes. For example, if one creates a Merkle tree with four leafs, how should one most effectively insert a fifth element? One solution tried that works in theory is to initialize the tree with a very large amount of leafs. However, this solution only postpones the problem to a later time; it could also create a future problem with memory, since the empty nodes still needs to be saved.

[1] Artificial Intelligence is "The theory and development of computer systems able to perform tasks normally requiring human intelligence..." [22]

Benefits

- Easily implemented.
- Fast verification because of its binary structure.
- Leaves can be updated.

Drawbacks

- Whenever a leaf is updated authentication path for all other leaves needs to be recomputed.
- Expanding the tree is costly because a new tree, twice the size, needs to be created and previous values need to be inserted.
- Tree will drastically increase in size over time.

5.1.4 Linked Timestamping

The Linked timestamping method was relatively easy to implement because it is built on the structure of Merkle trees which we have already implemented earlier but has higher integrity due to linked timestamping. The problematic part is that entries can not be updated so all updates become inserts and old nodes become deprecated but still take up space.

Benefits

- Fast insertion.
- Fast verification because of it's binary structure.
- Keyless signature infrastructure.[2]
- Higher integrity due to timestamping.
- Easy to pinpoint data manipulation due to the small tree size.

Drawbacks

- Node updates are impossible because of timestamp dependencies.
- Updated nodes become deprecated and new ones have to be inserted increasing the structure size.

[2] "A globaly distributed system for providing time-stamping and server-supported digital signature service."[16]

5.1.5 Digital Watermarking

The Digital watermark was the hardest to implement and it is questionable if it will work efficiently in a system with this many updates since the watermark generation is quite costly, they also say that the method should work best when used only a couple of times a day and that future work should focus on optimize the method. According to Lancine et al. [10] the security in the watermark lies in that it is hard to change values in the data set so that a group is generated with the same determinant and diagonal minor values.

Benefits

- Possible to detect where the tampering has occurred.
- Possible to detect what kind of tampering has happened (e.g inserting, modifying, inserting).

Drawbacks

- Hard to implement.
- Handles frequent updates and inserts poorly.

5.2 Verification

When erroneous data was injected into the database all of the methods except Auditing were able to detect the fraudulent behavior. The performance evaluation tool reacted by outputting the error message *Fingerprint Verification Error*. As mentioned earlier in Method, Auditing lacks the verification feature and was therefore not tested.

When a mismatch is found using the Digital signature method you are unable to identify which data has been compromised, the one in storage or the one that is newly computed.

Linked timestamping was able to pinpoint the error down to the tree where the injected data was stored. This means that one, some or all node/s in that tree have been manipulated. The same is true for the Merkle tree. However you can not pinpoint which leaf has been manipulated and the only way to find those leafs is by comparing original signatures to the ones stored in the tree.

The watermarking method could detect all changes. However, depending on the percentage of data changed and how spread the changed entries were, it was not always able to tell exactly which values were changed.

With Auditing the verification process would take place after the fact that it has happened. This would involve going through the audit logs to find changes or tampering to the data.

5.2.1 Transactions

In this section we are presenting our transaction findings.

The transaction results will be presented within a table where all the tests have been summarized and an average is presented. The rounds for all methods are compared to how much they lost or gained compared to a baseline, what this means is that we will show the percentage gain or loss. The baseline is in this case running the tests without any signature or verification process and no Oracle auditing. Every test was performed on approximate 30000 records except *Auditing* that was performed on approximate 20000 records. The raw numbers in milliseconds are presented in a table and can be found in the appendix, Table 8.1.

Method / Transaction	No Protection	Digital Signature	Auditing	Linked Timestamping
TPS[1]	100	96	73	88
Cashin R0	100	93	55	87
Cashin R1	100	97	85	87
Cashout R0	100	91	57	85
Cashout R1	100	95	50	96
Cashout R2	100	97	54	106[2]
Cashout R3	100	97	106[2]	87
Get balance R0	100	93	56	86
Get balance R1	100	97	59	95
Get transaction-history R0	100	92	56	85
Get transaction-history R1	100	97	72	88
P2P transfer R0	100	92	56	87
P2P transfer R1	100	98	71	89
Payment R0	100	93	56	87
Payment R1	100	97	83	85

Table 5.1: Time for each round shown as percentage gain or loss for each method tested compared to the baseline, No Protection

[1]TPS: Transactions Per Second.

[2]Will be explained in the analysis, *chapter 6*.

Chapter 6

Analysis

In this chapter we analyze and discuss the results from the performance evaluation tool and what performance impact they have.

6.1 Performance evaluation

As we mentioned in chapter 5 the transactions performed was done on approximately 30000 records with the exception for *Auditing* which was performed on approximately 20000 records. The reason for this big difference in records is based on the test duration and speed. When database auditing is turned on the same storage is being used for storing transaction data and for writing the audit log which means that the same input/output (IO) interface is being used by the system. Therefore the writing speed is much slower and a test that runs for X amount of time will not be able to interact with the same number of records as when the IO is used by one operation. Also when there are many frequent writes to the same IO a lot of memory is being consumed. Another thing that can also affect the times presented is that the Digital signature uses threads for both its signing and verification; while the Linked timestamping in its current implementation does not use threading when creating the signature.

6.1.1 Auditing

As can be seen in Table 5.1 the auditing method did not perform particularly well compared to the other methods. The reason for this is because we used full auditing activity for the Oracle database and this will result in that every time we want to use a SQL operation in the Oracle the database also has to write to the audit log that it performed that SQL operation.

The results presented for Auditing in Table 5.1 were all very low compared to the other methods except in *Cashout R3*. We did many tests to find out why and how it was possible that Auditing got such good results only in that particular round. We checked the code for the performance evaluation tool and it was identical to the one used in the baseline and there were no exceptions, failures or errors logged during the tests. We performed a longer 20 min tests to see if the results still were high in *Cashout R3* which they were. Based on all the tests performed and the fact that the only difference between the baseline test and the Auditing test is that Oracle auditing is turned on we draw the conclusion that there is something the Oracle database does that we are unable to explain.

During our evaluation of Auditing while using the performance tool it took around 20 minutes to set everything up and 10 minutes to run the simulation. During this time the file we generated was of approximately 450MB in size. This file contained around 1.5 million rows and each row contains one SQL operation. To filter through this file and find possible proof of tampering is almost impossible by hand and a script or an AI should be used. One would also need someway to manage these files since they will quickly grow in size the longer the system is in use. One possible solution would be to remove files after some time but then you run the risk that if something has been missed it will stay undetected.

6.1.2 Digital Signature vs Linked Timestamping

Here we present a performance evaluation summary of the transaction methods presented in chapter 5. We are comparing Digital signature and Linked timestamping against the baseline (presented in Table 5.1) and it can be noted that Digital signatures are generally faster method in all measurements except *Cashout R2*. *Cashout R2* is the round 2 operation that only *Cashout* performs. It is a simple request operation, it requests approvals from the database and upon receiving that data the transaction proceeds and next round can begin. A possible reason for this performance difference is the fact that this operation is only a read that does not require any kind of verification and it is only a coincidence that Linked timestamping was faster than both Digital signature and the baseline. If we look at the performance measures presented in Table 8.1 we can see that time difference is not very big $\sim \pm 10$ms.

6.1.3 Transactions per second

If we analyze the results for TPS, the most important measurement, we can see that by taking the smallest measure of security, Auditing, we lose 27% of the initial transaction speed. By implementing more advanced and effective methods data integrity, Digital signatures and Linked timestamping, we only lose 4% and 12% respectively.

6.1.4 Empirical analysis

In this section we present our empirical analysis of all the methods.
Digital signature: Has fast verification because it is a simple comparison between two hashes. Its integrity is somewhere in the middle, compared to the other methods, because of the verification results we previously presented (see section 5.2). Data inserts and updates are fast and easy to perform as you only need to change stored database hashes. All standardized digital signature algorithms are available online and easily integrated. Size is not affected.
Auditing: Has a low verification speed because the auditing is not well suited for this. It has to be done after the fact and must have some tool to go through the logs. The integrity is average since depending on the filters used on the logs one can detect tampering. For example, tampering done by someone with access to the database will audit the name of the user, what kind of operation the user performed, when it happened, and where the user did it. The data inserts and updates are easily implemented since it is done in the database. However, auditing everything will create large files that needs to be managed somehow.
Merkle hash tree: The verification speed is somewhere in the middle compared to the other methods as many hash computations need to be performed to be able to verify data. Data integrity is also somewhere in between the other methods because it is hard to pinpoint to which entry has been manipulated (see section 5.2). Inserts and updates are of medium/high complexity based on the fact that authentication paths need to be recomputed and tokens need to be redistributed for all leafs. Implementation is of medium complexity because it is a standard binary tree implementation. Size increase over time is high as when a tree is full a completely new tree of twice the size needs to be built.
Linked timestamping: The verification speed of Linked timestamping is almost the same as Merkle tree hash as it is built on the same method. Integrity is a bit higher than Merkle tree hash because of the timestamping that is applied to each calendar node. Inserts are very easy/fast as you are always appending new trees to the hash chain. Data updates are impossible due to the fact that every tree was created with a timestamp and that time is unknown after computation and can't be reused. It is not very complex to implement as it uses Merkle trees and hash chaining to create its structure. The size increases over time as updates are impossible and they become new inserts so over time the calendar can become relatively large.
Digital watermarking: The verification speed in Digital watermarking is quite slow due to its need to compute several matrix calculations every time it needs to verify. Since we did not manage to test the speed for our self this value is based on the view of the author. It does however give high integrity since it is very hard to tamper with the data undetected and it is possible to detect what and where the tampering has occurred. Both inserts and updates of data are expensive in watermarking due to the need to compute several matrix calculations each time.

Method / Option	Digital signature	Auditing	Merkle hash tree	Linked timestamping	Digital watermarking
Verification speed	High	Low	Medium	Medium	Low
Integrity	Medium	Medium	Medium	Medium/High	High

Table 6.1: Summary of the empirical analysis. How fast the methods verify data compared to the other methods in the table and how high data integrity the methods provide compared to each other. The scale goes from *Low* which is bad to *Hight* which is good.

Method / Option	Digital signature	Auditing	Merkle hash tree	Linked timestamping	Digital watermarking
Data insert complexity	Low	Low	Medium/High	Low	High
Data update complexity	Low	Low	Medium	Not possible	High
Implementation complexity	Low	Low	Medium	Medium	High
Size increase overtime	Low	High	High	Medium	Low

Table 6.2: Summary of empirical analysis. How well do the methods handle data inserts and updates, how hard was the implementation of the method and how the structural size of the methods increase over time. All the results are when methods are compared to each other. The scale goes from *Low* which is good to *High* which is bad.

6.1.5 Technological development

In this section we present how future technological development possibly may affect the methods we chose to evaluate. All of the information presented below is for the scenario explained in the beginning of this thesis and should not be applied to other scenarios.

Digital signature: As previously explained Digital signatures are built on hash functions and keys. If the specific hash function used gets cracked in the future and it no longer is *Collision resistant* then the attacker can form his own input and still produce the desired output hash.

Auditing: Does not have any specific security preferences that need to be fulfilled for it to be called Auditing. In our scenario it is just an option that is turned on in the database and logs are produced. The security in our scenario lies in how those logs are stored and who has access to them. Auditing can not automatically detect fraudulent behavior so when there are suspicions of such behavior someone trusted needs to analyze the logs. Storing these audits securely is of great importance and it is hard to say how technological development will affect the stored audits as there is no one right way of storing them. A suggestion is storing them in encrypted storage so that more then one person are needed with their security keys to access the logs, another suggestion is to make the audits read only so they can be read and analyzed but not changed after their creation, or one can apply the digital signature method on the audit logs; this way one will know if they have been tampered with.

Merkle hash tree and Linked timestamping: These are built on the same security fundamentals. Both methods can use the same cryptographic hash functions to compute the digests that represent the data. Therefore if any of the cryptographic hash requirements are cracked the values can be manipulated and if done correctly the evaluated methods will not be able to detect the fraudulent behavior.

Digital watermarking: The security in Digital watermarking lies in the fact that it is extremely hard to calculate the minor of a matrix and the determinant of the resulting matrix and produce a desired result. Therefore if someone in the future makes it possible to do this calculations and produce a desired result the security of Digital watermarking will be broken.

Chapter 7

Conclusions and Future Work

In this final chapter we summarize this thesis, present our conclusions, do a final analysis and lastly we propose what could be done in the future on this topic.

7.1 Conclusions

In our thesis we discussed different methods for achieving data integrity in a database. The purpose of this is that we want to solve the data integrity problem that arises from the presented scenario where a large e-commerce systems is the victim of internal fraud.

The methods tested were traditional Digital signature, Merkle hash tree, Oracle auditing, Linked timestamping, and Digital watermarking. They were implemented and tested to see if they truly do detect fraudulent behavior and three of the methods were also integrated and performance evaluated by a performance evaluation tool. A empirical analysis was also performed to see the benefits and drawbacks of each method.

We saw in our analysis that the implemented methods were indeed able to detect fraudulent behavior, however some better than others and will be presented in descending order of able to detect fraud against the integrity.

- **Digital Signature**: Detects manipulation but only when that exact entry is being verified.
- **Merkle hash tree**: Detects manipulation whenever a leaf in the tree is being verified. Can't pinpoint to the exact leaf that has been changed without further investigation.
- **Auditing**: Possible to detect tampering but it is a time consuming process and can only be detected after it has happened.
- **Linked timestamping**: Same results as the ones for Merkle hash tree, however the trees are small and its easier to pinpoint the changed leaf.
- **Digital watermarking**: The watermarking method could detect all changes. However, depending on the percentage of data changed and how spread the

changed entries were, it was not always able to tell exactly which values were changed.

Since we did not manage to implement two of the methods against the system only three of them could be used with the performance evaluation tool. Looking at the result from the transaction times we can see that enabling Auditing from the Oracle database quickly reduces our overall transactions per second by a lot. The more interesting values are between the Digital signature and Linked timestamping, where Digital signature suffered a lose of 4% from the baseline and Linked timestamping suffered a loss of 12%. This can be an acceptable loss for the increase gain in integrity.

Based on our work we reach the conclusion that for a system such at this Merkle trees work well to detect integrity fraud and looking at previous work (presented in chapter 3) and our own implementation they do provide adequate verification and signature times. Of the three methods tested Linked timestamping works well since it creates quite small trees consistently compared to the standard Merkle hash tree that would eventually need to make a large increase in size to fit more leafs.

7.2 Future Work

Due to limited time we were not able to integrate Merkle hash tree and Digital watermarking with the performance evaluation tool and optimize the methods for best performance. Therefore suggested future work is integration and performance analysis of the remaining data integrity methods.

Performance optimization could be done on the Linked timestamping method by:

1. Data signing should be performed on every aggregation round when either the storage of maximum number of leafs is filled or the aggregation round time has passed.

2. Make the signing method thread safe.

Also to be able to draw conclusions on how much Auditing is responsible for the distortion in our results one could in their future work implement different database communication interfaces. One interface for communication with the system and one for Auditing, and then compare their findings to our and see how much the IO affects the results.

As we stated earlier Auditing can not automatically detect fraudulent behavior it can only log database activity. Possible future work could be to implement an AI, script or rule based system that can go thorough the logs and flag the possible frauds.

References

[1] DB-Engines, "Db-engines ranking," may 2014. http://db-engines.com/en/ranking/relational+dbms accessed 6/9/2015.

[2] I. Shingari and S. S. Verma, "Achieving data integrity by forming the digital signature using rsa and sha-1 algorithm," tech. rep., M.Tech Computer Science Engineering, Dept. of Computer Science Engineering Mody Institute Of Technology and Science, 2013.

[3] M. J. Harvey, "E-commerce. computer sciences. 2002. encyclopedia.com," may 2015. http://www.encyclopedia.com/doc/1G2-3401200040.html accessed 6/9/2015.

[4] L. Corp, "data-integrity," feb 2015. http://www.yourdictionary.com/data-integrity accessed 6/9/2015.

[5] M. Rhodes-Ousley, *Information Security - The Complete Reference*. The McGraw-Hill, second ed., 2013.

[6] V. Gunupudi, "Performance evaluation of data integrity mechanisms for mobile agents," Master's thesis, University Of North Texas, 2003.

[7] B. Kalinski, "The mathematics of the rsa public-key cryptosystem," tech. rep., RSA Laboratories, apr 2006.

[8] P. Rogaway and T. Shrimpton, *Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance*, vol. 3017. Springer Berlin Heidelberg, 2004.

[9] G. Sivathanu, C. P. Wright, and E. Zadok, "Ensuring data integrity in storage: Techniques and applications," in *Proceedings of the 2005 ACM Workshop on Storage Security and Survivability*, StorageSS '05, (New York, NY, USA), pp. 26–36, ACM, 2005.

[10] L. Camara, J. Li, R. Li, and W. Xie, "Distortion-free watermarking approach for relational database integrity checking," *Mathematical Problems in Engineering*, vol. 2014, mar 2014.

[11] G. Becker, "Merkle signature schemes, merkle trees and their cryptanalysis," Master's thesis, Ruhr-Universisät Bochum, http://goo.gl/KNeVhj, feb 2008.

[12] R. C. Merkle, *Secrecy, Authentication, and Public Key Systems*. PhD thesis, Stanford University, jun 1979.

[13] S. M. Markus Jakobsson, Tom Leighton and M. Szydlo, "Fractal merkle tree representation and traversal," tech. rep., RSA Laboratories, MIT Laboratory for Computer Science, Akamai Technologies, 2003.

[14] B. Ederov, "Merkle tree traversal techniques," Master's thesis, Darmstadt University of Technology, apr.

[15] NIST, "Complete binary tree," jun 2015. http://xlinux.nist.gov/dads/HTML/completeBinaryTree.html accessed 6/9/2015.

[16] A. Buldas, A. Kroonmaa, and R. Laanoja, "Keyless signatures' infrastructure: How to build global distributed hash-trees," in *Secure IT Systems* (D. G. Hanne Riis Nielson, ed.), vol. Volume 8208 2013, pp. 313–320, 18th Nordic Conference, NordSec 2013, Ilulissat, Greenland, oct 2013.

[17] Guard, "Keyless signature infrastracture," feb 2015. http://guardtime.com/ksi-technology accessed 6/9/2015.

[18] A. Khan and S. A. Husain, "A fragile zero watermarking scheme to detect and characterize malicious modifications in database relations," *The Scientific World Journal*, vol. 2013, feb 2013.

[19] R. Halder, S. Pal, and A. Cortesi, "Watermarking techniques for relational databases: Survey, classification and comparison," *Journal of Universal Computer Science*, vol. 16, dec 2010. http://goo.gl/V4lKnd.

[20] S. I. I. R. Room, "An overview of hardware security modules," tech. rep., SANS Institute, 2002.

[21] Oracle, "Database auditing: Security considerations," apr 2015. http://goo.gl/g3QffU accessed 6/9/2015.

[22] O. Dictionaries, jun 2015. http://goo.gl/L8OdrP accessed 6/9/2015.

Chapter 8

Appendices

8.1 Raw transaction numbers

Method / Transaction	No Signing	Digital Signature	Auditing	Linked Timestamping
Transaction Per Second	51	49	37	45
Cashin R0 (ms)	379	407	687	434
Cashin R1 (ms)	3888	3988	4568	4487
Cashout R0 (ms)	391	428	690	460
Cashout R1 (ms)	903	948	1791	938
Cashout R2 (ms)	261	270	486	246
Cashout R3 (ms)	1438	1489	1352	1658
Get balance R0 (ms)	380	409	678	443
Get balance R1 (ms)	1032	1068	1761	1081
Get transaction-history R0 (ms)	378	411	679	445
Get transaction-history R1 (ms)	2057	2114	2849	2326
P2P transfer R0 (ms)	379	413	682	438
P2P transfer R1 (ms)	4669	4758	6570	5250
Payment R0 (ms)	380	407	677	439
Payment R1 (ms)	4962	5109	6000	5824

Table 8.1: Average result for each method with ∼30000 records