Verifiable Outsourced Database Model: A Game-Theoretic Approach

Faryed Eltayesh

A Thesis

in

The Department

of

Concordia Institute for Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Information Systems Security) at

Concordia University

Montréal, Québec, Canada

January 2017

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Faryed Eltayesh**

Entitled: **Verifiable Outsourced Database Model: A Game-Theoretic Approach**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Information Systems Security)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

Chair
Dr. Abdessamad Ben Hamza

External Examiner
Dr. Emad Shihab

Examiner
Dr. Roch Glitho

Supervisor
Dr. Jamal Bentahar

Co-supervisor
Dr. Rabeb Mizouni

Approved by
Chair of Department or Graduate Program Director

2017

Amir Asif, Dean
Faculty of Engineering and Computer Science

Abstract

Verifiable Outsourced Database Model: A Game-Theoretic Approach

Faryed Eltayesh

In the verifiable database (VDB) model, a computationally weak client (database owner) delegates his database management to a database service provider on the cloud, which is considered untrusted third party, while users can query the data and verify the integrity of query results. Since the process can be computationally costly and has a limited support for sophisticated query types such as aggregated queries, we propose in this research a framework that helps bridge the gap between security and practicality. The proposed framework remodels the verifiable database problem using Stackelberg security game. In the new model, the database owner creates and uploads to the database service provider the database and its authentication structure (AS). Next, the game is played between the defender (verifier), who is a trusted party to the database owner and runs scheduled randomized verifications using Stackelberg mixed strategy, and the database service provider. The idea is to randomize the verification schedule in an optimized way that grants the optimal payoff for the verifier while making it extremely hard for the database service provider or any attacker to figure out which part of the database is being verified next.

We have implemented and compared the proposed model performance with a uniform randomization model. Simulation results show that the proposed model outperforms the uniform randomization model. Furthermore, we have evaluated the efficiency of the proposed model against different cost metrics.

Acknowledgments

I am heartily thankful and owe my deepest gratitude to my supervisor, Dr. Jamal Bentahar, whose encouragement, guidance and support from the initial to the final stage have enabled me to develop a profound understanding of the subject and a passion for the scientific research field in general. I would like also to express my sincere gratitude to my co-supervisor, Dr. Rabeb Mizouni, for her guidance and support along the way.

I am forever grateful for the unconditional endless source of love and support, my beloved family. I would like also to show my gratitude to my friends with whom discussions were a valuable factor to the success of this research.

Lastly, I would like to thank my Universities, Misurata University and Concordia University, as well as the Libyan government for funding and supporting this research.

Contents

List of Figures

List of Tables

List of Acronyms

AS	Authentication Structure
ASAP	Agent Security Via Approximate Policies
DOBSS	Decomposed Optimal Bayesian Stackelberg Solver
LAX	Los Angeles International Airport
MHT	Merkle Hash Tree
MILP	Mixed-Integer Linear Program
MIQP	Mixed-Integer Quadratic Program
ODB	Outsourced Database
VDB	Verifiable Database
VO	Verification Object

Chapter 1

Introduction

This chapter introduces the research context, motivation, problem definition, research questions that we aim to answer, and summary of contributions. We conclude this chapter with an overview of the thesis organization.

1.1 Research Context

Database outsourcing has become one of the most attractive cloud computing services (Mykletun, Narasimha, & Tsudik, 2003; Narasimha & Tsudik, 2006; Xie, Wang, Yin, & Meng, 2007). In the outsourced database (ODB) model, three entities are involved: the database owner, who is considered to be a resource constraint client, database users who use and interact with the database, and third-party database service provider. The database owner delegates his database management to the database service provider, who provides mechanisms to enable the database owner and users to create, update, and query the database as needed (Li, Hadjieleftheriou, Kollios, & Reyzin, 2006; Narasimha & Tsudik, 2006; Wang, Chen, Huang, You, & Xiang, 2015). By outsourcing database management to the cloud, organizations take advantage of all benefits cloud computing has to offer including unlimited on-demand computational power in a pay-as-you-go basis (Goodrich, Tamassia, & Triandopoulos, 2008; Wang et al., 2015; Zhang & Safavi-Naini, 2014).

Despite the enormous advantages ODB offers, database outsourcing poses serious data security concerns (Chen, Li, Huang, Ma, & Lou, 2015; Zhu et al., 2013). Since data is a highly important asset to organizations and since it is stored outside the organization boundaries and control by potentially untrusted third-party, data privacy and integrity become the most challenging issues for organizations (Narasimha & Tsudik, 2006; Yuan & Yu, 2013). The database service provider can be malicious or may not take sufficient security measures to protect the hosted data against malicious insiders or outsiders (Narasimha & Tsudik, 2006; Thompson, Haber, Horne, Sander, & Yao, 2009). Therefore, it is important to provide a mechanism to secure the outsourced data from both a malicious database service provider or any malicious outsider (Narasimha & Tsudik, 2006). In addition to data integrity assurance, the mechanism has to assure the completeness of data. Data completeness ensures that the result set of any query is complete and that there was no data deleted from the database in an unauthorized way (Wang et al., 2015).

1.2 Motivation

The integrity concerns of outsourced databases were the drive behind the emergence of the verifiable databases (VDBs) (Goodrich et al., 2008; Narasimha & Tsudik, 2006; Yuan & Yu, 2013). In the VDB model, the database owner creates and uploads to the database service provider on the cloud the database and its special authentication structure (AS). Authentication structures are a type of data structures that use cryptographic primitives such as digital signatures to sign the underlying data and ensure its integrity. Next, when database users query the data, the database service provider responds to users' queries with the query results and a verification object (VO) extracted from the AS. Finally, users can use the VO to ensure that integrity of data has not been violated. The process of creating the AS and the verification uses cryptographic primitives such as digital signatures that ensure the data has not been manipulated by anyone other than the authorized parties (Chen, Li, Weng, Ma, & Lou, 2014; Narasimha & Tsudik, 2006; Wang et al., 2015).

Extensive research has been devoted in the last decades to solve the problem of VDB. However, existing solutions are theoretical and they are impractical for real-world applications (Chen et al., 2015; Li et al., 2006). The inefficiency originated from the costly computations required by existing solutions making it impractical for resource-constraint clients (Li et al., 2006; Zhu et al., 2013). Furthermore, the complex computations result in a limited support for sophisticated query types such as aggregated queries (Thompson et al., 2009; Wang et al., 2015).

1.3 Problem Definition

Most of the current solutions in the literature delegate the responsibility of the verification process to database users. In this scenario, when database users receive the result set of any query they send to the database service provider servers, they must perform the expensive verification process, which may include a large number of decryption and hashing operations, for each record in the result set.

Delegation of the verification process to database users suffers from four major limitations. First, these approaches are unpractical when it comes to sophisticated queries such as aggregated queries. This limitation is resulted from both the delegation of the expensive verification process to database users and the complex authentication structure used to authenticate data. Second, these approaches cannot be used in resource-constrained devices as the verification process is (computationally) expensive. Third, since database users only verify the data that they query, this poses a security concern where any manipulated unused data may go undetected for a long time. Finally, due to the weak separation of concerns in the design of the verification process , maintainability and extensibility become an issue.

1.4 Research Questions

In our research, we aim to answer the following questions:

- How can we ideally decouple database users from the verification process in order to provide a well structured separation of concerns design that would help overcome maintainability and extensibility limitations as well as would provide support for sophisticated query types?
- Since the computationally expensive verification process cannot be performed all the time practically, is it feasible to use periodic randomized verifications to eliminate the expensive verification process while maintaining a sufficient security levels?
- Can we incorporate game theory to effectively optimize the verification process results?

- Can we improve the performance further by tackling the methods of creating the authentication structures and the performing the verification process?

1.5 Contributions

In this research, we aim to address the above-mentioned limitations associated with existing solutions. Our work models the VDB security problem as a game theoretic problem using Stackelberg game. Mainly, our Stackelberg security game is a leader-follower game played between the verifier (leader), who is a trusted party to the database owner and the database service provider (follower). The new model includes the following entities: the database owner, database users, the database service provider, and the verifier. The database owner is responsible for creating the database special authentication structure, which is stored along with the database at the database service provider servers. The database service provider is responsible for hosting the database on the cloud and providing access to the authorized database users. In addition, some of the processing operations required for this model are performed at the database service provider side. Finally, the verifier is a software component running scheduled verifications on-behalf of the database owner. Our model provides both data integrity and completeness assurance. Furthermore, database users, who might be resource-constraint clients, are not involved in the verification process. As a result, the proposed model supports all query types and is easy to implement and integrate into existing systems. The contribution this work is:

(1) Employing Stackelberg game theory to schedule periodic verifications to be run by the verifier.

(2) Delegating the task of the verification process to a new entity called the verifier, who is a trusted party to the database owner and runs scheduled verifications on-behalf of the database owner, instead of database users.

(3) Introducing two stage verification process, namely the overall-table verification and the in-depth table verification processes, to reduce computation and communication overhead.

1.6 Thesis Organization

The rest of this thesis is organized as follows: In Chapter 2, we introduce the background and discuss the related work. In Chapter 3, we present a game theoretic analysis and design of the proposed model. In Chapter 4, we cover the implementation settings and we evaluate the proposed model. Finally, Chapter 5 concludes the work by outlining the main research outcomes and defining future work paths.

Chapter 2

Background and Related Work

This chapter is composed of two main sections. The first section introduces the preliminary concepts involved in the VDB model as well as game theory and Stackelberg game theory. The second section covers the VDB and the use Stackelberg game in security state of the art research in the literature.

2.1 Background

In this section, we define the preliminary building blocks of the verifiable database model including one-way cryptographic hash functions, public-key digital signatures, aggregated digital signatures, tow-party and three-party authentication model, Merkle hash tree, data correctness, and data completeness. In addition, we present an overview of game theory and Stackelberg game theory.

2.1.1 One-Way Cryptographic Hash Functions

One-way hash function $h(m)$ is a function that works in one direction by taking a message m of any length and producing its unique fixed-length hash digest as shown in Figure 2.1. A secure hash function must satisfy the following three conditions:

(1) Preimage-resistance which implies that given a hash digest h, it's computationally infeasible to find any message m such that $h = h(m)$. In other words, given a hash digest, it's computationally infeasible to find the original message that produced the hash digest (Rogaway &

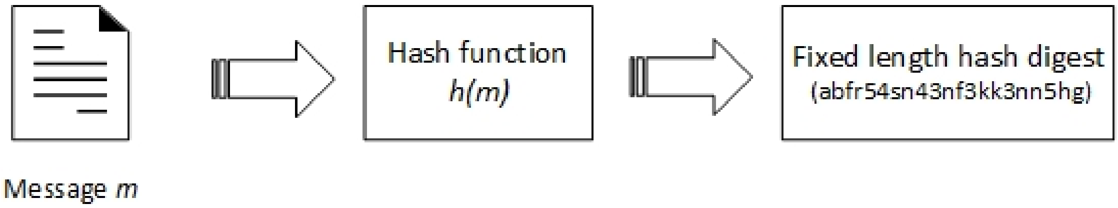


Figure 2.1: One-way hash function

Shrimpton, 2004).

(2) Second-preimage resistance which implies that given a message m and its hash digest $h(m)$, it's computationally infeasible to find a second message m' where $m \neq m'$ and $h(m) = h(m')$. In other words, it's infeasible to find a second message that produces the same hash digest as another different message (Rogaway & Shrimpton, 2004).

(3) Collision resistance which implies that it's computationally infeasible to find two different messages m and m' that produce the same hash digest (Rogaway & Shrimpton, 2004).

Hash functions are an important building block of verifiable databases because they help reduce the communication and computation overhead. For instance, instead of signing a database record of length 200 characters $(1600\ bits)$, we can hash this database record first and then we sign the obtained hash digest of 256 bits length. The secure hash algorithm (SHA) family is an example of hash functions (Pang, Zhang, & Mouratidis, 2009).

2.1.2 Public-Key Digital Signature Schemes

A public-key digital signature scheme is a mathematical scheme used to sign digital content in order to verify the authenticity, origin, and integrity of the signed message. In this context, the signer generates two keys, a public and a private key. The private key is kept secret and used to sign the message. On the other hand, the public key is distributed to clients who need to verify the signed message. RSA is an example of public-key digital signatures (Li et al., 2006; Pang et al., 2009; Pointcheval & Stern, 2000; Rivest, Shamir, & Adleman, 1978).

A secure digital signature scheme must provide the following properties:

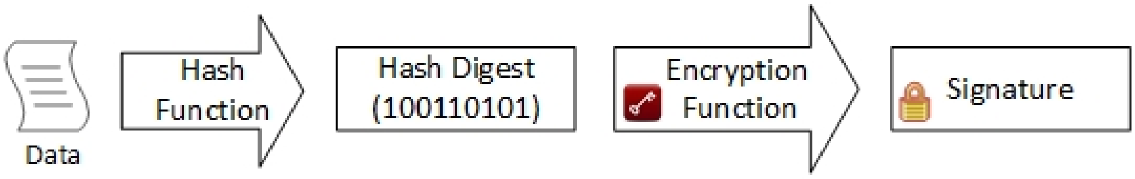


Figure 2.2: Digital signature signing process

(1) Authenticity which implies that the signer, who possesses the secret key, has actually signed the message (Pointcheval & Stern, 2000; Rivest et al., 1978).

(2) Unforgeability which implies that only the signer can generate a valid signature for the concerned message (Pointcheval & Stern, 2000; Rivest et al., 1978).

(3) Non-re-usability which implies that a signature of one message can not be used to sign a another message (Pointcheval & Stern, 2000; Rivest et al., 1978).

(4) Non-repudiation which implies that the signer can not deny the fact that a valid signed message was in fact signed by him (Pointcheval & Stern, 2000; Rivest et al., 1978).

(5) Integrity which ensures that the content has not been modified after it was signed (Pointcheval & Stern, 2000; Rivest et al., 1978).

Practically, digital signing is achieved using cryptographic encryption as well as cryptographic hash functions. There are two phases involved which are the signing phase and the verification phase. To illustrate the process, assume that we have data to be signed and then verified. To sign the data, the first step is to hash the data using any secure cryptographic hash function such as SHA. The result of the hashing process is the hash digest. Next, the signer uses his private key to encrypt the obtained hash digest which produces a signature of the input data as shown in Figure 2.2.

To verify the integrity of signed data, the signer needs to distribute his public key to the party concerned with the verification process first. In addition, the verifier party needs the signature and the data itself to be able to complete the verification process as shown in Figure 2.3. First, the verifier has to hash the received data. Second, the verifier decrypt the signature of the data. Finally the verifier compares the two hash digests and detect any data integrity if they are not equal.

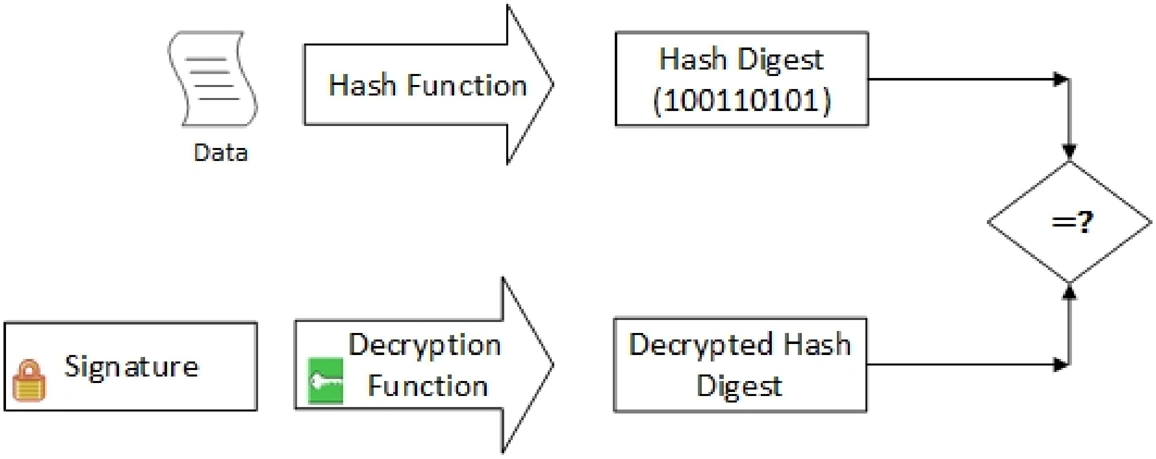


Figure 2.3: Digital signature verification process

2.1.3 Aggregated Digital Signature

The aggregated digital signature scheme aggregates multiple digital signatures into one signature to reduce communication and computation overhead. The resulted aggregated signature can be verified once instead of multiple verifications (Ma, Deng, Pang, & Zhou, 2005). This feature can save a great deal of bandwidth and computations overhead specially when dealing with very large databases. For instance, instead of sending signatures of 100 records from a database table, these 100 signatures can be aggregated into only one signature with a size equals to the size of one signature. Similarly, instead of verifying the 100 signature separately, the aggregated signature can be verified once and for all the 100 signatures together.

2.1.4 Two-Party and Three-Party Authentication Models

There are two parties involved in the two-party authentication model, a client and untrusted server. The client outsources his data management to the untrusted server. The server manages the data and provides the client with the ability to update and query the data. Considering that the server is untrusted, a mechanism to ensure that the data has not been manipulated has to be implemented (Goodrich et al., 2008; Narasimha & Tsudik, 2006; Papamanthou & Tamassia, 2007). Cryptographic primitives such as hash functions and digital signatures are used in this model to facilitate the integrity authentication process. This model provides private verifiability in which

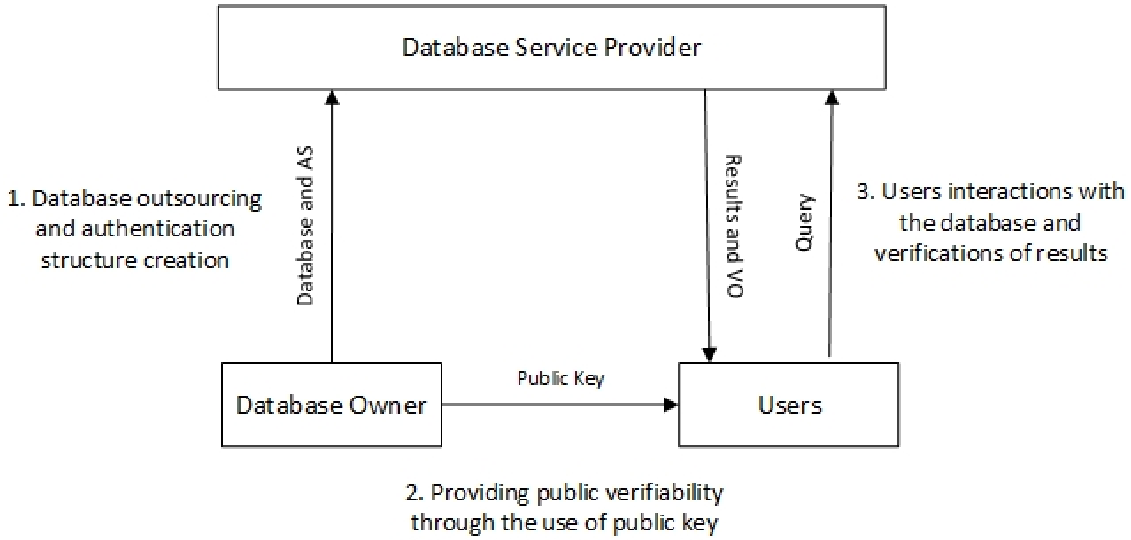

Database Service Provider
1. Database outsourcing and authentication structure creation
Database and AS
Results and VO
Query
3. Users interactions with the database and verifications of results
Database Owner
Public Key
Users
2. Providing public verifiability through the use of public key

Figure 2.4: Three-party authentication model

only the database owner can verify the data.

The two-party authentication model can be extended to the three-party model by adding public verifiability, which enables a third party to verify query results (Narasimha & Tsudik, 2006). The public verifiability is facilitated through the use of public-key digital signature in which the public key can be distributed to third parties to be used in the verification of the signature. Figure 2.4 illustrates the three-party authentication model.

Verifiable database is an integrity authentication model which uses the two-party or the three-party authentication models in order to provide integrity assurance of outsourced data. The entities involved in the verifiable database model are the database owner, the database service provider, and database users. Although in practice the database owner can be represented by one or more entities, for simplicity we will refer to him as a singular entity. The following definitions are important to understand the verifiable database model:

Definition 2.1.4.1. *Authentication Structure:* an authentication structure is a special type of data structures which is built based on a given data and used to authenticate the underlying data integrity. Authentication structures are created and signed by the database owner and uploaded along with data to the database service provider servers. Authentication structures use cryptographic primitives such as digital signatures to ensure the authenticity of data. Merkle hash tree is a well known type of authentication structures which will be described in details in the next subsection.

Definition 2.1.4.2. *Verification Process:* the verification process is an integrity verification procedure used by database users to verify the integrity of their query results. The integrity verification process includes data correctness and data completeness verification.

Definition 2.1.4.3. *Verification Object:* the verification object is an object extracted from the authentication structure of any given data and used to verify integrity of the queried part of that data. The verification object helps to reconstruct the authentication structure at the side of database users in order to verify the concerned data integrity.

2.1.5 Merkle Hash Tree (MHT)

Merkle hash tree, which was proposed by Merkle (1989), is a data structure used to authenticate data. MHT data structure is a type of binary tree built based on a given set of data and used to authenticate the underlying data. In MHT, each leaf node stores the hash of a data value of the underlying data to be authenticated while each intermediate node stores the hash of the concatenation of its two children nodes. Finally, the root node stores a digital signature of the hash of the concatenations of its two children. To verify the authenticity of data, the tree can be recalculated from the underlying data values recursively. The calculated root can be verified against the signed root to detect any integrity violations (Li et al., 2006; Narasimha & Tsudik, 2005; Pang et al., 2009; Wang et al., 2015). Figure 2.5 shows an example of MHT. In this example, given a dataset to be authenticated which consists of data elements $\{d1, d2, d3, d4\}$, we follow the following steps to create the MHT authentication structure:

- Use a secure cryptographic hash function such as SHA to create the leaf nodes of the tree. For instance, the leaf node $N1$ concerned with data element $d1$ stores the hash digest $h(d1)$.
- Recursively create intermediate nodes where each intermediate node stores a hash of concatenating the content of its two children nodes. For instance, the intermediate node $N5$ stores the hash of concatenating its two children nodes $h(N1|N2)$.
- When reaching the root node $N7$, hash the content of its two children nodes and sign it using a secure cryptographic public-key digital signature such as RSA. In the example, the

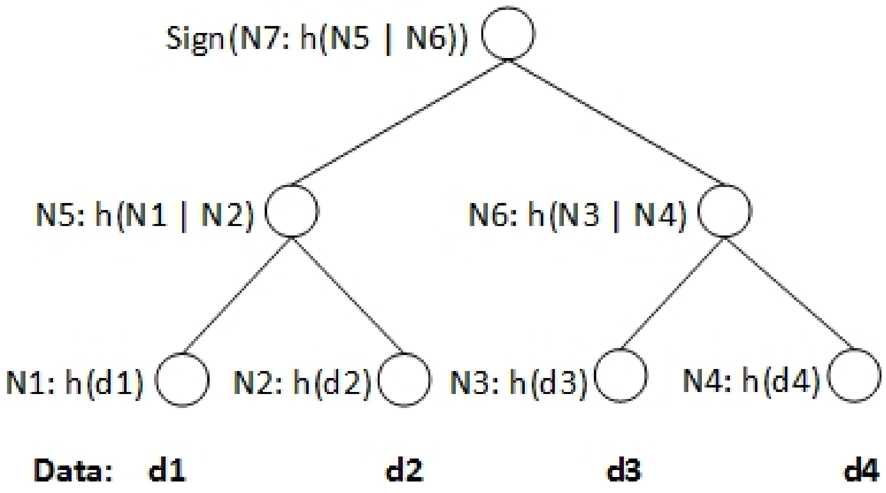


Figure 2.5: Example of MHT

root node $N7$ stores the signature of concatenating its two children nodes' content $N7 = Sign(h(N5|N6))$.

To authenticate data before consuming it, suppose a user queries the data and receives the data element $d1$ as his query result set. To be able to reconstruct the tree, the user needs besides the query result set a verification object including the nodes $(N2, N6, N7)$. Next, the user can follow the following steps to authenticate his query result before consuming it:

- Hash the query result $d1$ to create the leaf node $N1 = h(d1)$.
- Using the obtained leaf node $N1$ and the leaf node $N2$ from the verification object, create the intermediate node $N5 = h(N1|N2)$.
- Using the obtained intermediate node $N5$ and the intermediate node $N6$ from the verification object, create the root node $N7' = h(N5|N6)$.
- Decrypt the node $N7$ from the verification object and verify that $Decrypt(N7) = N7'$

2.1.6 Data Correctness and Data Completeness

The major security requirement of verifiable databases is to provide data correctness assurance. Data correctness ensures the integrity and authenticity of the outsourced database. This means that when interacting with the data, a query result set must be proved authentic with respect to the database owner. This proof implies that the data has not been tampered with in any an unauthorized way (Narasimha & Tsudik, 2005; Wang et al., 2015; Yuan & Yu, 2013).

Another security requirement of verifiable databases is to provide data completeness assurance. Data completeness ensures that a query result set is complete and that the database service provider has not deleted any data from that database and has not omitted any data from the result set in any an unauthorized way (Narasimha & Tsudik, 2005; Wang et al., 2015; Yuan & Yu, 2013).

It's worth mentioning that providing data correctness does not necessarily guarantee data completeness. This is because the verifier uses the data itself as well as data from the verification object in order to be able to perform the verification process. As a result, when deleting data in an unauthorized way from the database permanently or from the result set, the verifier can only verify the integrity of the present data in the result set. Therefore, there has to be a mechanism to detect any an unauthorized data deletion. These two security requirements must be analyzed and addressed independently when designing a verifiable database model.

2.1.7 Game Theory and Stackelberg Game Theory

A game is any situation where individuals (players) interact and have a set of choices to choose from. Thus, each player has partial control over the game through their actions (M'hamdi & Bentahar, 2012). According to Osborne and Rubinstein (1994), ”Game theory is a bag of analytical tools designed to help us understand the phenomena that we observe when decision-makers interact. The basic assumptions that underlie the theory are that decision-makers pursue well-defined exogenous objectives (they are rational) and take into account their knowledge or expectations of other decision-makers' behavior (they reason strategically)” (p. 1).

In general, a game consists of the following three main elements (Osborne & Rubinstein, 1994):

(1) A set of P players where each player is referred to as p_i.

(2) For every player p_i, a set s_i of strategies. Thus a game consists of a set S of outcomes where $S = [s_1 \times s_2 \times s_3 ... \times s_n]$.

(3) For every player p_i, a utility function U_i which determines player's p_i payoff and his preference over the set of strategies S.

The following are a general concepts pertain to game theory:

Definition 2.1.7.1. *Strategy:* a strategy or a Pure Strategy is a set of actions where each action is a response for a certain situation in a given game.

Definition 2.1.7.2. *Mixed Strategy:* a mixed strategy is a strategy that consists of a probability distribution over the actions of a pure strategy which make it harder for the opponent to predict.

Definition 2.1.7.3. *Nash Equilibrium:* nash equilibrium represents players' strategies that are mutually best responses. No player would deviate from his choice even after learning about the other player's strategy.

Game theory classifies types of games into a number of classes based on different aspects. For the purpose of this research, we are concerned with a class of games called Stackelberg games. Stackelberg model was developed by Heinrich von Stackelberg in 1934 in the book titled ”Market Structure and Equilibrium” (Von Stackelberg, 1934). When first appeared, it was purely an economics model to represent the market competition between firms. The basic idea behind Stackelberg game theory is that a competition between firms can be characterized based on the fact that certain firms has more control over the market while other firms has less control. The former is called a leader as a result of being dominant in certain characteristics such as the size and reputation. The latter is called a follower which means that it has a limited influence on the concerned market. Stackelberg game model analyzes and represents this phenomena and grants to the leader firm an optimal strategy concerned with taking certain decision related to the market with respect to the follower firm expected responses.

In security, Stackelberg game is used to model attacker-defender security problems (Korzhyk, Conitzer, & Parr, 2010; Wahab, Bentahar, Otrok, & Mourad, 2016). In Stackelberg game, a leader commits to a mixed strategy first, and then a follower optimizes his strategy in order to maximize his reward (An, Tambe, Ordonez, Shieh, & Kiekintveld, 2011; Korzhyk et al., 2010; M'hamdi & Bentahar, 2012; Paruchuri et al., 2008b). Although the leader has to move first and the follower gets to observe the leader strategy before choosing his strategy, the leader has the first mover advantage, which guarantees a higher payoff for the leader (Wahab et al., 2016). To illustrate the benefit of being a leader in a Stackelberg game, let us analyze a simple game with a payoff table as shown in Table 2.1. This example game is a general example that has been used in a wide variety of resources including (Korzhyk et al., 2010; M'hamdi & Bentahar, 2012; Paruchuri et al., 2008b; Pita et al., 2008). The row player is the leader and the column player is the follower. There exists only one pure-strategy Nash equilibrium when the leader plays strategy A and the follower plays strategy C. This gives the leader a payoff of 3. Although playing strategy B is strictly dominated for the leader, the leader can receive a payoff of 4 if he can commit to playing strategy B before the follower chooses. This is because the follower will always play strategy D to obtain a higher payoff. However, if the leader can commit to a mixed strategy of playing A with (0.5) probability and playing B with (0.5) probability, then the follower will always play strategy D which gives the leader a payoff of 4.5 (Yin, Korzhyk, Kiekintveld, Conitzer, & Tambe, 2010).

Table 2.1: Example of a Payoff table

	C	D
A	(3 , 2)	(5 , 1)
B	(2 , 1)	(4 , 3)

Game theory uses mathematical representations to describe and solve games precisely (Osborne & Rubinstein, 1994). For the course of our specific research problem, we will deal with representing our game mathematically and solving it using linear programming techniques, which is defined as a mathematical optimization method used to maximize or minimize a given function called the objective function with respect to specific constraints and condition (M'hamdi & Bentahar, 2012).

2.2 Related Work

In this section, we survey the state of the art research related to the verifiable database approaches and we highlight the limitations associated with each approach. We conclude this section by reviewing the use of Stackelberg game theory in security in the literature.

2.2.1 Verifiable Database

There has been an ongoing substantial work on the verifiable database three-party model. In this model, a database owner outsources his data management to an untrusted third-party service provider on the cloud, who answers queries sent from database users on-behalf of the database owner. Since the database is stored at an untrusted third-party, this model has to ensure detecting any data integrity violation with outstanding probability (Goodrich et al., 2008; Papamanthou & Tamassia, 2007; Wang et al., 2015). The current approaches to solve this research problem can be categorized into three different classes:

Tree-based approaches: one popular tree-based approach is the Merkle hash tree (MHT) introduced by Merkle (1989). In MHT, each leaf node contains a hash of the underlying data whereas each intermediate node stores the hash of its two children nodes. Finally, the root node stores the signed hash of its two children nodes. When verifying, the tree can be reconstructed bottom up from the underlying data so that the reconstructed root can be verified against the stored signed root. The first work that addressed the problem of verifiable database using MHT was proposed by Devanbu, Gertz, Martel, and Stubblebine (2002). The MHT in this case is constructed over each entire database table where each tuple represents a leaf node in the tree. When querying the data, the database service provider responds to database users queries with the result set as well as a VO containing the hash of all tuples in the table. The obvious drawback of this approach is the extremely inefficient communications and computations overhead. Ma et al. (2005) Has developed a more efficient model called the attr-MHT by constructing MHT on individual tuples instead of the whole table to reduce communication and processing overhead. In the attr-MHT, each attribute of a tuple represents a leaf node in MHT and each signature corresponds to one tuple. When querying the data, the database service provider respond to database uers' queries with the result set along with a VO containing the tree nodes of the attributes which is not included in the query result set. Other remarkable work that reduced computation overhead used the sophisticated Merkle B-tree and the Embedded Merkle B-tree (Li et al., 2006). The drawbacks of using the tree-based approaches are the associated computation and communication overhead requirements (Wang et al., 2015). Furthermore, these approaches have a limited support for sophisticated query types such as aggregated queries.

Probabilistic integrity verification approach: the idea behind this approach is that the database owner inserts fake tuples into the database in different locations. Those tuples are used to verify the integrity of data with certain probability (Xie et al., 2007). When querying the data, the database service provider responds to database users' queries with the result set. Database users then can check for the fake tuples inserted that satisfies the query. If they find the expected fake tuples, database users then can draw a certain probability that the data integrity has or has not been manipulated. The drawbacks of this approach is that it requires storage overhead to store the fake tuples locally and distribute it to all database users in order to be used later on in the verification process (Wang et al., 2015). In addition, the number of fake tuples required to guarantee a certain probability assurance are linear to the size of the database. This renders this approach completely inefficient for large-scale databases. Furthermore, the process of maintaining and identifying the fake tuples can be extremely costly process from a computational stand point. Finally, we argue that the database service provider is rational and he might has an interest in tuples that reflect a real entity. Thus, there is a high chance that the fake tuples might not be of any interest to the database service provider. As a result, a sensitive data integrity breach might go undetected while relying on this approach might still indicate a high probability of valid data integrity. Those concerns affect both the correctness and completeness of the underlying data.

Signature-based approach: in this approach, the database owner hashes and signs the concatenation of each tuple's data and stores it along with the tuple at the database service provider servers (Wang et al., 2015; Yang, Papadias, Papadopoulos, & Kalnis, 2009). Although this approach uses a much simpler authentication structure, the VO when querying a large database can become inefficient from bandwidth and computations prospective. The database service provider has to send along with every query a linear number n of signatures equals to the number of tuples in the results set. In addition, this approach does not provide data completeness assurance. Mykletun, Narasimha, and Tsudik (2006) have proposed an improved signature-based version using signature aggregation. When querying the database in this approach, the signatures of multiple tuples are aggregated into one signature to reduce communication and computation overhead (Pang et al., 2009). As a result, the database service provider responds to users query with the result set as well as one aggregated digital signature for the whole result set. The aggregated signature has a size equals to one single signature and can be verified only once for the whole result set instead of a verification per each tuple. The limitation of the signature aggregation is that although it provides data correctness assurance, it lacks the assurance of data completeness. Narasimha and Tsudik (2006) have proposed yet another improved version that uses signature aggregation and chaining to overcome the completeness assurance limitation. The idea behind this version is that in a sorted database table, each tuple is signed and its signature chained with the signature of its adjacent tuples. When querying, the database service provider responds to range queries by chaining the signature of the predecessor and successor tuples in the VO. This way, database users can verify that the result set is complete and that the server has not omitted any results that satisfy the query. The drawback of this approach is its complexity and inapplicability where for every tuple inserted, deleted, or updated, the database service provider has to calculate the affected tuples signature with regard to the signature of its adjacent tuples. In addition, data completeness assurance here is provided based on the assumption that the result set would contain some tuples in order to detect any omitted tuples. However, in practice, this might not be the case where the result set might be empty. Furthermore, since the verification process works by authenticating individual tuples based on their signatures chaining with the signatures of adjacent tuples, this approach is only useful when using range queries that retrieve all tuples in that range. Otherwise, every distinct tuples' signature must be accompanied with its adjacent tuples' signatures which can be an extremely expensive process specially when dealing with large databases.

Table 2.2: Summery of limitations of the current VDB approaches

VDB Approach	Computational Cost	Aggregated Queries	Security Level	Data Correctness	Data Completeness
Tree-based	High	Not supported	Average	Supported	Not supported
Probabilistic integrity verification˙	High	Supported	Low	Supported	Not supported
Signature-based	High	Not supported	Average	Supported	Supported

Most of the previous verifiable database approaches require that database users have to verify every single query in order to ensure data integrity. As a result, the verification process is highly expensive for computationally weak clients. Furthermore, since database users only verify the data that they are querying, we argue that this model presents a security concern where any integrity violation in data that is not being queried might go undetected for a long time. Table 2.2 summarizes the limitations of current VDB approaches. On the other hand, there is a recent work proposed by Zhu et al. (2013) that has molded the problem of outsourced storage in a way similar to our work. In their work, the authors have concluded that it is crucial to use periodic verifications in order to eliminate the processing and communication overhead incurred by current VDB models. Their model was designed to verify the integrity of memory blocks and provides general storage verifiability not specific to the problem of outsourced database. Moreover, unlike our approach that uses game theory to randomize verifications, their approach uses a random sampling audit, and we will show in Chapter 4 that our approach outperforms the uniform randomization.

2.2.2 Stackelberg Game Theory in Security

Game Theory has become a popular methodology for solving sophisticated security scheduling and patrolling problems (An et al., 2011; Wahab et al., 2016). Stackelberg game is used to model attacker-defender security problems (Korzhyk et al., 2010; Wahab et al., 2016).

Pita et al. (2008) have used Stackelberg game theory to model the security problem police officers face at Los Angeles International Airport (LAX)(Paruchuri et al., 2008b; Wahab et al., 2016). In these settings, the police have a number of routes (targets), which lead to different sensitive locations, and a limited number of resources (checkpoints and canine unites) to cover those routes. Taking into consideration this limitation of the number of resources available and the uncertainty about the attacker types faced as well as the observability of the police strategy, authors have modeled the security problem at LAX as a Bayesian Stackelberg security game to randomize the assignment of resources to effectively cover targets.

In M’hamdi and Bentahar (2012) work, the authors have used Stackelberg game theory to schedule reputation maintenance in agent-based communities. In these settings, an agent-based community consists of rational community members who have the incentive to interact and provide services for each other, for other communities, or for end users. In addition, for each community, there exists a controller agent, who is responsible for maintaining the community reputation to attract other agents to the community. The controller agent has to perform a reputation maintenance check randomly to ensure that community members are acting truthfully. After each maintenance check, the community member reputation is updated and might receive a reward or a punishment. The problem of this model is that it is not feasible to perform the maintenance checks at all times. In addition, there are uncertainty about the community member faced. Authors have modeled this security scheduling problem as Stackelberg security game to effectively and efficiently schedule the reputation maintenance.

Chapter 3

Game Theocratic Model Design[1]

This Chapter introduces problem definition and the proposed model design first, which is based on Stackelberg game theory. Second, we will present the game theoretic analysis of the proposed solution. Finally, we will discuss how to find the optimal mixed strategy of the verifier.

3.1 Problem Definition

As mentioned in Chapter 2, most of current verifiable database model problems are a result of mixing the concerns between the entities involved in the model. In current models in the literature, after the database owner creates and uploads the authentication structure of the database, database users are responsible for carrying out the verification process themselves. This mixing of concerns adds the following complexities to the verifiable database model:

(1) Resource-constraint database users might not be capable of performing the expensive verification processes which may include a large number of sophisticated decryption processes.

(2) The sophisticated authentication structures associated with the verifiable database model affect the practicality when used with conjunction with complex database users' queries such as the aggregated queries. In practice, most of the current verifiable database models do not support such types of queries.

[1] the content of this chapter has been accepted for publication in the proceedings of The 32nd ACM Symposium on Applied Computing Conference

(3) Form extensibility and maintainability prospectives, this mixing of concerns adds up to the complexity of the underlying systems and makes them much harder to extend and maintain.

(4) The presence of this mixing of concerns limits the choices of the authentication structures applicable in order to provide an adequate level of security assurance. For instance, to provide data completeness assurance, current verifiable database models has to use sophisticated authentication structures.

(5) From a security prospective, database users only verify the data that they query. This means if any unused data was manipulated, this manipulation can go undetected for a long time which makes it harder to roll back to the authentic version of data.

In our design, we aim to mainly address the mixing of concerns by introducing a logical separation of concerns. To meet this requirement, the proposed model revokes the responsibility of the verification process from database users and assigns it to an independent entity called the verifier as will be described in the next section. The verifier, who is a trusted party to the database owner, will be responsible for carrying out a periodic verifications to insure the integrity of data. Furthermore, the interaction between the verifier and the database service provider will be analyzed and modeled using Stackelberg game theory and solved mathematically using linear programming.

3.2 Model Design

The proposed model is designed as Stackelberg security game. The decision to model the VDB problem as Stackelberg game is based on the wide use of this type of game to model attacker-defender security problems. For instance, it's been proven in (Pita et al., 2008) that modeling security problems as a Stackelberg game guarantees an optimal patrolling strategy to the defender which translates to an optimal payoff. This attacker-defender scenario is perfectly applicable to the VDB model where the verifier schedules periodic verifications to detect any violations from the database service provider. Our model includes the following four entities: the database owner, database users, the database service provider, and the verifier. The database owner is responsible for creating the database special authentication structure, which is stored along with the database at

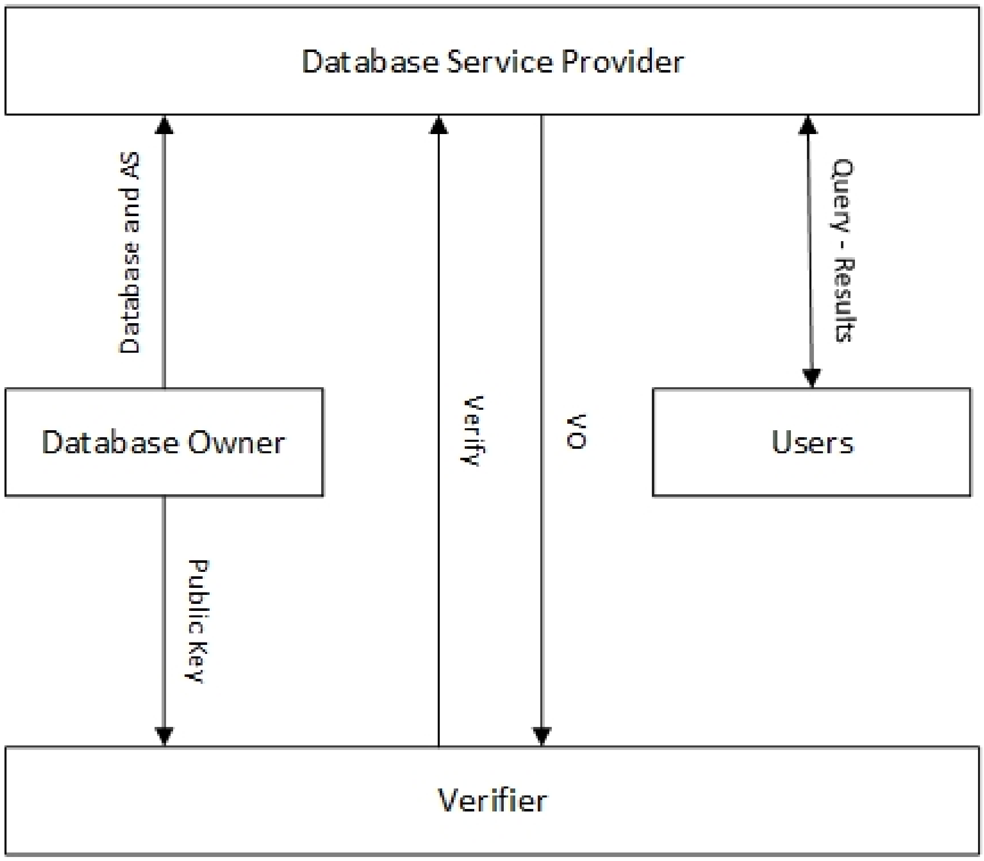


Figure 3.1: Proposed VDB model using Stackelberg game theory

the database service provider servers. The database service provider is responsible for hosting the database and providing access to the authorized database users. In addition, some of the processing required for this model is performed at the database service provider side. Finally, the verifier is a software component running scheduled verifications on-behalf of the database owner as shown in Figure 3.1. The game is mainly played between the verifier (leader) and the database service provider (follower). First, the verifier runs Stackelberg solver algorithm to find the optimal mixed strategy to commit to. Among the two most efficient Stackelberg solver algorithms existent to date, we have chosen DOBSS (Paruchuri et al., 2008a) as it was proven in (Paruchuri et al., 2008a, 2008b) to be superior to its competitive ASAP. The mixed strategy in the proposed model includes for every table in the database its probability of being verified. Experimental results showed that DOBSS was significantly faster than ASAP in solving complex problems. Next, the verifier runs a uniform randomization method in order to randomize n number of time slots. Finally, for every

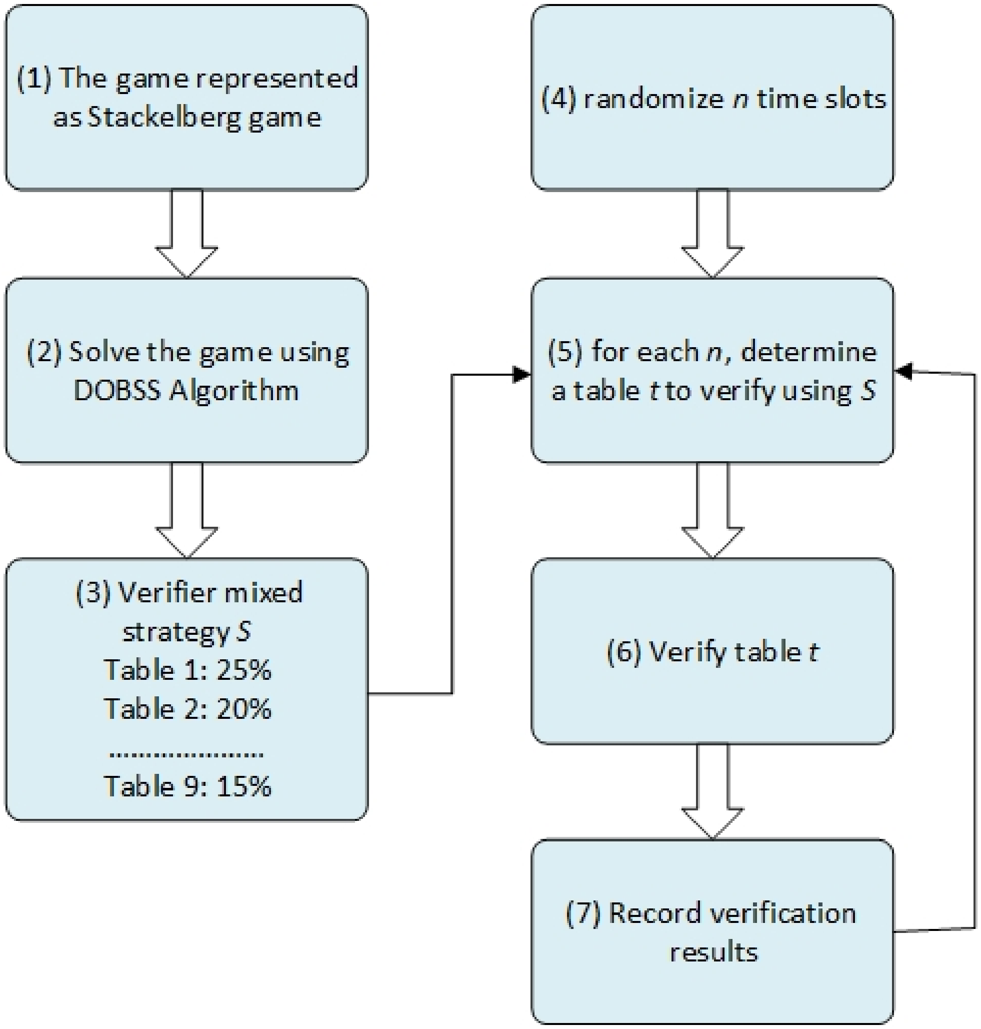

(1) The game represented as Stackelberg game
(2) Solve the game using DOBSS Algorithm
(3) Verifier mixed strategy *S*
Table 1: 25%
Table 2: 20%
....
Table 9: 15%
(4) randomize *n* time slots
(5) for each *n*, determine a table *t* to verify using *S*
(6) Verify table *t*
(7) Record verification results

Figure 3.2: Flowchart of the verifier scheduling problem

random time slot, the verifier uses the previously obtained mixed strategy to determine which table to verify at that particular time. Figure 3.2 shows the flowchart of steps involved to solve the verifier scheduling problem.

3.3 Game Theoretic Analysis

As mentioned previously, the game is played between the verifier (leader) and the database service provider (follower). In order to find the optimal mixed strategy of the verifier to commit to, the first step is to analyze the game and construct its payoff matrix. The payoff matrix determines for each player in the game the reward (r) / penalty (p) resulted from players' actions. In addition, the proposed model ranks each table based on the tables' data sensitivity (tds). This rank is realistic since different tables have different data sensitivity levels. For instance, a table that stores credit cards information has more value than a table that stores country names. Thus, sensitive data tables should be verified more often than tables having less data sensitivity rank. We use (tds) to construct the a logical payoff matrix of the game where every player's reward/penalty is proportional to the table being verified/manipulated. For instance, if the database service provider manipulates a table with highly sensitive data, his reward/penalty will be high. It's worth mentioning that the payoff formulation we used to characterize the VDB model is driven by our game-theoretic analysis. The following are the possible outcomes of the game:

- **Data manipulation detected**: this is the case where the verifier verifies certain table t_i and detects data integrity violation. As a result, the verifier receives a payoff of ($r \times tds_i$), which is a reward that is proportional to the table data sensitivity. On the other hand, the database service provider receives a negative payoff of ($-p \times tds_i$).

- **Data manipulation undetected**: this is the case where the database service provider manipulates data in a certain table t_i and the verifier either verifies another table or does not perform verification at all. As a result, the verifier receives a penalty of ($-p \times tds_i$) while the database service provider receives a reward of ($r \times tds_i$).

- **Successful verification**: this is the case where the verifier performs verification while there was no data manipulation performed by the database service provider. As a result, the verifier receives a payoff of small negative negligible value c, which is the computational power consumed to perform the verification. On the other hand, the database service provider receives a payoff of 0.

- **Neutral situation**: in this case, the verifier does not perform verification and there was no data manipulation performed by the database service provider. As a result, both players of the game receive a payoff of 0.

Table 3.1 shows the strategy profile and payoff matrix of the game for a certain table t_i. The row player is the verifier whereas the column player is the database service provider.

Table 3.1: Strategy profile and payoff matrix of the game

	Manipulate	Not
Verify	$((r \times tds_i), (-p \times tds_i))$	$(-c, 0)$
Not	$((-p \times tds_i), (r \times tds_i))$	(0, 0)

Table 3.2 presents an example of a game. For simplicity, we assume that only three tables exist in the database. The row player is the verifier while the column player is the database service provider. For every table in the database, the solution suggests the optimal mixed strategy S for the verifier to commit to taking into account the optimal strategy Q of the database service provider.

Table 3.2: Example of a game

	Table 1	Table 2	Table 3	S
Table 1	(4.5 , -4.5)	–	–	45%
Table 2	–	(4.5, -4.5)	–	45%
Table 3	–	–	(0.4, -0.4)	10%
Q	1	1	0	

3.4 Finding the Verifier Optimal Mixed Strategy

To solve the problem mathematically, we will use the Stackelberg solver algorithm DOBSS (Refer to Paruchuri et al. (2008b) work for more information about DOBSS solver algorithm). The general methodology in this Chapter was inspired by the methodology used by Paruchuri et al. (2008b) and by Pita et al. (2008). However, our work uses its independent analysis and design special to the verifiable outsourced database model. We will start by defining the game as a mixed-integer quadratic program (MIQP). Then, we will transform the MIQP into a Mixed-Integer Linear Program (MILP) equivalent. Finally, we will solve the MILP problem using a linear programming solver tool.

Given a time interval, there exists a number of discrete moments in which the verifier has to play. In each individual moment, the verifier can either choose to verify (strategy 1) certain table or not (strategy 0). Similarly, there exists a discrete number of moments where the database service provider has to play. The database service provider can choose to either act maliciously (strategy 1) or act truthfully (strategy 0).

We denote by T and TDS the index set of database tables' names and its corresponding data sensitivity respectively. Thus, t_{i} and tds_{i} are the individual table $table_{\mathrm{i}}$ and its corresponding data sensitivity rank respectively. We also denote by Q the index set of the database service provider pure strategies. q_{j} is the strategy chosen by the database service provider for the table j, which represents either the database service provider manipulates table j or not. Thus, $q_{\mathrm{j}} \in \{0, 1\}$. We denote by S the verifier policy (mixed strategy), which consists of a vector of probability distribution over the set of tables. s_{i} is the probability of verifying table i. Thus, $s_{\mathrm{i}} \in (0 \ldots 1]$. We also denote by R and C the payoff matrices of the verifier and the database service provider respectively. R_{ij} and C_{ij} are the verifier and database service provider reward respectively when the verifier plays strategy s_{i} and the database service provider plays strategy q_{j}.

To obtain the optimal mixed strategy, the verifier has to use the backward induction to optimize his strategy based on the database service provider optimal strategy. Consequently, by fixing s_{i}, the database service provider has to solve the following problem to optimize his strategy:

$$\begin{aligned} \max_{Q} \quad & \sum_{j\in T}\sum_{i\in T} C_{ij}\,(1-s_i)\,q_j \\ \text{s.t.} \quad & \sum_{j\in T} q_j < |T| \\ & q_j \in \{0,1\} \qquad\qquad \forall j \in T \end{aligned} \tag{1}$$

Since the Optimization Problem 1 is a maximization problem and since s_i is fixed, q_j should be set as follows:

$$\sum_{i\in T} C_{ij}\,(1-s_i) \le 0 \rightarrow\ q_j = 0 \quad \forall j \in T \tag{2}$$

$$\sum_{i\in T} C_{ij}\,(1-s_i) \ge 0 \rightarrow\ q_j = 1 \quad \forall j \in T \tag{3}$$

Form equation 2 and equation 3, we can conclude that q_j has to satisfy the following two equations:

$$(1-\ q_j)\sum_{i\in T} C_{ij}\ (1-s_i) \leq 0 \quad \forall j \in T \tag{4}$$

$$q_j \sum_{i\in T} C_{ij}\ (1-s_i) \geq 0 \quad \forall j \in T \tag{5}$$

By embedding Equations 4 and 5 as constraints in the verifier optimization problem, the verifier has to solve the following problem in order to obtain the optimal mixed strategy:

$$\begin{aligned}
\max_{S,Q} \quad & \sum_{i\in T}\sum_{j\in T} R_{ij}\ s_i\ q_j \\
\text{s.t.} \quad & \sum_{i\in T} s_i = 1 \\
& \sum_{j\in T} q_j < |T| \\
& (1-\ q_j)\sum_{i\in T} C_{ij}\ (1-s_i) \leq 0 \quad && \forall j \in T \\
& q_j \sum_{i\in T} C_{ij}\ (1-s_i) \geq 0 && \forall j \in T \\
& s_i \in (0\ldots 1] && \forall i \in T \\
& q_j \in \{0,1\} && \forall j \in T
\end{aligned} \tag{6}$$

The objective function generates the optimal mixed strategy S that maximizes the verifier total payoff. The first and sixth constraints limit the mixed strategy to be a probability distribution over the set of tables. The second and seventh constraints ensure that the follower strategy is a pure strategy. The third and fourth constraints are used to consider the follower optimal strategy.

The next step is to linearize the MIQP obtained in Optimization Problem 6 by replacing the variables $Z_{\mathrm{ij}} = s_{\mathrm{i}} \times q_{\mathrm{j}}$ which will generate the following DOBSS equivalent MILP that can be solved using a linear programming solver tool:

$$\begin{aligned}
\max_{S,Q} \quad & \sum_{i\in T}\sum_{j\in T} R_{ij}\, Z_{ij} & \\
\text{s.t.} \quad & \sum_{i\in T} Z_{ij} = 1 & \forall j \in T \\
& \sum_{j\in T} Z_{ij} < |T| & \forall i \in T \\
& (1 -\ q_j) \sum_{i\in T} C_{ij}\, (1 - Z_{ij}) \le 0 & \forall j \in T \\
& q_j \sum_{i\in T} C_{ij}\, (1 - Z_{ij}) \ge 0 & \forall j \in T \\
& Z_{ij} \in [0 \ldots 1] & \forall i, j \in T \\
& q_j \in \{0, 1\} & \forall j \in T
\end{aligned} \tag{7}$$

Chapter 4

Implementation and Evaluation

This Chapter covers the implementation settings and evaluation of the system. The first section introduces the overall system architecture, and main interface functionalities. In addition, the first section shows how the game matrix is constructed and solved in order to obtain the mixed strategy and defines the transformation from a mixed strategy to verification schedule as well as covering different aspects of the verification process. In the second section, we evaluate the security, Stackelberg game scheduling performance, and the efficiency of the proposed model.

4.1 Implementation

In this section, we start by introducing the implemented system architecture and design. We conclude this section by presenting the different implementation settings of the main components of the implemented system.

4.1.1 System Architecture

The implemented system consists of two main components, the database owner component and the verifier component. The verifier component is of more importance in the course of this research because it involves the Stackelberg game. There is no direct interaction between these two components except when initially delivering the public key from the database owner to the verifier to be used in the verification process. In addition, when the verifier component detects data integrity violation, it may report the incident to the database owner automatically if required.

The database owner component is responsible for creating and uploading the authentication structure to the database service provider which is integrated to the business logic of the actual system. Once the initial database and its authentication structure are uploaded to the database service provider servers, the database owner component updates the authentication structure associated with any inserted, updated, or deleted tuples and its correspondent database tables.

The verifier component consists of a front-end, a business logic, and a back-end. The front-end is designed to provide user interface for input and for tracking the current verification session status. The business logic provides the required computations to construct and solve the Stackelberg game as well as performing the verifications accordingly. The back-end is used to store the settings of the system and the history of sessions results.

Figure 4.1 illustrates the flow chart and steps involved in the verifier scheduling problem. The system user, who is responsible for administering the developed system, interacts with system interface to set the input which consists of the target database address, the Stackelberg game input, and the number of verification schedule time slots to be randomized by the system. Once the system users specifies the target database address to be verified, the system reads the schema of the target database and enables the system user to enter the Stackelberg game input which is a sensitivity value assigned to every database table in the target database that will be used to construct the game matrix. Next, the system calculates the optimal mixed strategy for the verifier to commit to. Fianlly, when the system user runs the session, at every randomized time slot, the system uses the mixed strategy to identify a random table to verify and proceeds with the verification process. The pseudo code shown in Algorithm 1 describes the steps involved to solve the verifier scheduling problem.

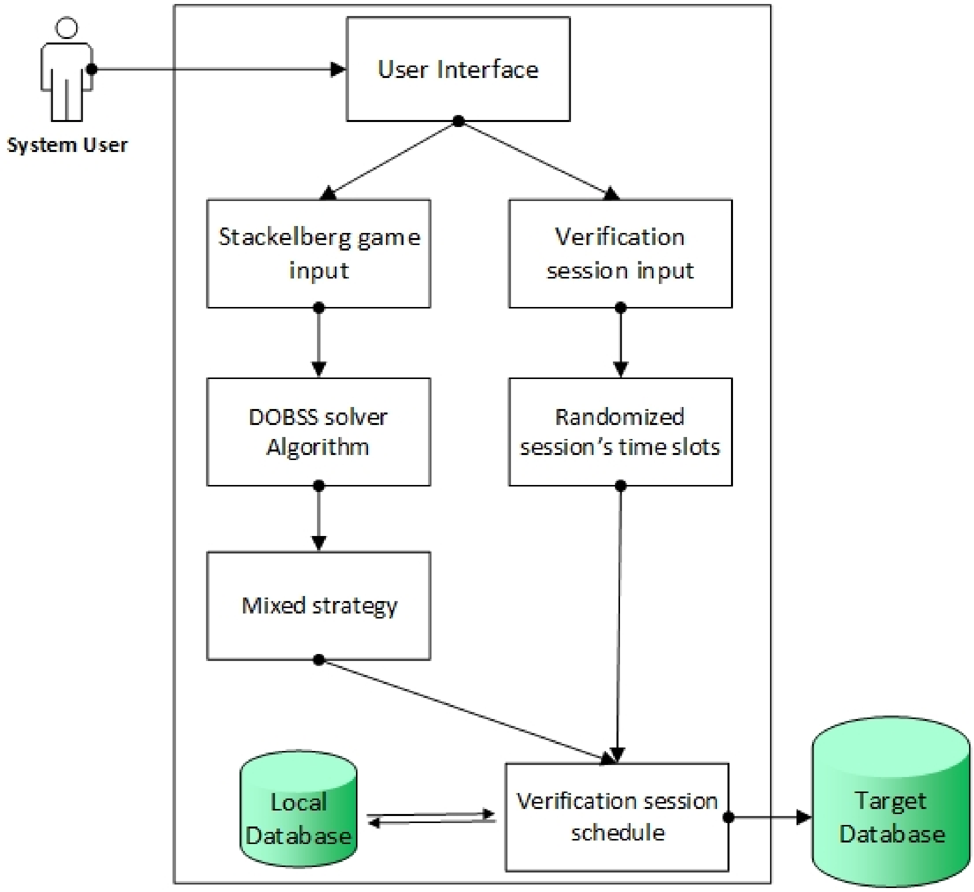


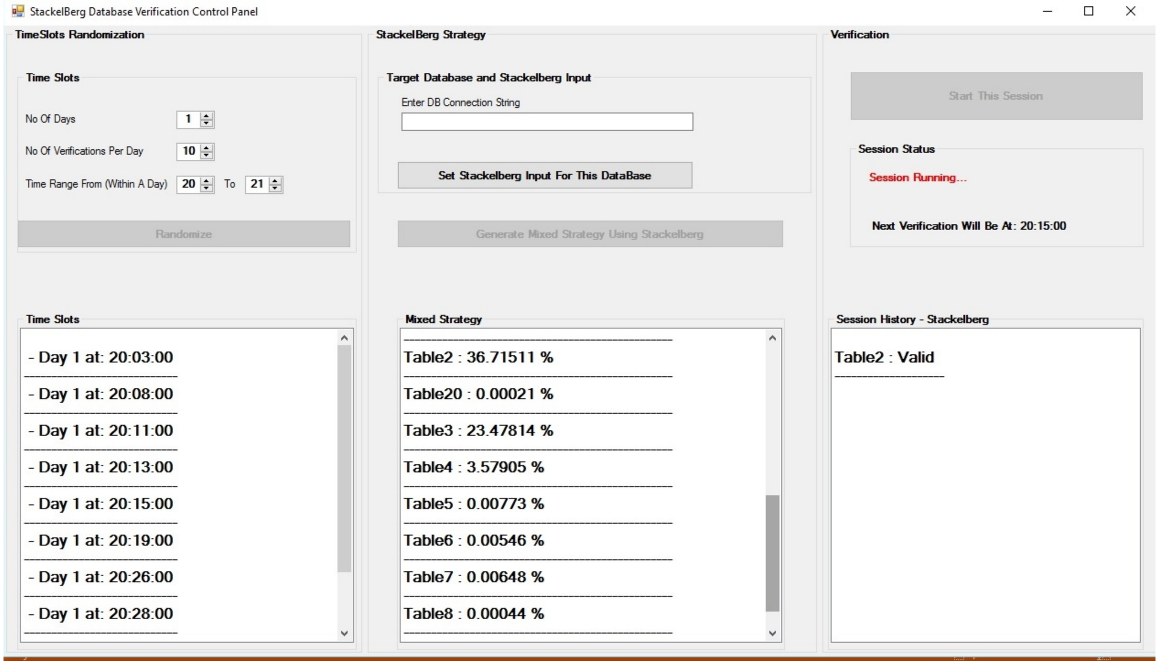
Figure 4.1: Dataflow and steps involved in the verifier component

Algorithm 1 Pseudo code of the verifier scheduling problem

```
Require: The game matrix, DOBSS representation, the number of time slots $n$
Find the verifier mixed strategy $S$ by solving the Optimization Problem 7
Create a vector $V$ of $n$ randomized time slots using uniform randomization
for $j = 0 \to (|\mathrm{V}| - 1)$ do
    For time slot $j$, find what table $t$ to verify using $S$
    Verify table $t$ using the overall-table verification process (Section 4.1.6)
    if ($t$ integrity is violated) then
        Identify individual affected records using the in-depth verification process (Section 4.1.6)
        Report the incident
    end if
end for
```

4.1.2 System Interface

The interface consists mainly of three columns as shown in Figure 4.2 The first column provides controls that allow the system user to specify a verification session of any number of days and a number of verifications time slots per day. Next, the system randomizes the verifications time slot for each day. The obtained session days and time slots are displayed in the list box underneath. The second column provides controls that allow the system user to specify the target database path in order to read the database schema. After reading the target database schema, the interface provides a pop-up window, which displays every table name in the database schema and enables the user to set the data sensitivity for each table. Finally, the interface enables the user to automatically construct and solve the Stackelberg game. The obtained optimal mixed strategy is displayed in the list box underneath. The third column is used to run and track the session as previously specified. The results of every verification is displayed in the list box underneath and stored in the back-end database.


StackelBerg Database Verification Control Panel
TimeSlots Randomization
StackelBerg Strategy
Verification
Time Slots
Target Database and Stackelberg Input
Enter DB Connection String
Set Stackelberg Input For This DataBase
Randomize
Generate Mixed Strategy Using Stackelberg
Start This Session
Session Status
Time Slots
Mixed Strategy
Session History - Stackelberg

No Of Days 1
No Of Verifications Per Day 10
Time Range From (Within A Day) 20 To 21

Session Running...
Next Verification Will Be At: 20:15:00

```
- Day 1 at: 20:03:00
- Day 1 at: 20:08:00
- Day 1 at: 20:11:00
- Day 1 at: 20:13:00
- Day 1 at: 20:15:00
- Day 1 at: 20:19:00
- Day 1 at: 20:26:00
- Day 1 at: 20:28:00
```

```
Table2 : 36.71511 %
Table20 : 0.00021 %
Table3 : 23.47814 %
Table4 : 3.57905 %
Table5 : 0.00773 %
Table6 : 0.00546 %
Table7 : 0.00648 %
Table8 : 0.00044 %
```

```
Table2 : Valid
```

Figure 4.2: System interface

4.1.3 Matrix Construction and Mixed Strategy Generation

Given the database schema and user input of every table data sensitivity, the system constructs the payoff matrix of the game. Next, this payoff matrix is used by DOBSS solver algorithm in order to calculate the verifier optimal mixed strategy. The obtained mixed strategy specifies for every table its probability of being verified by the verifier. The obtained mixed strategy grants the maximum payoff for the verifier. After obtaining the mixed strategy from the previous step, and at every verification time slot, the system sends the mixed strategy to a method that chooses one table randomly according to the specified mixed strategy. It is worth mentioning that the mixed strategy is generated only once. Then the verifier has to commit to that mixed strategy.

4.1.4 From Mixed Strategy to The Verification Schedule

The verification schedule maps every verification process to a certain table in the target database to be verified. Verification moments are chosen randomly using uniform randomization within the predefined period and the number of verification time slots specified by the system user. At every

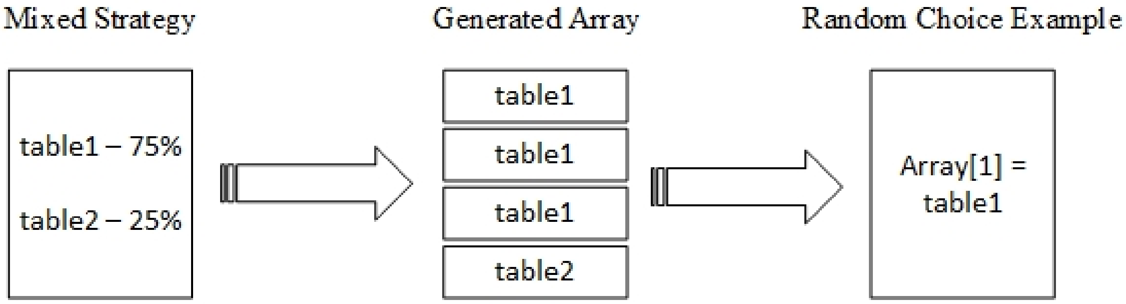


Figure 4.3: Mixed strategy to verification schedule

verification moment, the system uses the optimal mixed strategy in order to choose a table to verify with probability according to the mixed strategy. To illustrate the point, suppose that we have two balls in a dark box. One ball is red and the other is blue. If we grasp a ball at random, there is a 50% probability that we choose a ball with either color. Now what happens if we put three blue balls and one red ball in that box. When grasping a ball at random, there is a 75% probability of ending up with a blue ball and a 25% probability of ending up with a red ball. We have used the same concept when transforming from the mixed strategy to the verification schedule. According to the probability assigned to every table in the mixed strategy, we construct an array of the tables where every table is inserted a number of times proportional to its probability in the mixed strategy. For instance, if the obtained mixed strategy is verifying $table_1$ with probability of 75% and $table_2$ with probability of 25%, we insert into the array three instances of $table_1$ while only one instance of $table_2$ is inserted. Finally, a random index of the array is chosen and we end up with a table to verify chosen randomly with probability according to the mixed strategy as shown in Figure 4.3.

4.1.5 Authentication Structure Creation

Before the verification can be carried out, the database owner has to upload to the database service provider the authentication structure along with the database. The authentication structure in our case is signature based and it is of two types called table-based authentication structure and tuple-based authentication.

The table-based authentication structure is implemented by signing the hash of concatenation of each table data. The resulted table-based signature is saved in a special table dedicated to storing the names of the target database tables and its correspondent signatures. The verifier then can access this special table to fetch the signature associated with any table undergoing the verification process.

The tuple-based authentication is implemented by signing the hash of the concatenation of each tuple data. The resulted tuple-based authentication structure is stored inside the table of the concerned tuple. Every tuple signature is added to the concerned tuple in a new column dedicated for storing its signature.

Unlike the tuple-based authentication structure, which provides only integrity assurance, table-based authentication structure provides both integrity and completeness assurance. Moreover, using these two different signing processes saves a great deal of communications and computations overhead as will be illustrated in the next section.

4.1.6 The Verification Process

The verification process consists of two stages, the overall-table verification stage and the in-depth table verification stage. The former stage uses the table-based authentication structure whereas the latter stage uses the tuple-based authentication structure.

The overall-table verification stage is the initial verification process. Since it uses the table-based authentication structure, it involves only one communication to the database service provider server and one signature verification operation. Therefore, it's very efficient to employ as the first verification stage only to find out if any integrity violation (data correctness and data completeness) exists in the verified table. If there was no data integrity violation detected, there is no need to proceed to the comparably computationally expensive in-depth verification stage. This verification stage provides data correctness and data completeness assurance.

The in-depth table verification stage is the second verification stage. It's only activated when there is data integrity violation detected in a certain table from the initial overall-table verification stage. Since the in-depth table verification stage uses the tuple-based authentication structure, its use is to identify the specific individual affected tuples by the integrity violation.

Figure 4.4 shows the detailed flow chart diagram of the verifier component including all the processes.

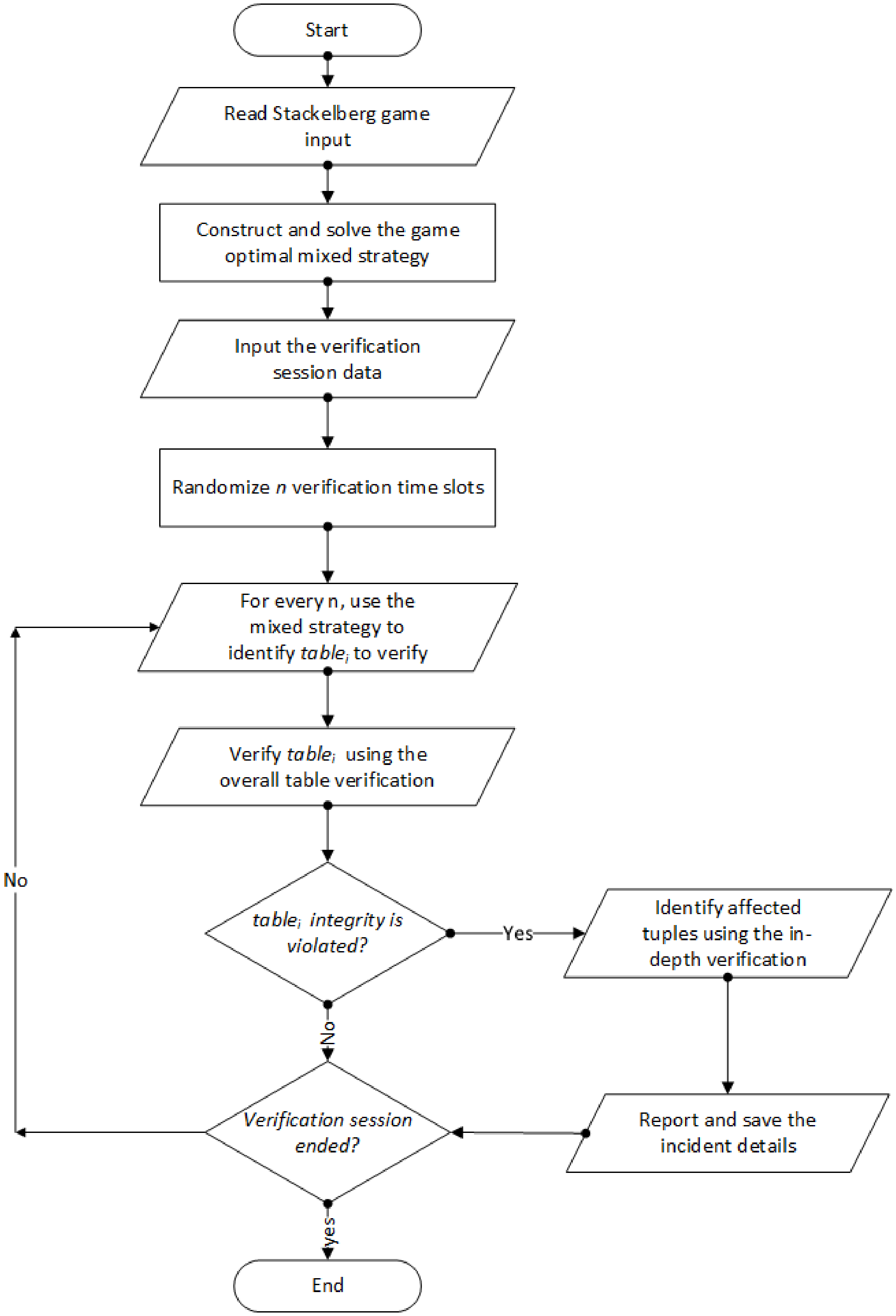

Start
Read Stackelberg game input
Construct and solve the game optimal mixed strategy
Input the verification session data
Randomize n verification time slots
For every n, use the mixed strategy to identify $table_i$ to verify
Verify $table_i$ using the overall table verification
$table_i$ integrity is violated?
Yes
Identify affected tuples using the in-depth verification
No
Report and save the incident details
Verification session ended?
No
yes
End

Figure 4.4: Detailed flow chart of the verifier component

4.2 Evaluation

In this section, we start by performing theocratic security analysis of the implemented system. Next, we evaluate the Stackelberg scheduling effectiveness by comparing our implemented system performance against a uniform randomization verification model. Finally, we evaluate and summarize the efficiency of our model against different cost metrics.

4.2.1 Security Analysis

The proposed model has to provide two security requirements. First, it has to ensure that data in the target database has not been tampered with. Second, it has to ensure that no data in the target database has been deleted in any an unauthorized way.

Remark 1. *The proposed model provides data integrity assurance*

Justification 1. *Since the proposed model uses a digital signature to sign the two different authentication structures used by the model, and assuming that the used digital signature is secure enough and that the private key is kept secure, no one other than the authorized parties (who have the private key) can alter the signed data without being detected. Assume that table x has the content m. Therefore, the table based authentication structure constructed by the proposed model will be $Sign(h(m))$. As a result, if table x content was altered to be m' where $m \neq m'$, the verification process will yield that $Decrypt(h(m)) \neq h(m')$. Thus, the verification will fail and the manipulation will be detected.*

Remark 2. *The proposed model provides data completeness assurance*

Justification 2. *Since the proposed model uses the table-based authentication structure, which signs the hash of concatenation of the whole table data, and assuming that the used digital signature is secure enough and that the private key is kept secure, any data deletion performed by any an unauthorized party will be detected when verifying the manipulated table. Assume that table x has the content m. Therefore, the table based authentication structure constructed by the proposed model will be $Sign(h(m))$. As a result, if some or all of table x content was deleted to be mz*

where $m \neq mz$, the verification process will yield that $Decrypt(h(m)) \neq h(m - z)$. Thus, the verification will fail and the manipulation will be detected.

4.2.2 Stackelberg Scheduling Evaluation

In addition to the security proof, we need to evaluate the Stackelberg verification scheduling effectiveness of the proposed model. In the literature, Stackelberg-based randomization is usually compared to the uniform randomization regardless of the fields the randomization is applied to, and since our proposed VDB model's design is considerably different than any VDB model design in the literature, we will compare the effectiveness of our model's Stackelberg-based randomization against a uniform randomization model. We have simulated a malicious database service provider and we ran the proposed model against a model that uses a uniform randomization. In these settings, we have set a database of 20 tables hosted at the simulated malicious database service provider side. Each table contains $50,000$ tuples. The simulated malicious database service provider each time manipulates some tables based on his optimization function. The different settings of each test and its average results are presented next.

Figure 4.5 shows the average number of verifications required by each model to detect a manipulation of one table at a time for 10 runs. The x-axis enumerates the manipulations performed by the malicious database service provider whereas the y-axis represents the average number of verifications needed for each model to detect the integrity violation. The results clearly show that our model outperforms the uniform randomization model. Average verifications performed ranged between 2 and 9 for our model whereas the uniform randomization model required from 6 to 32 verifications before detecting the integrity violation.

Figure 4.6 shows the average number of detections achieved by each model when limiting the number of verifications to 10. The x-axis enumerates the manipulations performed by the malicious database service provider while the y-axis represents the average number of detections recorded by each model. The results show that our model detected the integrity violation between 1 and 3 times whereas the uniform randomization detections ranged between 0 (in runs 1, 2, 5, 7, 8, and 10) and 1 detection.

Figure 4.7 shows the average number of detections achieved by each model when running both

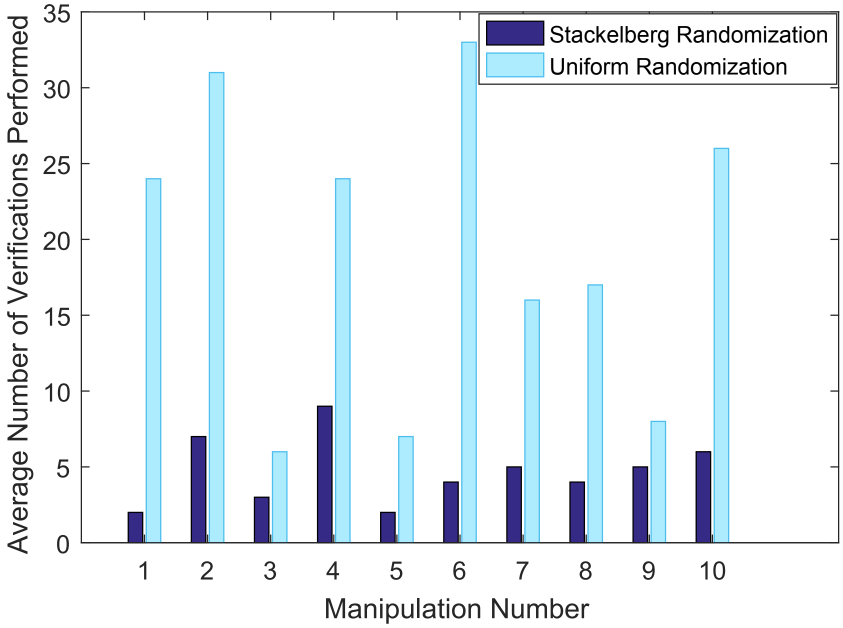


Figure 4.5: Averge number of verifications performed by each model

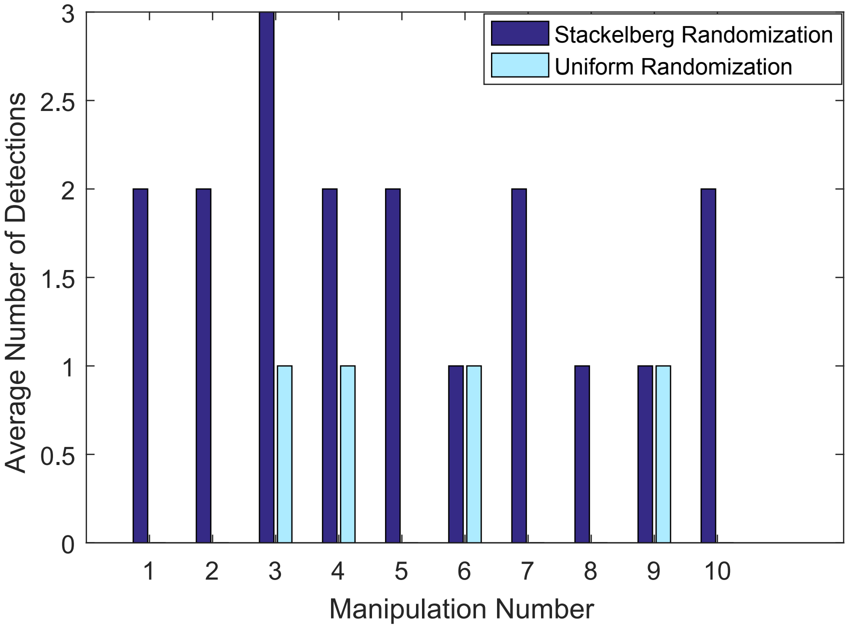


Figure 4.6: Average number of detections over 10 verifications

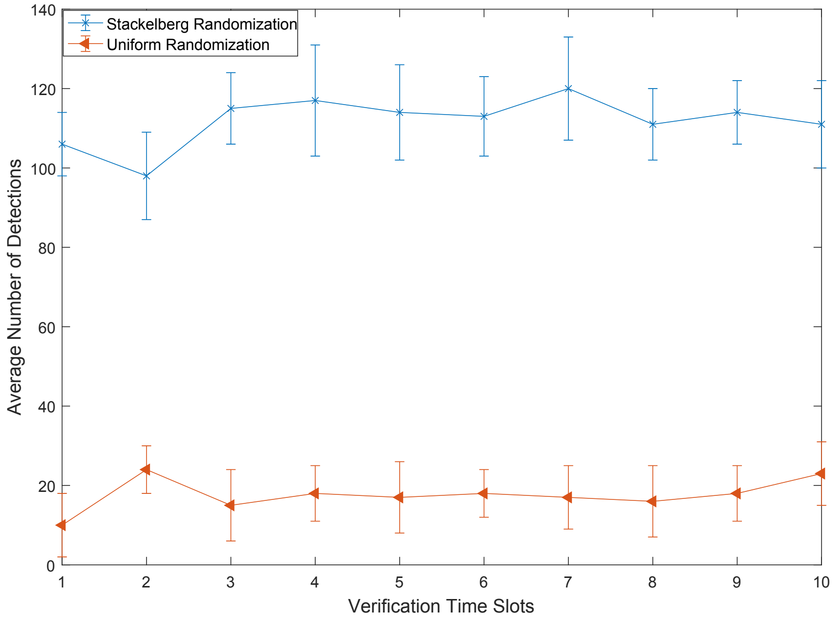


Figure 4.7: Average number of detections over 10 verification time slots

models with unlimited number of verifications for 10 time slots each of 30 seconds. The x-axis enumerates the time slots while the y-axis represents the average number of detections recorded by each model. The results show that the proposed model clearly outperformed the uniform randomization model with average number of detections amounted to six times greater than the uniform randomization model.

Figure 4.8 shows the average number of verifications performed by each model to detect all the manipulations when manipulating one or more tables. The x-axis represents the number of tables manipulated while the y-axis represents the average number of verifications performed before detecting all the manipulations. The results clearly show that our model took less verification attempts to detect the manipulations even when manipulating multiple tables at once.

Figure 4.9 shows the average time in seconds it took each model to detect all the manipulations when manipulating one or more tables. The x-axis represents the number of tables manipulated while the y-axis represents the average time elapsed before detecting all the manipulations by each model. The results clearly show that our model was faster to detect the manipulations even when

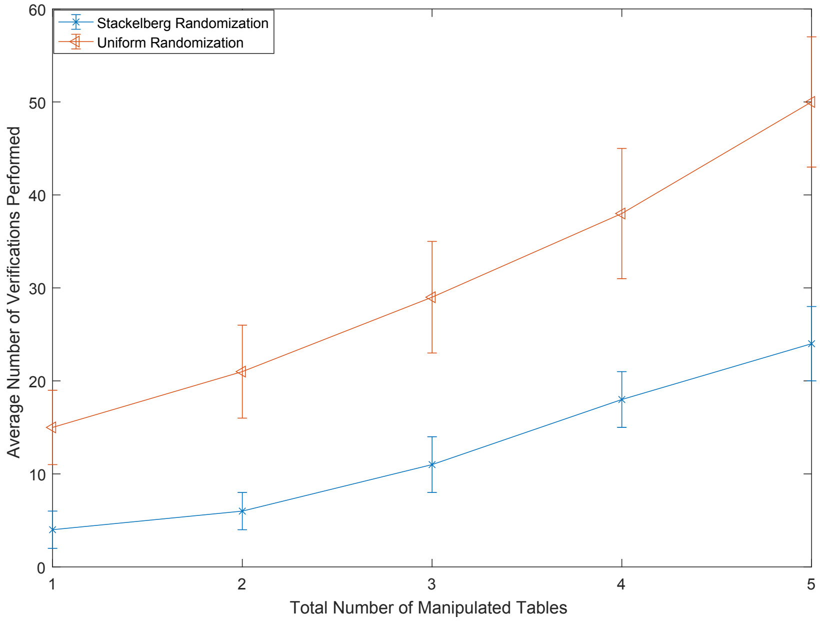


Figure 4.8: Average number of verifications required to detect multiple tables manipulations

manipulating multiple tables at once.

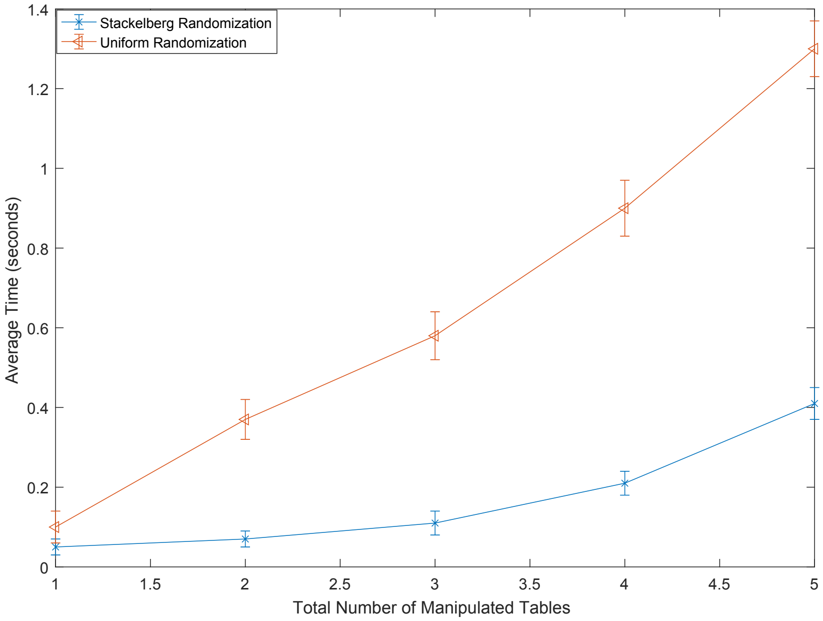


Figure 4.9: Average time required to detect multiple table manipulations

4.2.3 Efficiency Analysis

We first identify the cost of the different processes involved in the proposed model. Then we present an analysis of the processing The three main contributions of our proposed model greatly improve the efficiency of the VDB model. First, employing the periodic verifications saves a great deal of computation and communication overhead. Second, delegating the verification task responsibility to the verifier entity not only provides a logical separation of concerns, but also enables the proposed model to support all query types since database users are not involved in the process unlike most of the models in the literature. Third, the two stage verification process introduced in our model considerably reduces the required number of hash and digital signature operations. Thanks to the table-based authentication structure, the verifier needs not to activate the expensive in-depth verification stage except when detecting an integrity violation in a certain table using the overall-table verification stage.

Next, we first identify the cost of the different processes involved in the proposed model. Then we present an analysis of the processing and communication cost required at the side of each party involved for verifying both the integrity and completeness of the database.

There are two main processes in the proposed model. The first process is the creation of the authentication structure whereas the second process is the verification process. The processing cost of creating the table-based authentication structure is one hash operation and one signature operation per table. On the other hand, the processing cost of creating the tuple-based authentication structure is one hash operation and one signature operation per tuple. Similarly, the processing cost of the overall-table verification stage is one hash and one decryption (verifying the signature) per table. Likewise, the processing cost of the in-depth table verification stage is one hash and one decryption per each tuple in a table. The following is the processing and communication cost analysis incurred at each party involved in the proposed model:

- **Database owner:** for each data insertion process, the database owner has to perform one hash operation and one signature operation per each inserted tuple in order to create the tuple-based authentication structure and upload it along with the inserted tuples. In addition, the database owner incurs one communication to the database service provider and one signature operation in order to create the table-based authentication structure. The hash of the table-based authentication structure is performed at the database service provider side. Similarly, the update process involves one communication to the database service provider and one signature operation in order to update the affected tuples' authentication structure. Furthermore, the database owner incur one communication and one signature to update the table-based authentication structure. Finally, for each deletion operation, the database owner has to update only the table-based authentication structure, which involves one communication and one signature operation performed at the database owner side. Table 4.1 summarizes the costs incurred at the database owner side.

- **The database service provider:** the database service provider is involved in calculating the hash digest to be sent to the database owner for signing or to the verifier to be used in the verification process. These operations are performed particularly by the relational database management system using the transactional structured query language T-SQL. Furthermore,

Table 4.1: Computation and communication cost incurred at the database owner side

Process	Operation	Communication Cost	Hashing Cost	Signing Cost
Insertion	Tuple-based AS creation	0	$1/tuple$	$1/tuple$
	Table-based AS creation	1	0	$1/table$
Update	Tuple-based AS creation	1	0	$1/tuple$
	Table-based AS creation	1	0	$1/table$
Deletion	Tuple-based AS creation	0	0	0
	Table-based AS creation	1	0	$1/table$

the database service provider processes digital signature aggregation of multiple tuples signatures to reduce the communication and computation overhead when the in-depth verification is performed.

- **Database users**: Database users in the proposed model have no role in the verification process.

- **The verifier:** the overall-table verification stage involves one communication and one decryption operation per table. On the other hand, the in-depth verification stage involves one communication per table and one decryption operation per tuple for all tuples in a table. The hash operations are performed at the database service provider to reduce communication and computation overhead. Table 4.2 summarizes the costs incurred at the database owner side.

Table 4.2: Computation and communication cost incurred at the verifier side

Operation	Communication Cost	Decryption Cost
The over-all table verification	1	$1/table$
The in-depth table verification	1	$1/tuple$

Chapter 5

Conclusion and Future Work

5.1 Contributions

In this research, we have proposed a practical verifiable outsourced database model using Stackelberg game theory. To our knowledge, this is the first work that employs a game theoretic approach to address the integrity concerns of outsourced databases. The implementation of game theory in the problem of the verifiable database significantly improved the practicality and flexibility of the model. First, the Stackelberg game theoretic approach in our proposed model enabled effective periodic verifications scheduling to reduce the computation and communication costs. Second, it noticeably improved the verifiable database model design by providing a well separation of concerns implementation. To illustrate, the verification is performed by an independent entity called the verifier instead of it being performed by database users. This has improved the model's practicality and flexibility by supporting all query types including the sophisticated ones such as the aggregated queries. Finally, the security level provided by proposed model is improved by performing the verification process over the whole data instead of only verifying the queried data as was the case in the previous models in the literature.

5.2 Future Work

For future work, the verification scheduling effectiveness can be improved further by dynamically analyzing the previous verification sessions history using suitable machine learning algorithms in order to update the game input, which could result in optimally adapting the mixed strategy obtained for different unforeseen attacker types.

References

An, B., Tambe, M., Ordonez, F., Shieh, E. A., & Kiekintveld, C. (2011). Refinement of strong stackelberg equilibria in security games. In *Aaai*.

Chen, X., Li, J., Huang, X., Ma, J., & Lou, W. (2015). New publicly verifiable databases with efficient updates. *IEEE Transactions on Dependable and Secure Computing*, *12*(5), 546–556.

Chen, X., Li, J., Weng, J., Ma, J., & Lou, W. (2014). Verifiable computation over large database with incremental updates. In *European symposium on research in computer security* (pp. 148–162).

Devanbu, P., Gertz, M., Martel, C., & Stubblebine, S. G. (2002). Authentic third-party data publication. In *Data and application security* (pp. 101–112). Springer.

Goodrich, M. T., Tamassia, R., & Triandopoulos, N. (2008). Super-efficient verification of dynamic outsourced databases. In *Topics in cryptology–ct-rsa 2008* (pp. 407–424). Springer.

Korzhyk, D., Conitzer, V., & Parr, R. (2010). Complexity of computing optimal stackelberg strategies in security resource allocation games. In *Aaai*.

Li, F., Hadjieleftheriou, M., Kollios, G., & Reyzin, L. (2006). Dynamic authenticated index structures for outsourced databases. In *Proceedings of the 2006 acm sigmod international conference on management of data* (pp. 121–132).

Ma, D., Deng, R. H., Pang, H., & Zhou, J. (2005). Authenticating query results in data publishing. In *International conference on information and communications security* (pp. 376–388).

Merkle, R. C. (1989). A certified digital signature. In *Conference on the theory and application of cryptology* (pp. 218–238).

M'hamdi, M. A., & Bentahar, J. (2012). Scheduling reputation maintenance in agent-based communities using game theory. *Journal of Software*, *7*(7), 1514–1523. Retrieved from http://dx.doi.org/10.4304/jsw.7.7.1514-1523 doi: 10.4304/jsw.7.7.1514-1523

Mykletun, E., Narasimha, M., & Tsudik, G. (2003). Providing authentication and integrity in outsourced databases using merkle hash trees. *UCI-SCONCE Technical Report*.

Mykletun, E., Narasimha, M., & Tsudik, G. (2006). Authentication and integrity in outsourced databases. *ACM Transactions on Storage (TOS)*, *2*(2), 107–138.

Narasimha, M., & Tsudik, G. (2005). Dsac: integrity for outsourced databases with signature aggregation and chaining. In *Proceedings of the 14th acm international conference on information and knowledge management* (pp. 235–236).

Narasimha, M., & Tsudik, G. (2006). Authentication of outsourced databases using signature aggregation and chaining. In *International conference on database systems for advanced applications* (pp. 420–436).

Osborne, M. J., & Rubinstein, A. (1994). *A course in game theory*. MIT press.

Pang, H., Zhang, J., & Mouratidis, K. (2009). Scalable verification for outsourced dynamic databases. *Proceedings of the VLDB Endowment*, *2*(1), 802–813.

Papamanthou, C., & Tamassia, R. (2007). Time and space efficient algorithms for two-party authenticated data structures. In *International conference on information and communications security* (pp. 1–15).

Paruchuri, P., Pearce, J. P., Marecki, J., Tambe, M., Ordonez, F., & Kraus, S. (2008a). Efficient algorithms to solve bayesian stackelberg games for security applications. In *Proc. of aaai* (pp. 1559–1562).

Paruchuri, P., Pearce, J. P., Marecki, J., Tambe, M., Ordonez, F., & Kraus, S. (2008b). Playing games for security: an efficient exact algorithm for solving bayesian stackelberg games. In *Proceedings of the 7th international joint conference on autonomous agents and multiagent systems-volume 2* (pp. 895–902).

Pita, J., Jain, M., Marecki, J., Ordóñez, F., Portway, C., Tambe, M., … Kraus, S. (2008). Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the 7th international joint conference on autonomous* *agents and multiagent systems: industrial track* (pp. 125–132).

Pointcheval, D., & Stern, J. (2000). Security arguments for digital signatures and blind signatures. *Journal of cryptology*, *13*(3), 361–396.

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21*(2), 120–126.

Rogaway, P., & Shrimpton, T. (2004). Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *International workshop on fast software encryption* (pp. 371–388).

Thompson, B., Haber, S., Horne, W. G., Sander, T., & Yao, D. (2009). Privacy-preserving computation and verification of aggregate queries on outsourced databases. In *International symposium on privacy enhancing technologies symposium* (pp. 185–201).

Von Stackelberg, H. (1934). *Marktform und gleichgewicht*. J. springer.

Wahab, O. A., Bentahar, J., Otrok, H., & Mourad, A. (2016). A stackelberg game for distributed formation of business-driven services communities. *Expert Systems with Applications*, *45*, 359–372.

Wang, J., Chen, X., Huang, X., You, I., & Xiang, Y. (2015). Verifiable auditing for outsourced database in cloud computing. *IEEE Transactions on Computers*, *64*(11), 3293–3303.

Xie, M., Wang, H., Yin, J., & Meng, X. (2007). Integrity auditing of outsourced data. In *Proceedings of the 33rd international conference on very large data bases* (pp. 782–793).

Yang, Y., Papadias, D., Papadopoulos, S., & Kalnis, P. (2009). Authenticated join processing in outsourced databases. In *Proceedings of the 2009 acm sigmod international conference on management of data* (pp. 5–18).

Yin, Z., Korzhyk, D., Kiekintveld, C., Conitzer, V., & Tambe, M. (2010). Stackelberg vs. nash in security games: Interchangeability, equivalence, and uniqueness. In *Proceedings of the 9th international conference on autonomous agents and multiagent systems: volume 1-volume 1* (pp. 1139–1146).

Yuan, J., & Yu, S. (2013). Flexible and publicly verifiable aggregation query for outsourced databases in cloud. In *Communications and network security (cns), 2013 ieee conference on* (pp. 520–524).

Zhang, L. F., & Safavi-Naini, R. (2014). Verifiable delegation of computations with storage-verification trade-off. In *European symposium on research in computer security* (pp. 112–129).

Zhu, Y., Ahn, G.-J., Hu, H., Yau, S. S., An, H. G., & Hu, C.-J. (2013). Dynamic audit services for outsourced storages in clouds. *IEEE Transactions on Services Computing*, *6*(2), 227–238.